Technische Universität München

TUM School of Computation, Information and Technology

# Public Key Infrastructures and Blockchain Systems

Utilizing Internet Public Key Infrastructures to Leverage Their Trust and Adoption in Blockchain Systems

Ulrich Simon Stefan Gallersdörfer, M.Sc.

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigten Dissertation.

| | | |
|---|---|---|
| Vorsitz: | | Prof. Dr.-Ing. Georg Carle |
| Prüfer der Dissertation: | 1. | Prof. Dr. Florian Matthes |
| | 2. | apl. Prof. Dr. Georg Groh |

Die Dissertation wurde am 24.04.2023 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 16.10.2023 angenommen.

# Acknowledgements

First and foremost, I want to thank my advisor Prof. Dr. Florian Matthes, for allowing me to conduct research under his guidance on blockchain technologies. I greatly appreciated the discussions, research, and education as a Ph.D. student. It fills me with great pleasure to witness the continued growth of blockchain research at the chair.

I want to thank the team at the *Digital Credentials Consortium* and *Open Learning* at *Massachusetts Institute of Technology* for inviting me to Cambridge and fostering a fantastic collaboration during my time there. Namely, I want to thank J. Philipp Schmidt, Kim Duffy, Brandon Muramatsu, Dmitri Zagidulin, Gillian Walsh, Krishna Rajagopal, and Sanjay E. Sarma.

So many great colleagues accompanied me at the chair. Special thanks go to Elena Scepankova for our discussions that inspired me to develop the idea for this dissertation. Further, thanks to colleagues that directly or indirectly influenced my work: Patrick Holl, Felix Hoops, Oleksandra Klymenko, Nektarios Machner, Sascha Nägele, Burak Öz, and Dr. Bernhard Waltl.

I am grateful for all the students I had the opportunity to advise in their theses and those who collaborated with me on various projects. Special thanks go to Jonas Ebel, Friederike Groschupp, Pascal Herrmann, Felix Hoops, Kilian Käslin, Tuan Anh Ma, Metodi Manov, Jan-Niklas Strugala, and David Stübing.

Thanks to Lena Klaaßen and Christian Stoll for our outstanding collaboration. Thanks to the entire team at CCRI for having my back during the busy times.

Finally, I want to thank my family and friends for their unwavering support: My parents, Andrea and Andreas. My siblings, Evi, Felix, and Jonas. Thank you for your support, encouragement, and love. I am extending my appreciation to Michi and Franzi. I couldn't have asked for better friends.

Munich, April 24, 2023
Ulrich Gallersdörfer

# Abstract

Blockchain networks provide access to novel applications for end-users and enterprises alike. Regardless of these networks' trustless design, entities accessing them are confronted primarily with the absence of information about their respective counterparties. This deficiency seems counterintuitive, as users of blockchain networks rely mainly on an information structure domiciled in the World Wide Web (WWW). Underlying protocols have solved the authentication issues of counterparties and enabled the Internet to achieve its current socioeconomic importance. However, these protocols are not facilitated in a blockchain context.

In this thesis, our main research goal is to analyze, conceptualize, design, implement, and evaluate the utilization of Internet Public Key Infrastructures (PKI), with the aim of leveraging their trust and adoption in blockchain systems. While several approaches and applications are in place to mitigate counterparty authentication issues, they fail to consider the interplay between WWW and blockchain networks or lack generalizability for other forms of PKI or environments.

This thesis introduces a framework for using cryptographic key material managed by PKI in a blockchain-based context and establishing identity assertions and authentication of on-chain identifiers. To this end, the framework relies on the technical foundations, requirements, and security considerations defined in the protocols that form the foundation of the Internet, namely Transport Layer Security (TLS), Domain Name System (DNS), and the certificate standard X.509. The framework supports two forms of authentication: a) off-chain authentication, verifying the correctness of identity assertions in a user-specific context, and b) on-chain authentication, allowing a network-wide consensus about the authenticity of identity assertions. The verification processes leverage systems such as Certificate Transparency (CT) to detect and prevent illicit behavior. The framework lays the foundation for a broad set of applications that contribute to a trustworthy blockchain ecosystem.

We classify the applicability and assess the performance of the framework. First, we evaluate the relevance and practicability of both forms of authentication. Second, we include Ethereum Name Service in our analysis, comparing the costs, security, adoption, and fulfillment of the initially posed requirements.

# Zusammenfassung

Blockchain-Netzwerke bieten Nutzern und Unternehmen gleichermaßen Zugang zu neuartigen Anwendungen. Trotz der Sicherheit dieser Technologien werden Entitäten, die auf diese Netze zugreifen, Informationen über ihre Gegenparteien vorenthalten. Dies erscheint widersprüchlich, da sich die Nutzer von Blockchain Netzwerken weitgehend auf Informationen verlassen, die aus dem World Wide Web (WWW) stammen. Die zugrundeliegenden Protokolle lösten Probleme der Authentifizierung und ermöglichten es dem Internet, seine heutige sozioökonomische Bedeutung zu erlangen. Dennoch werden sie in einem Blockchain-Kontext nicht genutzt.

Diese Arbeit hat die Analyse, Konzeption, Gestaltung, Implementierung und Evaluierung der Nutzung von Public Key Infrastrukturen (PKI) des Internets zum Ziel, um deren Vertrauen und Akzeptanz in Blockchain Systemen zu nutzen. Es existieren zwar mehrere Ansätze und Anwendungen, um Probleme bei der Authentifizierung der Gegenpartei zu mitigieren, doch sie berücksichtigen nicht das Zusammenspiel zwischen dem WWW und Blockchain-Netzwerken oder lassen sich nicht für andere Formen von PKI oder Systeme verallgemeinern.

In dieser Arbeit wird ein Rahmenwerk für die Verwendung von kryptografischem Schlüsseln, die von PKI verwaltet werden, in einem Blockchain-basierten Kontext und zur Erstellung von Identitätsbestätigungen und Authentifizierung von On-Chain Adressen eingeführt. Dabei stützt sich das Framework auf die technischen Grundlagen und Anforderungen, die in den jeweiligen Protokollen definiert sind, wie z.B. Transport Layer Security (TLS), Domain Name System (DNS) und der Zertifikatsstandard X.509. Das Framework unterstützt zwei Formen der Authentifizierung: a) Off-Chain, die die Korrektheit von Identitätsbehauptungen in einem benutzerspezifischen Kontext verifiziert, und b) On-Chain, die es ermöglicht, einen Netzwerk-weiten Konsens über die Authentizität von Identitätsbehauptungen zu bilden. Die Verifikationsprozesse nutzen Systeme wie Certificate Transparency (CA), um unerlaubtes Verhalten zu erkennen und zu verhindern. Das Framework bildet die Grundlage für eine breite Basis von Anwendungen, die zu einem vertrauenswürdigen Blockchain-Ökosystem beitragen.

Wir analysieren die Anwendbarkeit sowie die Performance des Frameworks. Zunächst evaluieren wir die Relevanz und Praktikabilität beider Formen der Authentifizierung. Im Anschluss inkludieren wir Ethereum Name Service in unsere Analyse und vergleichen Kosten, Sicherheit, Adaption und die Erfüllung initial definierter Anforderungen.

# Contents

# List of Tables

# List of Figures

# Listings

# List of Abbreviations

| | |
|---|---|
| CA | Certificate Authority |
| CRL | Certificate Revocation List |
| CT | Certificate Transparency |
| DID | Decentralized Identifier |
| DLT | Distributed Ledger Technology |
| DNS | Domain Name System |
| DNSSEC | DNS Security Extensions |
| ENS | Ethereum Name Service |
| FQDN | Fully Qualified Domain Name |
| GWei | Gigawei (0.000000001 ether) |
| ICO | Initial Coin Offering |
| OCSP | Online Certificate Status Protocol |
| PKI | Public Key Infrastructure |
| RFC | Request for Comments |
| SSL | Secure Sockets Layer |
| TLS | Transport Layer Security |

# Chapter 1.

# Introduction

## 1.1. Motivation

Blockchain technology serves as the basis for Web 3.0, cryptocurrencies, and the Metaverse. By enabling near-instant transactions, trustless data storage, and tokenization of digital and real world assets, blockchain networks enable novel use cases and business models beyond traditional Web 2.0 enterprises (Blocher et al., 2019). Gartner estimates that blockchain adds over USD 3.1 trillion in new business value and, thus, will serve as the backbone of the financial industry by 2030 (Lovelock et al., 2017).

As a first use case, blockchains serve as **simple payment layers** (Nakamoto, 2008). Independent parties exchange monetary values via pseudonymous, self-generated addresses. These addresses consist, similarly to other payment methods, such as IBAN, of random characters and numbers, making it hard to identify the counterparty and ensure the correctness of the address. While users are responsible for ensuring that they are using the correct addresses, tricking users into sending funds to malicious addresses has been a significant problem in the past:

The CoinDash ICO (Initial Coin Offering) intended to collect investments from users (Wieczner, 2017). However, when users sent their funds to the address that was displayed on the website of CoinDash, the money wound up in a hacker's wallet that silently swapped the correct address with the malicious address, netting 43,000 ethers (about USD 12.9 million at that time). We give more insights into the CoinDash ICO hack in Section 4.1.

With the introduction of **smart contracts** in Ethereum in 2015, blockchain networks became more versatile and enabled more elaborate use cases (Wood et al., 2014). As the underlying programming languages are Turing-complete, any business model can be deployed on the networks and leveraged for different purposes. As these smart contracts

rely on the same address format, again, users are confronted with a similar issue: Is the address of the contract authentic, and does it belong to an entity that I trust?

Assume universities use public blockchains to publish integrity information about the credentials they issue. Every participating university has its smart contract that enables the submission and revocation of cryptographically verifiable information. An employer receives from a prospective employee a credential, which is also recorded in a smart contract. The address of the smart contract is displayed on the credential. While the smart contract exists and stores the cryptographic information in question, it remains unclear whether it is owned and operated by the university or an illicit entity, as the employer cannot verify the relationship between the university and the smart contract.

From these two examples, it becomes apparent that determining the authenticity of addresses is of utmost importance to identify persons, Internet of Things (IoT) devices, companies, or institutions in order to interact or conduct business with them.

Blockchain networks today face similar issues as the World Wide Web (WWW) in its early days: How do users find and recognize entities they want to interact with? Similar to Internet Protocol (IP) addresses, blockchain addresses are hard to memorize and easy to misspell (Glomann et al., 2020). Today, more than 600 million domain names exist (Domain Name Stat, 2023), and billions of people use them daily (Ritchie et al., 2023). For blockchains, a similar service called Ethereum Name Service (ENS) was proposed in 2017 (Ethereum Name Service, 2023c). ENS allows users to claim a domain name with the Top Level Domain (TLD) ending `.eth`. While the service works as intended, it introduced two new problems (Xia et al., 2022):

- **Bootstrapping:** ENS initially had no user base and no recognition. Users and institutions faced a "chicken-or-egg" problem, as the lack of adoption did not incentivize the purchase of domains in the first place. Finding adoption from users, companies, and service providers alike remains a key challenge for any naming service.

- **Domain Squatting:** ENS, as a fully decentralized system, has no way of regaining ownership of once-sold domains. In contrast to DNS, domains lost in ENS cannot be transferred to their rightful owner after being claimed by another entity. For example, one entity has registered and owns `google.eth`, `mcdonalds.eth`, and `redbull.eth`, hinting at domain squatting (Xia et al., 2022).

While ENS solves concerns of recognizing the counterparty in blockchain networks, it introduces these two novel problems. To solve the underlying issues without introducing

new obstacles, we want to leverage the traditional domain landscape including its well-established naming scheme in blockchain networks and map domain names to blockchain addresses.

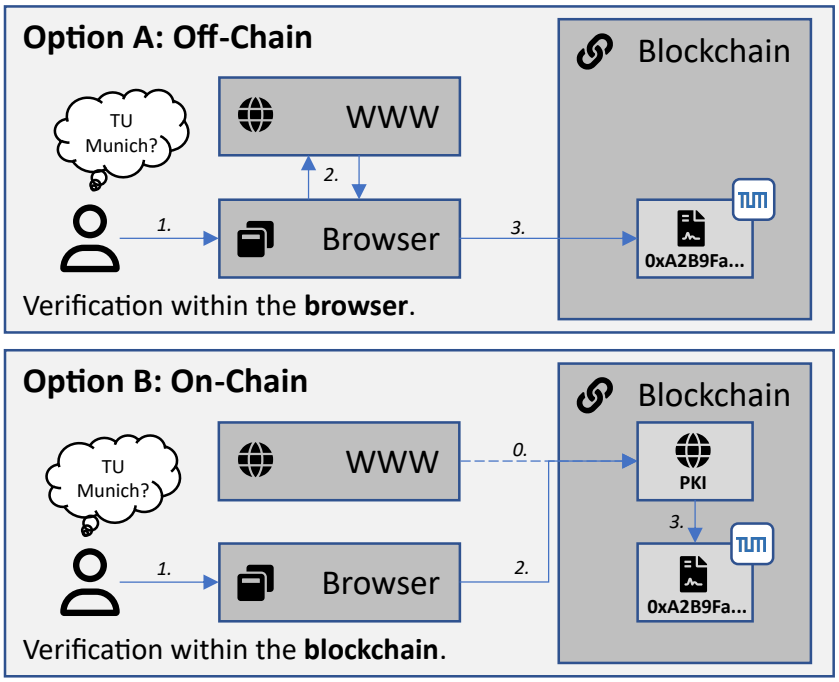



**Figure 1.1.:** Two approaches for verifying a domain-address relationship. The user instructs the browser to access the smart contract of TU Munich in step 1. In option A, the browser collects all necessary information from the WWW in step 2 to verify the authenticity and connect to the smart contract in step 3. In option B, parts of the WWW are first mirrored and verified to the blockchain network in step 0. The browser directly accesses the mirrored part and accesses the previously verified smart contract. A detailed explanation of how domain names can be leveraged within blockchain networks can be found in Chapter 3.

More technically, we identify existing hierarchical Public Key Infrastructures (PKI) as a key to introducing domain names in blockchain networks. We aim to a) solve the current issues with hard-to-memorize blockchain addresses and, simultaneously, b) avoid the need for bootstrapping and enable already widely accepted domain names to be used in blockchain networks. We identify two approaches for resolving and verifying the domain-address binding: off-chain in the browser and on-chain in the respective network. A simplified depiction of both approaches is displayed in Figure 1.1. We describe and evaluate these approaches and compare them with existing applications that rely on comparable infrastructures.

## 1.2. Research Questions

The utilization of WWW technologies in a blockchain-based context poses several challenges. We investigate potential and existing architectures and system designs and evaluate their effectiveness. We aim to answer the following research questions.

### RQ1: To what extent can one leverage domain names and related Public Key Infrastructures in a blockchain environment?

**RQ 1.1** *In which ways have domain names and related PKIs been utilized in a blockchain context?*

Preliminary insights into cryptocurrency naming services suggest that research into leveraging parts of PKIs of the WWW is available. To understand potential approaches and limitations, we analyze the current state of research in the area of PKIs and blockchain networks. We address this question in Chapter 2.

**RQ 1.2** *What are the technical pathways to enable domain name usage in a blockchain context?*

There are several ways to leverage the underlying PKI of domain names, their structures, and their data within blockchain networks. We propose dimensions in which PKIs can be utilized, and we identify relevant configurations. In Chapter 3, we discuss relevant technical pathways.

Following our initial insights of RQ1, we identify two pathways for enabling domain name usage in blockchain networks. First, *verifying the relationship between domain and address in a local context*, which we address in RQ2. Second, *verifying the relationship between domain and address in an on-chain context*, which we address in RQ3.

### RQ2: What are ways to verify the connection between a domain name and a blockchain address in a local context?

**RQ 2.1** *How can cryptographically verifiable assertions between blockchain addresses and domain names be made?*

The binding between a domain name certificate and an address within a blockchain system must be robust and cryptographically verifiable. We propose cryptographically verifiable assertions in Section 3.4.

**RQ 2.2** *How does the life cycle of cryptographic key material influence the system's functionality?*
Cryptographic key material used within both the certificates and assertions might expire or become invalid for other reasons. We need to properly consider these processes within our systems. In Section 4.2, we investigate the influence of the cryptographic key material.

**RQ 2.3** *What are the limitations of verifying assertions in a local context?*
The specific context of a browser and the availability of information can influence functionality, usability, and security assessments. We evaluate these limitations in Sections 4.3 and 4.4.

The results from RQ2 suggest that an off-chain verification approach comes with considerable downsides. These downsides lead us to consider a revised design that conducts verification in an on-chain context.

## RQ3: How can we verify the connection between a domain name and a blockchain address in an on-chain context?

**RQ 3.1** *What are the technical pathways for enabling the usage of certificates and endorsements in an on-chain context?*
In contrast to an off-chain verification approach, verification that takes place on-chain has more complexities. We describe and explain ways to enable such verification in Chapter 5.

**RQ 3.2** *How does the life cycle of cryptographic key material influence the functionality of the proposed system?*
In contrast to the off-chain verification approach, in which secondary information can be directly accessed, this information is not available in an on-chain context. We discuss these implications in Section 5.2.

**RQ 3.3** *What are the limitations of verifying assertions in an on-chain context?*
The on-chain context provides more transparency to all entities but also requires an extensive analysis and understanding of risks and attack scenarios, as the blockchain context differs from adversarial scenarios in TLS. We provide rationale in Section 5.4.

**RQ 3.4** *How can the interactions of users of Ethereum Name Service (ENS) regarding DNS Security Extensions (DNSSEC) be analyzed?*
ENS has utilized DNSSEC in its systems since August 2021. We want to understand how many domains ENS has issued since its inception and how the data can be obtained. We describe a process for data collection, cleanup, enrichment, and verification which we describe in Section 5.5.

**RQ 3.5** *How do existing approaches for leveraging domain names in a blockchain context compare to each other?*
Given that ENS leverages DNSSEC to fulfill similar goals, it is desirable to compare costs, security, adoption, and fulfillment of requirements. The evaluation is conducted in Chapter 6.

## 1.3. Contributions

This dissertation contributes to the state of research in information exchange between Web 2.0 and blockchain networks by leveraging PKI information in the context of blockchain networks. We enable secure linkage and references from the WWW to blockchain-specific entities and addresses, prevent or detect specific fraudulent behavior, enable data authentication and provenance within blockchain networks, and enable authentication of smart contract applications based on PKI certificate information. Approaches described in this dissertation partly eliminate bootstrapping issues and reduce onboarding costs for entities rooted in Web 2.0. Further, we develop a methodology for extracting information of DNSSEC-enabled ENS domains and analyzing their current state.

More precisely, we contribute to this field in the following ways:

- **analyzing the state of research** for leveraging PKI information within blockchain networks,
- **understanding and evaluating dimensions** and their arrangements to enable the **usage of certificate information** in a blockchain context,
- enabling a **generalized framework** for **utilizing certificate information** in both an on-chain and off-chain context,
- **analyzing security implications** and strategies for both the underlying cryptographic key material of the PKI and the additional signature data,

- **developing and analyzing** a dataset of all domains that have been bridged to **ENS using DNSSEC**,
- **analyzing the costs and limitations** of the described approaches including a comparison to the **DNSSEC approach of ENS**, and
- **outlining future work** and **improvements** of the approaches described within this dissertation.

## 1.4. Outline

This dissertation is structured in seven chapters and includes six appendices. An overview of the structure is given in Figure 1.2.

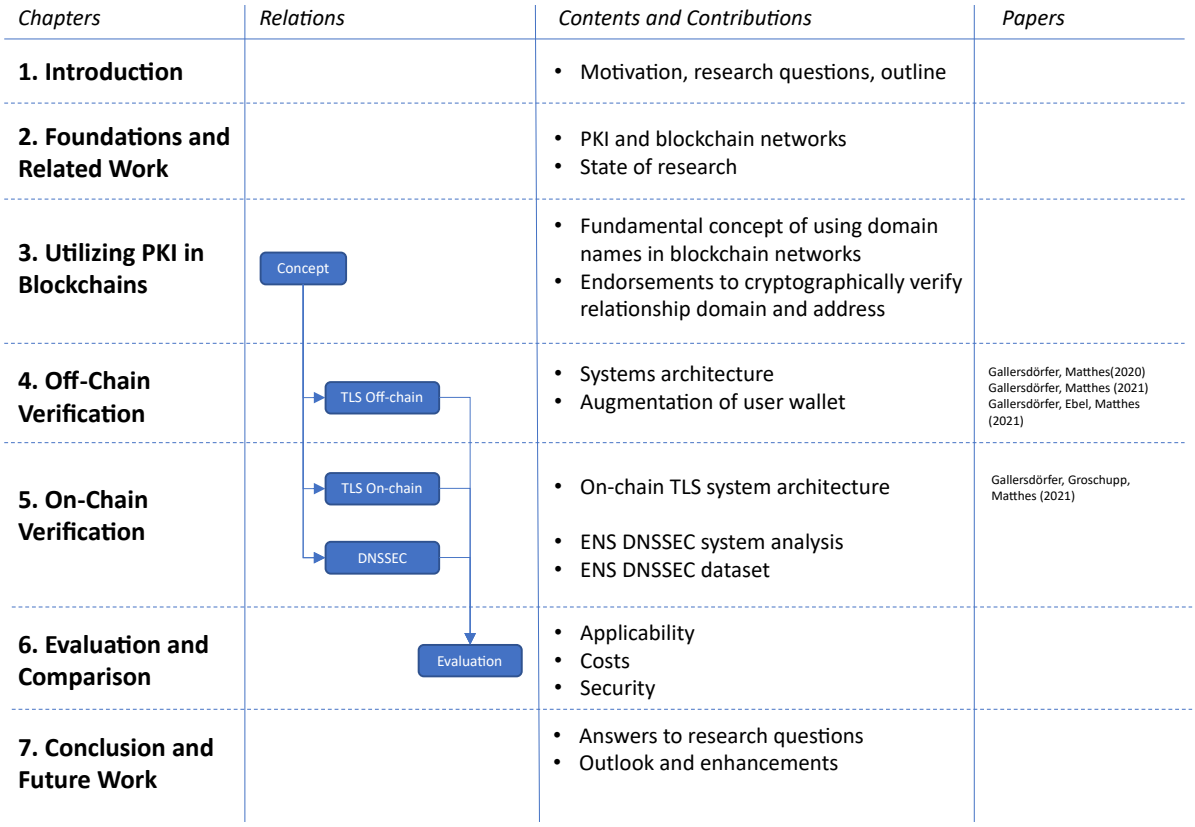



**Figure 1.2.:** Structure of this dissertation including chapters and contents.

In the following, we describe the chapters and their contents. Further, we highlight relevant related work and prior publications, including additional contributions we made within this dissertation.

Chapter 2, "**Foundations and Related Work**", introduces key concepts of the underlying technologies that form the basis of this thesis, such as PKI and TLS, including X.509 certificates and other security protocols such as CT and DNSSEC. The target system, blockchain networks, is also briefly explained, including concepts such as addresses and smart contracts. In this chapter, we also provide an overview of the state of the research on enabling the usage of certificate information within blockchain networks.

In Chapter 3, "**Utilizing Public Key Infrastructures in Blockchains**", we provide a detailed analysis of the core concept and approach of enabling information managed in PKIs within blockchain environments. This analysis includes an insight into the technical dimensions of such systems. It discusses which approaches are helpful in the context of the problem statements introduced earlier, proposing a solution space for leveraging certificate information in a blockchain context. We introduce the endorsement as an essential building block, an idea initially introduced in (Gallersdörfer and Matthes, 2020, 2021a). We contribute by defining the structure and formalizing all potential approaches to leverage PKI information within blockchain environments. Chapter 4, "**Off-Chain Verification**", introduces and describes the first approach, namely the off-chain TLS verification of endorsements stored in a blockchain network. More particularly, the underlying problem statement is described and analyzed. The second step establishes the system architecture, components, and processes. A threat model is developed, and the security implications of additional signatures in TLS are assessed. This concept was first introduced in (Gallersdörfer and Matthes, 2020, 2021a). Further, we describe the augmentation of a user wallet to support the system, which was initially developed and described in (Ebel, 2021) and (Gallersdörfer et al., 2021a). We contribute to this augmentation by revisiting TLS warnings in browsers to validate the underlying research observations, sharing insights into the updated results, and discussing and evaluating limitations and mitigation strategies.

Chapter 5, "**On-Chain Verification**", highlights two alternative system designs for leveraging TLS certificate information in the context of blockchains. On-chain verification enables a blockchain-wide consensus on the validity of endorsements, enabling novel use cases. First, we describe the problem statements that necessitate the on-chain authentication of domain names. Then, we describe the architecture and properties of the approach initially proposed in (Groschupp, 2020) and (Gallersdörfer et al., 2021c). We introduce the architecture design of ENS' approach to leverage DNSSEC and provide a rationale for both system architecture designs. We further contribute by developing a methodology to extract and analyze the DNSSEC-related domains transferred to ENS.

Further, in Chapter 6, "**Evaluation and Comparison of Approaches**", we discuss and evaluate the three approaches described in Chapters 4 and 5. We verify the suitability of the underlying ecosystems of TLS certificates and DNS Security Extensions. We discuss and evaluate the applicability and practicability of all systems. Further, we give insights into the adoption of ENS. Also, we give an overview of the costs of all approaches, discuss their security implications and analyze the fulfillment of the requirements stated in Chapter 3.

In Chapter 7, "**Conclusion and Future Work**", we summarize and conclude our work by answering all research questions posed in Section 1.2. Lastly, we suggest potential avenues for future research into leveraging certificates managed within PKIs in a blockchain-based context, including security enhancements, cost reductions, and trust anchor removal of the PKI system.

## 1.5. Prior Publications and Citation Style

Prior publications exist for this dissertation. Further, the author advised students completing theses partly related to this area of research. We display all publications, theses, and semester work relevant to this research in Appendix A. All theses related to this research project are highlighted and cited in the appropriate passages.

In the manuscript, we directly mark text cited from prior publications by the following methods:

- enclosing text in quotation marks ("This is a quote"),
- indenting the paragraphs in question,
- marking when we removed text from the quoted paragraphs ([...]),
- highlighting additions to the text ([*This is an addition*]), and
- providing a citation after the quotation
  – "TeSC: TLS/SSL-Certificate Endorsed Smart Contracts" (Gallersdörfer and Matthes, 2021a).

Changes that are cosmetic (e.g., creating bullet points out of enumerations) or that enhance the readability (e.g., spelling errors or updated references and citations) of quotations without changing their meaning are **not highlighted**. Please note that quotations can span multiple sections. Footnotes referenced from within quotes are also part of the individual quotations unless marked otherwise. We standardized terms for enhanced readability.

# Chapter 2.

# Foundations and Related Work

This chapter explains, discusses, and categorizes the underlying technologies and systems that enable our conceptualization, system architecture, and implementation for leveraging Public Key Infrastructures in a blockchain environment. It provides an overview of the related state-of-the-art research, differentiates existing research from our own, and categorizes it within the existing landscape.

In more detail this chapter is separated into the following sections:

- Section 2.1, *"Public Key Infrastructures in the Internet"*, introduces the systems constituting the World Wide Web including the Domain Name System, Transport Layer Security, X.509 certificates, Certificate Transparency, and DNS Security Extensions. We highlight the concepts of these systems that enable our system architecture.

- In Section 2.2, *"Blockchain Networks"*, we outline the concepts of the blockchain and the target network on which our systems are deployed. We briefly cover the core concept of the technology and underline the importance of smart contracts within these networks, i.e., trustless, decentralized applications.

- In Section 2.3, *"State of the Research"*, we provide an overview of related work and research that targets the intersection of Public Key Infrastructures and blockchain networks by either a) proposing novel system designs that overcome issues of PKIs with blockchain-based architectures or b) leveraging existing PKIs in a blockchain-based context. Furthermore, we highlight the research gap in the existing literature.

## 2.1. Public Key Infrastructures in the Internet

The inner workings of the Internet and World Wide Web as they exist today are managed by several governing bodies; the Internet Engineering Task Force (IETF), the Internet Corporation for Assigned Names and Numbers (ICANN), and the Internet Assigned Numbers Authority (IANA), among others. These entities are organized within the Internet Society (ISOC), a non-governmental organization that "*supports and promotes the development of the Internet as a global technical infrastructure [...]*" (Internet Society, 2022b). These entities are responsible for specific areas of the systems, their contents, and their rules and obligations. For example, the IETF comprises many working groups that work on and propose technical specifications, recommendations, and standards in the form of so-called Request for Comments (RFCs). On the other hand, the ICANN coordinates and manages the allocation of domain names in the DNS, the IANA, among other entities, is responsible for the assignment of IP addresses (IP version 4 and IP version 6 addresses) (IANA, 2022).

The Internet, its involved standards, and its concrete implementations serve as an infrastructure for digital services. For example, the WWW makes use of this underlying infrastructure (such as the Hypertext Transport Protocol [HTTP], TLS, and DNS) to enable an information service. Anyone can use browsers and access websites whose contents are described by the Hypertext Markup Language (HTML), which was also proposed in its earlier versions within the IETF as RFC 1866 (Berners-Lee and Connolly, 1995). Later versions of HTML, such as HTML5, were proposed by working groups of the World Wide Web Consortium (W3C) (Hickson and Hyatt, 2011).

Our work relies on selected core technologies; therefore, they are crucial for the remainder of this thesis. In the following, we cover the **DNS**, allowing the mapping of human-readable names to IP addresses, and **TLS** including **X.509 certificates** that enable secure authentication and communication between two entities over the Internet. Technologies such as **Certificate Transparency (CT)** and **DNSSEC** are either used in related work or pose promising avenues for future work. On their current trajectory, they are gaining increasing relevance within the Internet infrastructure ecosystem (Internet Society, 2022a).

### 2.1.1. Domain Name System

The DNS serves as one of the Internet's core protocols as a hierarchically organized directory service. For example, it enables the usage of the WWW as it is known

today, allowing people to utilize easy-to-memorize names to visit websites. In particular, DNS helps to resolve human-readable names in the form of a fully qualified domain name (FQDN) to IP addresses. Entities interested in connecting to the server behind a specific FQDN such as *https://www.example.org* can do so, independent of the intended underlying protocol (i.e., whether the user wants to visit the web server via `HTTPS`, or open a shell connection via `SSH`). DNS was initially described in RFC 882 and 883 and updated in RFC 1034 and 1035 (Mockapetris, 1983a,b, 1987a,b). We describe specific extensions to DNS that are noteworthy for this thesis in Section 2.1.3.

Core to the DNS is the specification of domain names. Figure 2.1 depicts the specific parts of an FQDN, namely the protocol, sub-domains, domain names, and top-level-domain (TLD). The last dot (".") is part of an absolute fully qualified domain name as specified in the respective standard, but is omitted in practice.

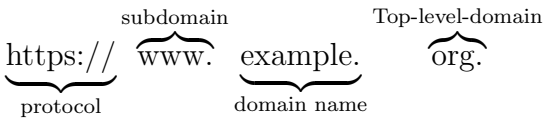



**Figure 2.1.:** Structure of a fully qualified domain name. The protocol (*https://*) is not part of an FQDN, but is shown for completeness.

DNS is hierarchically structured but decentrally organized. The name server responsible for the root zone in the form of the "." forwards requests for specific TLDs to the respective name servers, which in turn answer or forward requests to the name servers responsible for the individual domain names. Entities owning a domain (e.g., *example.org*) can run their name server, defining IP addresses and further information for the respective subdomains (e.g., *sd1.example.org* and *sd2.example.org*). This approach results in efficient request processes. Formal definitions and standards for the inner workings of the request process, zone files, and resolutions exist in the respective RFCs and are not reiterated within this thesis (Mockapetris, 1983a,b, 1987a,b).

The relevance of the DNS not only for the Internet but for society as a whole is undisputed. Given the worldwide recognition and acceptance of DNS, most forms of news, communication, or, more generally, exchange of information are based on the Internet and domain names. The DNS is estimated to manage 1.01 billion internet hosts (Statista, 2022). This widespread adoption has led people not to question the integrity and authenticity of a website when its domain name is recognized or seems to have a relationship with a trusted company, government, or entity. Domain names have increased in importance as a form of identity, resulting in high prices for relevant and

easy-to-remember names (Styler, 2022). Further, legal disputes about trademark rights for specific domain names have taken place (e.g., *lambo.com* for the car manufacturer *Automobili Lamborghini S.p.A.*[1]).

In our work, we recognize the importance of domain names and their increased adoption as a form of identity. Leveraging such a system in a blockchain-specific context might help solve or mitigate issues we raised in Section 1.1, as it can be unclear whether an entity is the party it claims to be. DNS might help to understand whether an on-chain address is connected to a domain and thus a trusted business or government. Nonetheless, while DNS is the key pillar for human readable names, the integrity, authenticity, and privacy of communication depend on cryptographic premises defined within TLS and X.509.

### 2.1.2. Transport-Layer-Security and X.509

The integrity, authenticity, and privacy of the communication between an entity that connects to a website and the respective web server are not automatically established by DNS. However, other systems are in place to provide these security guarantees, namely TLS[2] and the certificate standard X.509.

#### Transport Layer Security (TLS)

The latest version of TLS, 1.3, is defined in RFC 8446 (Rescorla, 2018), and previous versions 1.2 and 1.1 were defined in RFC 5246 and RFC 4346, respectively (Dierks and Rescorla, 2006; Rescorla and Dierks, 2008). TLS *"allows client/server applications to communicate over the Internet in a way that is designed to prevent eavesdropping, tampering, and message forgery"* (Rescorla, 2018). More concretely, the properties of client/server application communication via the Internet provided by TLS are, as defined in section 1 of RFC 8446, *authentication*, *confidentiality*, and *integrity* (Rescorla, 2018).

1. **Authentication:** The client-side authentication is optional, whereas the authentication of the server side is mandatory.

[1] `https://domainnamewire.com/2022/08/11/lamborghini-wins-lambo-com-as-namepros-username-defense-fails/`, accessed on 4th September 2022.

[2] TLS is a derivative standard from the Secure Sockets Layer (SSL) standard initially proposed in the Netscape browser in 1992, which has been deprecated. Nonetheless, these terms are often used interchangeably; for consistency and clarity, we only use the term TLS.

2. **Confidentiality:** The data transported by the TLS layer is readable only by both endpoints; no party in between can read the communication as long as other security properties hold up[3].

3. **Integrity:** The TLS layer ensures the integrity of the data, meaning that an attacker cannot modify the data without detection.

TLS defines a set of steps to establish secure communication between both parties by a) facilitating a session-specific encryption key and b) relying on trusted cryptographic information as defined in the X.509 certificate standard. Again, formal definitions, processes, and methodologies are well-defined in respective RFC documents (Rescorla, 2018) and are not reiterated in this thesis.

Therefore, TLS establishes a secure connection between two parties, but its security relies on valid and trustworthy certificates.

### X.509 Certificate Standard

X.509 is a standard that describes the structure and creation of digitally signed certificates for hierarchically organized PKIs. It is formalized in the ISO/IEC 9594-8 (ISO, 2022) and RFC 5280 (Boeyen et al., 2008). This standard also defines the structure and functionality of the certificate revocation lists (CRLs), which allow certificate authorities (CAs) to withdraw compromised key material.

X.509 is used predominantly for three specific applications:

- **Websites/Web hosts :** On the Internet, X.509 certificates[4] are leveraged to authenticate the respective website or host. Modern browsers utilize lists of trustworthy issuers of such certificates to verify the host's authenticity and establish encryption between both nodes in the network. The certificates are bound to the respective domain name and are issued only to the entity controlling the domain.

- **E-mail Communication:** The communication between mail servers leverages transport encryption to protect the contents of emails. However, these systems do not allow for end-to-end encryption between the sender and receiver of an email. Secure/Multipurpose Internet Mail Extensions (S/MIME) is a standard for

[3]For example, if malicious parties can obtain private keys of certificates, the confidentiality of TLS can be broken.

[4]Often, they are referred to as SSL-certificates or TLS-certificates. We interchangeably use the term X.509 or TLS certificate.

encrypting and signing emails. The X.509 certificate contains the email address. The certificate's authenticity is established by the CA or by the company setting up the PKI. A classification for these certificates exists, providing information on the quality of user authentication.

- **Code Signing:** With respective flags in the X.509 certificate[5], entities can leverage certificates for code signing. Code signing is the process of signing software programs and delivering this signature alongside the actual program. This ensures the integrity of the software and protects against unintended manipulation when proper key management is in place.

As previously mentioned, X.509 is organized in a strict hierarchical way. In contrast to web-of-trust systems, in which individual users weigh the authenticity and trustworthiness of certificates on a per-case basis, single entities issue new certificates, which, in turn, can issue certificates themselves. In the case of X.509, a hierarchical tree structure allows an entity to verify a certificate and the path up to a root certificate. This root certificate needs to be trusted as well. Often, these certificates are stored in the device's operating system or the web browser.

To issue new certificates, entities need to prove that they are in control of a to-be-certified asset (e.g., a web domain or an email address[6]). Suppose they want to include additional information in the certificate (such as the organization's name, country of origin, or something else), the correctness of this information needs to be verified as well. Certificates containing such additional information are often referred to as *Extended Validation* (EV) certificates, as this information might be displayed prominently in a browser. These certificates comprise only a fraction of all certificates (about 0.9% of the top 1,000,000 websites use an EV certificate (BuiltWith, 2022)).

There are two approaches to proving the ownership of a domain; either controlling the respective DNS entry or managing the host to which the DNS entry is linking. Proving the ownership of the DNS entry enables the creation of wildcard certificates and the equipping of any subdomain with the newly generated certificate. Establishing control of the web host enables the request of regular certificates linking to one specific FQDN. Both approaches can be facilitated via the Automated Certificate Management Environment (ACME), requesting and issuing new certificates for domain owners via, for example, Let's Encrypt (Let's Encrypt, 2022).

[5]The issuing entity can set `Code Signing` in *X509v3 Extended Key Usage* to enable code signing for the respective certificate.

[6]For relevance, we cover the issuance of certificates for domain names only.

The revocation of certificates, or, more generally, digital signatures, is a cumbersome process. Initially, verifying the integrity and correctness of a signature alongside its signed content does not involve any third party; after the certificate is retrieved from a third party, no further information is required to verify the certificate chain. Therefore, the verification processes is extended to include additional integrity information. Two methodologies are used to revoke credentials:

- **Certificate Revocation Lists (CRLs):** CRLs are lists issued periodically by certificate authorities or a separate trusted entity containing any certificate they intend to revoke. This period is usually 24 hours or less. The CAs themselves sign these lists to prevent spoofing. CRLs are vulnerable to denial of service (DoS) attacks[7]. Further, distributing all revoked certificates result in high network traffic.
- **Online Certificate Status Protocol (OCSP):** OSCP is a protocol leveraged to obtain the validity state of an X.509 certificate and serves as an alternative to CRL. Instead of requesting the entirety of revoked certificates for a single issuer, it asks explicitly about the revocation state of a specific certificate. While more efficient in terms of traffic usage, this method raises concerns about users' privacy, as it discloses the surfing behavior of the individual to the relevant CA. A solution for this problem is OCSP stapling. Instead of each browser requesting the validity state, the server requests it periodically and provides it as a response "stapled" to the TLS certificate (Eastlake, 2011).

#### Implications

Protocol designed to address the problems described in Section 1.1 should have properties similar to those of TLS and other existing technologies. First, the mandatory authentication of the server side and optional authentication of the client side are desirable. Entities that host websites could also deploy smart contracts protected by similar security mechanisms. The protocol could also ensure client-side authentication, but other means of authentication could be leveraged. Additionally, data integrity is relevant in this protocol, as information about the to-be-addressed smart contract needs to be valid. Although TLS provides confidentiality, it is irrelevant for solving the above mentioned issues. The secrecy of on-chain transactions is diametrically opposed to the goal of transparency and verifiability of public blockchain systems; therefore, leveraging the same

[7] If the entity hosting the CRL cannot provide the CRL to requesting browsers, the browsers cannot verify if a certificate is revoked.

mechanics in an on-chain context does not provide any confidentiality. Nonetheless, the confidentiality of on-chain transactions and their contents receives increased attention through systems such as TornadoCash and other cryptographic mechanisms, such as Zero-knowledge proofs (ZKP) and homomorphic encryption, allowing for the verification of contents without actually revealing them.

### 2.1.3. Other Security Measures

Other security methodologies arose due to issues with existing technologies and protocols. In this section, we cover two of the more broadly used technologies, *CT* and *DNSSEC*.

#### Certificate Transparency

CAs send newly issued certificates directly to the requesting entity (e.g., the host). One issue with this approach is that it is unknown whether or how many certificates exist for a given entity or domain. Malicious actors that have obtained the private key of a CA or the CA itself could issue new valid certificates for a given domain. This missing transparency is dangerous, as such certificates can easily be misused. This is a known attack scenario: In 2011, DigiNotar was a victim of such an attack, and consequently, malicious certificates for the domains *Google*, *Yahoo*, and *The Tor Project* were issued and used in man-in-the-middle (MITM) attacks. This attack sparked discussions about removing the single point of failure from the Internet infrastructure and has led to the development and usage of CT (Amann, Gasser, Scheitle, Brent, Carle, and Holz, 2017).

CT aims to provide a tamper-proof and append-only log of all newly issued certificates by any CA. RFC 6962 (Laurie et al., 2013) defines its core functionality and RFC 9162 provides version 2.0 (Laurie et al., 2021). The company Google has been substantially involved in the development and extension of CT and forces CAs to adhere to the standard; in case of non-adherence, Google provides warnings for certificates of respective CAs in Google's browser Chrome (Sleevi, 2016). Some browsers, such as Edge, support CT, and others, such as Mozilla Firefox, do not[8].

From a technical perspective, CT provides a linked hash tree. CT provides the current root hash ($t_n$) signed by the respective companies, including Google. On the left side of the tree, the last root hash ($t_{n-1}$) is provided, whereas the right side of the tree comprises all new certificates issued to CT since $t_{n-1}$. It is impossible to remove certificates once they have been submitted to CT. Also, it is relatively easy to prove the inclusion of

[8]See `https://bugzilla.mozilla.org/show_bug.cgi?id=1281469`, accessed on 8th February 2023.

a certificate in CT. The respective browsers also require this proof before deeming a certificate valid.

CT does not provide security on the certificate itself; it verifies the correctness of the certificate, such that it does not include an invalid certificate in the log, but it cannot decide if a malicious actor issued the certificate. This is the responsibility of the domain owner. CT allows anyone to monitor newly issued certificates for a given domain. In case a certificate is issued without the owner's intention, measures such as invalidating the individual certificate and an attack analysis can take place[9].

### DNS Security Extensions

DNSSEC are standards defined in RFC 4033, 4034, 4035, 5011, and 5155 (Arends et al., 2008; Rose et al., 2005a,b,c; StJohns, 2007). They were initially proposed in 1999 in RFC 2535 (Eastlake, 1999) but failed due to their complexity. A proposal was made in 2005, again based on the previously mentioned RFCs. It aims to ensure the authenticity and integrity of DNS records. With DNSSEC, requesting entities can verify the integrity of the record and prove that it is identical to the version issued by the owner of the zone file. Regular DNS relies on unsigned communication, enabling the manipulation of responses from the individual name server. DNSSEC helps to overcome this limitation. Privacy and confidentiality are, in contrast, not goals of DNSSEC.

To enable the authenticity and integrity of DNS records, DNSSEC relies on public key cryptography. As the DNS system is strictly hierarchically organized, a similar approach to TLS and X.509 was chosen. For every zone file that supports DNSSEC, a key is used to sign the respective responses. This key hash is stored within the zone file, and a key signature is stored in the overarching zone file.

## 2.2. Blockchain Networks

Blockchains, more generally known as Distributed Ledger Technologies (DLT)[10], describe networks of interconnected computers, often called nodes, that ensure the integrity of the

[9] The chance of a compromised CA issuing malicious certificates can be considered lower than an attack on one's infrastructure. Therefore, if such wrongly issued certificates exist, verifying the integrity of the own infrastructure is advisable.

[10] DLT describes technologies that manage a distributed state in a decentralized network, whereas blockchains fulfill the same goal by relying on blocks that bundle transaction information. As technology exists that does not rely on blocks, the more general term DLT is often used. For readability reasons, we will use both terms interchangeably. The results of our work can be leveraged in both blockchain and DLT networks.

state and its continuation of a specific application. For this, they exchange information on newly occurred transactions and states. Consensus mechanisms ensure that the integrity of the network is secured and that all honestly behaving nodes accept the same state as correct.

A blockchain fulfills three core properties, namely the *absence of trust*, being *tamper proof*, and *transparency* (Narayanan et al., 2016).

- **Absence of Trust:** The system does not require a specific or privileged third party to control or run the network. Instead, independent and equal entities run the network by contributing to its consensus mechanism.
- **Tamper Proof:** No single party can unilaterally alter the system, its contents, and the history of all recorded transactions.
- **Transparent:** The system, contents, and the history of all recorded transactions are transparent to all participants, and their integrity can be verified.

For the reader to follow the contents of this thesis, the inner workings of a blockchain network are irrelevant. Nonetheless, we briefly touch on the layers of blockchain networks and the concepts behind public and private blockchains in subsection 2.2.1, as these become relevant for later sections. The basis for our work is the concepts and limitations of the programmability of smart contracts in blockchain networks, which we cover in subsection 2.2.2. While the concepts introduced in this thesis apply to any programmable blockchain, we explicitly leverage and utilize Ethereum and the smart contract language Solidity.

### 2.2.1. Underlying Concepts

Blockchain networks rely on the already-established Internet and leverage regular transport protocols, such as TCP. Therefore, we locate these networks in layers 5 (session) to 7 (application) in an *Open Systems Interconnection Model* (Zimmermann, 1980). The layers of blockchain networks are complex and can deviate in the number of layers, separation, and system boundaries. For this thesis and essential distinctions relevant to this thesis, we define three separate layers:

- **Network Layer:** The network layer contains the computers and nodes that connect and exchange messages about the network state. Basic syntactic and semantic rules in this layer also apply to these messages. This layer includes any

consensus mechanism, such as Proof of Work or Proof of Stake. Usually, these networks are bootstrapped from a few hard-coded nodes that share information about other nodes, forming a loosely coupled network utilizing a gossip-like protocol. Limitations introduced in this layer lead to the formation of *private* or *permissioned* blockchain networks, as we outline below.

- **Application Layer:** The application layer is responsible for the networks' intents and purposes, e.g., what the developers of the network intended to do. For example, Bitcoin establishes an open payment system in this layer. For other networks, such as Ethereum, this payment system is augmented with a Turing-complete low-level bytecode language that allows the creation of smart contracts that can interact similarly to regular accounts. This layer is responsible for the application-specific state (e.g., which account owns which funds or information storage).

- **Interface Layer:** The third layer provides the users with the system's data and enables them to interact with the underlying application. Interfaces take the form either of automated systems such as Application Programming Interfaces (APIs), or actual interfaces in the form of wallets or decentralized applications (dApps)[11].

Changes introduced to these layers can severely impact the functionality of perception of the respective networks and, thus, are considered cornerstones of these networks. We make a significant differentiation between **public permissionless**, **public permissioned**, and **private** blockchain networks.

In **public permissionless** blockchain networks, anyone can participate in the network itself and in forming the consensus (rather than just validating it). Any node in the network has the same rights and cannot restrict other entities in any way. This is the case for most cryptocurrencies, as any limitation on participating in the consensus mechanism could severely weaken interest in the platform. Examples of public permissionless networks are Bitcoin and Ethereum.

In **public permissioned** blockchain networks, anyone can participate by connecting to existing peers and fetching block data, as well as verifying its integrity and adherence to the consensus rules, but cannot participate in the consensus itself. This results in a privileged group of nodes that run and govern the network and are essential to its correct functioning. Depending on the limitations imposed by the privileged nodes, other

[11]To be more accurate, dApps refer to both the underlying smart contract as well as the interface that lets the user interact with it. The smart contract is often decentralized, whereas the interface is hosted on a centralized server such as a website.

entities might be able to interact on the application layer. These networks are often run for a specific purpose or to serve a particular public interest group while these groups remain in control to a certain extent.

In **private** blockchain networks, only previously designated nodes can connect to the network and, depending on the design of the network, might or might not be able to participate in its consensus. These are often highly specialized networks that serve a specific industry group interested in sharing non-public data between members, as data is accessible only for predefined nodes and participants. Manual onboarding, such as connecting to virtual private networks (VPN), can be cumbersome. Industry consortia often rely on private blockchain networks.

### 2.2.2. Blockchain Programmability

Blockchain networks can support arbitrary programming languages; therefore, these systems can theoretically be Turing-complete.

One often-mentioned limitation of blockchain programmability is runtime complexity. Given that all nodes in the network verify and execute a transaction, execution costs are multiplied by the number of nodes in the network. Because of runtime issues and transaction complexity, execution time is severely limited in these networks. Further, malicious actors must be prevented from halting the network by occupying all nodes with non-trivial computational tasks, rendering the network's utility zero. In public permissionless networks, transaction fees based on the complexity of transactions prevent excessive transaction spamming. In permissioned or even private networks, individual malicious actors can be banned or are non-existent due to the preselected user group.

### 2.2.3. Smart Contracts

Besides regular users that interact with the blockchain relying on externally owned accounts (EOA)[12], smart contracts exist. A smart contract[13], in contrast to an EOA, is controlled by the code it was set up with. Once invoked, a smart contract behaves as specified by the code. Apart from one limitation, a smart contract has the same

[12]EOAs are blockchain addresses that are controlled by a private key and, thus, by the entity that controls the private key.

[13]There are more general definitions for the term smart contract, e.g., by Nick Szabo in 1994: "A smart contract is a computerized transaction protocol that executes the terms of a contract" (Szabo, 1997). For relevancy reasons, we limit ourselves to discussing smart contracts as a concept in the Ethereum blockchain.

capabilities and functionality as an EOA; it can receive and send transactions (with or without money), interact with other smart contracts, and create new smart contracts. However, a smart contract cannot invoke itself automatically but needs to be called by an EOA that executes the smart contract in question. Therefore, the originator of any blockchain interaction is always an EOA.

Ethereum relies on low-level bytecode that is executed in the Ethereum Virtual Machine (EVM). Multiple programming languages, such as Solidity or Vyper, allow human-readable code and compilation to EVM bytecode. Given its adoption and broad community support, the code referred to in this thesis is Solidity.

Many public permissioned and private blockchain networks besides Ethereum rely on the EVM for their purposes. Even other public permissionless blockchain networks support the EVM.

### 2.2.4. Bridging Information Between the Off-Chain World and On-Chain Blockchain

The three properties and core principles of blockchain networks mentioned in Section 2.2 lead to additional restrictions on the capability of these systems. While these programming languages can be Turing-complete, access to sources of randomness is severely limited. Input variables for pseudo-random number generators (PRNGs) are known prior and, thus, results are predictable. Under specific circumstances, these input factors can be influenced, rendering the purpose of a random number obsolete.

Also, all state transitions (resulting from transactions) need to be deterministic. A transaction execution must yield identical results on any machine in a network. Blockchain networks do not support any non-deterministic behavior. The requirement for deterministic behavior removes all possibilities to call external services on the blockchain, e.g., calls to external APIs or other data sources are not natively supported.

The primary mitigation strategy for this limitation is the usage of so-called *oracle services*. Oracles are entities that own an EOA, monitor on-chain activity and requests for data, and collect required information *off-chain*[14] to push it on-chain. While this might be a straightforward approach, it has a considerable downside: **centralization**. The oracle is a centralized entity that controls the data flow. This introduces centralized, privileged entities in a decentralized context. The oracle might manipulate data, stop serving specific entities, or block data publishing entirely. Additionally, the oracle likely

[14] *Off-chain* refers to any activity that happens in the Internet outside of the blockchain.

requires a fee for it to operate. Lastly, given the monetary aspects of public permissionless blockchain networks, oracles might be tempted to change randomness values in their favor.

There are several methods to enhance the trustworthiness of an oracle service, but tradeoffs need to be considered.

The **decentralization of the oracle service** is a straightforward approach. Instead of relying on a single oracle, multiple oracle services could access identical information and publish respective data to the blockchain. In an on-chain setting, e.g., in a smart contract, some form of voting could take place, such that the value reported by the majority of oracles is accepted. This introduces novel problems such as increased costs, certification of oracle service providers[15], and collaboration attacks, in which multiple oracles can conspire to push false values to the chain to gain a financial advantage. Oracles that generate randomness are easier to implement. Users publish the hashes of random numbers with a commit and reveal scheme (Gallersdörfer et al., 2020) on-chain to reveal their contents later and compute an overall random number that no entity could have predicted.

The **verification of the integrity of the published information with asymmetric cryptography** can be a good approach to enhancing an oracle's trustworthiness. At the initial setup of a smart contract system, a public key is stored on the blockchain and later used to verify the information. Assume a trusted entity (not the oracle provider) publishes its data with respective signatures online. Oracle services then pick up the information and push it on-chain. If the smart contract has stored the original public key, it can verify the integrity of the information pushed on-chain by the oracle service. We leverage this approach in this thesis. This approach also has downsides, as storing and verifying data is expensive, significantly increasing costs.

A detailed overview of the state of the research in this area is given in the next section.

## 2.3. State of the Research

We identify two core areas of related work adjacent to our research. First, some research leverages the properties and cryptographic key material of TLS certificates to enable applications within blockchain networks, e.g., proofs of website contents. Second, one system leverages DNSSEC to allow the usage of standard domain names within a blockchain network.

[15]Otherwise, an oracle provider could just create multiple identities.

We acknowledge that much research has been done around either augmenting current PKI systems with elements from blockchain networks or entirely replacing them with blockchain-based systems, enhancing accountability and transparency. While existing systems face issues and limitations, our research focuses on improving usability and security within blockchain networks and accepts these systems as given. Updating and enhancing current systems require tremendous effort and time, as it takes many years for the community to fully adopt such systems (e.g., DNSSEC). An extensive overview is given in (Brunner et al., 2020).

### 2.3.1. Leveraging Cryptographic Key Material in Blockchain Networks

The cryptographic key materials of TLS certificates or other PKIs have previously been used in research. As a result, we refer to two significant results and explain the delta of our approach.

- **Town Crier** is a research project that enables *"authenticated data feeds for smart contracts"* (Zhang et al., 2016). The authors identify the same problem described in subsection 2.2.4: In order to access data outside of the blockchain, oracle providers are required to push data to the blockchain. The specific use case depends on the trustworthiness of these oracles; incentives might exist for these oracles to behave maliciously. To prevent malicious behavior, Town Crier establishes an approach that forces these oracle providers to a) rely on a Trusted Execution Environment (TEE), the secure hardware enclave named Intel SGX, and b) provide proofs of correct execution on the blockchain. This enclave ensures that processed data, e.g., the contents and the TLS certificate, are valid and cannot be tampered with. This also enables the publication of private data so that only the intended recipient is able to read the data on-chain. While TLS certificates are leveraged, this approach, unfortunately, relies on the correct functioning of the secure hardware enclave from Intel, which has been broken in the past (Nilsson et al., 2020). The technology was adopted in Chainlink in 2019 and continues to be used within their systems[16].

- **TLSNotary** and *Oraclize.it*[17] leverage TLS certificates to prove the authenticity of the contents of websites to a third party called auditor that can verify the

[16]See `https://blog.chain.link/town-crier-and-chainlink/`. Accessed 11th September 2022.
[17]Oraclize.it has been rebranded to Provable.

correctness of the respective presented contents. It currently supports TLS 1.1 and TLS 1.2. Initially designed as technology-agnostic, it did not require or rely on a blockchain network. In Provable's smart contract ecosystem, it is one of the multiple proof types for the authenticity of data published to the blockchain. Therefore, it does not rely on a TEE; it depends on *"a locked-down AWS instance of a specially-designed, open-source Amazon Machine Image"* (Provable, 2022). This also introduces dependence on a specifically designed system, which users must trust.

In both cases, our proposed system does not rely on trusted execution environments or specific software setups to prove the correctness of on-chain data. Instead, by publishing key material directly to the blockchain and bootstrapping the smart contract system with trusted root certificates, we avoid the dependence of other entities on our approach.

### 2.3.2. Enabling Domain Name Usage in Blockchain Networks

EN) is a system of smart contracts that allows anyone to reserve and claim domains for the TLD `.eth` on the Ethereum blockchain (Ethereum Name Service, 2023b). Currently, no major browser supports the TLD managed by ENS, and it is not an officially recognized registrar within the traditional DNS PKI systems. Opera supports ENS-domains and displays their content if a website is available via Interplanetary File System (IPFS) (Benet, 2014), as seen in Figure 2.2[18]. Brave supports crypto-specific DNS systems, such as ENS and UnstoppableDomains, but requires users to initially opt-in to leverage these technologies[19]. ENS is managed by the ENS DAO, which controls the development and rules of the ENS ecosystem. More than 2.3 million names have been created[20] as of September 2022. The ownership of each name is purely in the control of the respective registrant, and the DAO itself cannot reclaim previously issued domain names. This is in contrast to the existing DNS, in which entities are able to take legal action to claim access to a specific domain name. The technical setup of ENS is explained in great detail in their docs[21], therefore, we do not reiterate the entire technical foundations. We discuss parts of ENS architecture in Section 5.3. Previous research highlights issues within the ENS ecosystem, such as domain squatting, malicious websites, and others (Xia et al., 2022).

[18]The protocol `ipfs://` is not entered manually, but automatically added when visiting `vitalik.eth`.
[19]`https://brave.com/brave-crypto-dns-strategy/`, accessed 11th September 2022.
[20]`https://dune.com/makoto/ens`, accessed 11th September 2022.
[21]`https://docs.ens.domains/`, accessed on 11th September 2022.

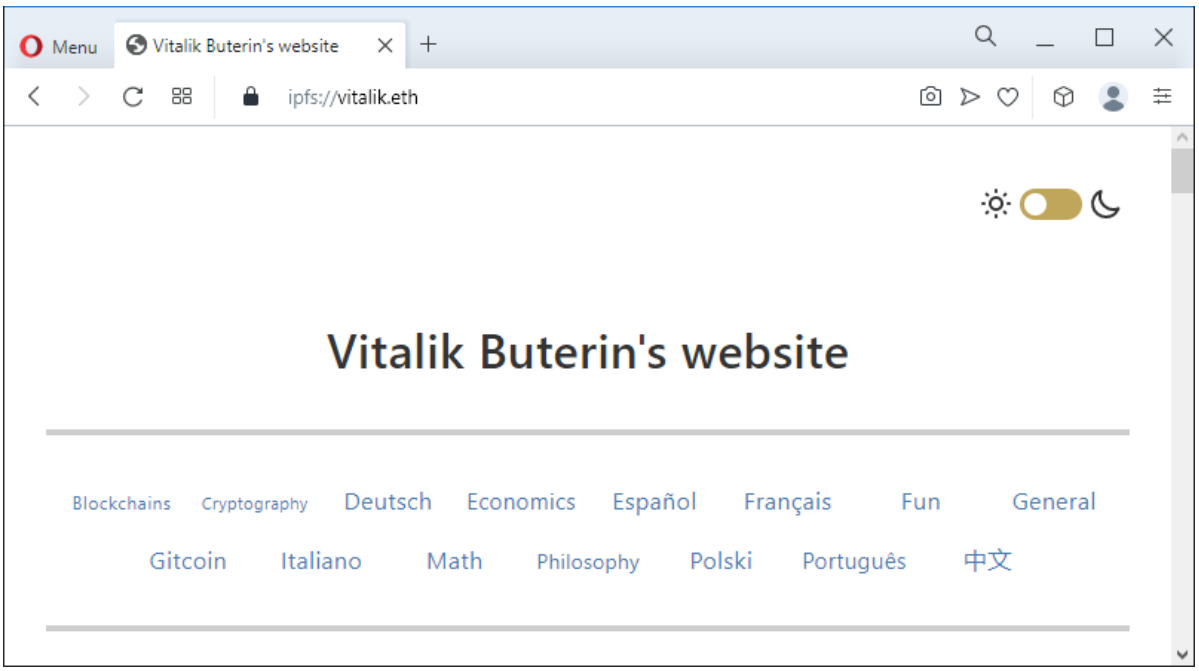



**Figure 2.2.:** Screenshot of the Chrome browser accessing `ipfs://vitalik.eth`, the website of the Ethereum co-founder Vitalik Buterin. Accessed on 27th February 2023.

Interesting, and more relevant to our research, is the experimental support of domain names managed in the traditional DNS system. Instead of requiring users to obtain and purchase a `.eth` domain, they can claim the ownership of an already existing domain managed in the traditional DNS system[22]. Users must provide proof of ownership utilizing DNSSSEC, which we introduced in Section 2.1.3. The respective TLD needs to support DNSSEC, and the users need to update their zone files to point to the registrar address on Ethereum. The requirement to use DNSSEC restricts the number of domains utilizable in such an environment, as not all TLDs support DNSSEC. We investigate the availability of DNSSEC in Section 6.1. An important limitation of this approach is that domain owners can not claim subdomains. The owner of the second-level domain (e.g., *example.org*) is allowed to issue subdomains (e.g., *sd1.example.org*) in ENS without a requirement to honor potentially deviating records in the zone file for the domain in question.

Key differences between this system and the TLS-based systems described in this thesis exist:

- ENS' main focus is on growing the `.eth` domain landscape, as this is the main source of revenue. While the DNSSEC support is integrated into the ecosystem, it

[22] `https://veox-ens.readthedocs.io/en/latest/dns.html`, accessed 11th September 2022.

is still in an experimental phase, although initial work began in 2017[23].

- ENS relies on DNSSEC, whereas our approach generalizes the approach for any form of X.509 certificate. In our case, support for proprietary or private PKI systems is possible, whereas DNSSEC is limited to the existing DNS system.

We thoroughly analyze the ENS DNSSEC approach in Section 5.3 and highlight further key differences in Section 5.4. Further, in Section 5.5, we develop a dataset of all domains bridged to ENS using DNSSEC.

## 2.4. Summary

This overview highlights the importance of the systems that provide the basis for the WWW and the novel design of blockchain technology and smart contracts. This thesis identifies the gap between WWW and blockchain networks as a potential area for research. It contributes to narrowing this gap to mitigate issues arising from the loose interconnection between these systems. Using standardized cryptographic premises existing within the WWW and applying them in a blockchain-specific context yields benefits of the existing WWW in a blockchain context to enhance the usability and security of these networks. Following concepts enable this potential:

- **Trust in DNS**: The DNS, as a central pillar of the WWW, ensures the authenticity and trustworthiness of entities, e.g., companies, governmental institutions, or, in some cases, individuals. From this perspective, one might recognize domain names as a globally valid "pseudo"-identity. We are unaware of any other system that provides trust, security, and adoption comparable to DNS. Although security issues and other downsides exist, DNS adoption has occurred worldwide.

  In this thesis, we utilize DNS and domain names to extend the scope of the validity and trustworthiness of these certificates in a blockchain context to leverage their advantages. Additionally, we discuss the relevance and recognition of domain names in a blockchain context by end users.

- **Cryptographic Premises**: The existence and usage of cryptographic key material for establishing secure connections to web servers and other services, including certificates and certificate chains, allow for a standardized and efficient verification

[23] `https://github.com/ensdomains/dnssec-oracle`, accessed on 11th September 2022.

of the authenticity of the counterparty and ensures the integrity of content and information.

Based on a standardized format and processes, we propose in this thesis the usage of certificate and certificate chain information for straightforward and cost-efficient issuance and authentication of identity attributions that are initially rooted in the DNS.

- **Blockchain as Immutable Storage and Execution Layer**: Blockchain systems, notably public networks, are recognized as immutable ledgers capable of storing millions of records for prolonged periods. Given the deterministic and Turing-complete execution environment of some of these networks, it is possible to create autonomous entities, called smart contracts, within these networks that obey only the initially provided application code.

  In our work, we first utilize blockchain technology to store parts of the certificate and key and signature information to protect integrity and ensure its availability of signature statements. Second, this information can be verified in an on-chain context, relying on the Turing-complete execution environment in blockchains and enabling a globally-shared perception of the authenticity of stored certificates and signatures. Third, verification is also enabled in an off-chain context, allowing users to perceive the authenticity of material stored on-chain individually.

In the next chapter, we introduce the fundamental concept for utilizing certificates of PKIs in a blockchain context, discuss potential solutions, and define the endorsement as a way to link from a domain name to a blockchain address.

# Chapter 3.

# Utilizing Public Key Infrastructures in Blockchains

This chapter covers the core concept and idea enabling the utilization of PKIs in a blockchain environment. We provide insight into the approach and methodology of utilizing X.509-certificates and their signatures within blockchain systems, derive and explain requirements, and analyze the solution space for the architectural design and implementation.

In more detail, we split this chapter into the following sections:

- Section 3.1, *"Methodological Approach"*, gives an overview of the core idea and how components of existing WWW infrastructures can be leveraged in Blockchain networks. Further, we describe the processes and data required to enable such usage.
- In Section 3.2, *"Requirements"*, we derive the requirements for the architecture design. First, we describe issues and use cases related to the problem statement in Chapter 1. Second, we analyze the requirements originating from these use cases, and third, we detail the requirements set out by the RFCs.
- We provide an understanding of the solution space in which PKIs can be used in blockchain networks in Section 3.3 *"Solution Space"*. In this section, we differentiate between two core concepts described in this thesis.
- Lastly, in Section 3.4, *"Endorsement"*, we introduce the first essential component of the system: the endorsement. The endorsement represents a verifiable claim that the entity owning a certificate and its associated rights (e.g., a claim to a domain) wants to attribute itself to an on-chain entity (e.g., an address or smart contract).

Parts of Section 3.4 are based on the material of one prepublication, namely "TeSC: TLS/SSL-Certificate Endorsed Smart Contracts" (Gallersdörfer and Matthes, 2021a).

## 3.1. Methodological Approach

In the previous chapter, we outline the importance and the role of PKIs in the WWW as a trust anchor of the naming scheme and in providing cryptographic key material to enable the integrity, privacy, and authenticity of communication between single entities. More generally, these properties apply not only in the context of the WWW but also in any other hierarchically organized PKI that serves identical purposes for a group of entities (e.g., a company-wide or inter-organizational PKI). We display an exemplary PKI in Figure 3.1, containing the certificate chain of the website `https://example.org`.

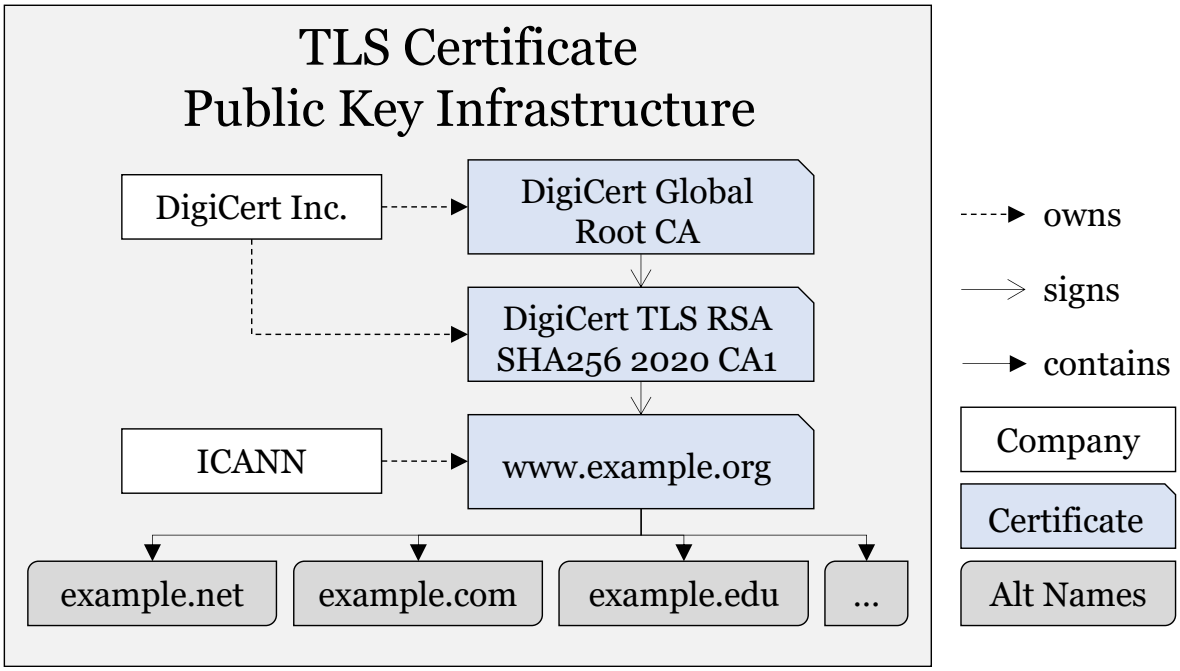



**Figure 3.1.:** Exemplary PKI Structure for the TLS certificate of `https://example.org`. Please note that `www.example.org` is included both as `Common Name` and `Subject Name` in the certificate in question. `DigiCert Global Root CA` is the certificate from the Root CA, and `DigiCert TLS RSA SHA256 2020 CA1` is an intermediary certificate, whereas `www.example.org` is the domain certificate or leaf certificate. Website accessed on 27th February 2023.

The cryptographic key material and its organization into hierarchical structures and signature chains enable an efficient verification of properties[1] and statements[2]. Given the limited resources available in Blockchain networks, the leveraging of cryptographic

[1] Properties are attributes assigned to specific entities within the PKI, e.g., a domain name or, in the case of S/MIME, e-mail addresses. In Figure 3.1, `Alt Names` are a form of property.

[2] We refer to statements for signed or encrypted data that is leveraged for different purposes, e.g., a key exchange for symmetric encryption.

key material or certificates must be as efficient as possible to reduce costs.

X.509 certificates and their properties have been widely adopted in the WWW. After IP address information has been exchanged via DNS, certificates enable secure communication and authentication through TLS. These certificates and their properties can be utilized within Blockchain networks. As a result, the certificate or the property itself is insufficient, as a statement must be formulated to extend the attribute's scope to a blockchain address. The root certificates, the certificate verification path, the certificate itself, and the statement, including its signature, are required to verify the attribution to the blockchain address successfully.

A valid signature can be interpreted as a statement from the owner of a certificate to bind a blockchain address to its entity, thereby endorsing the authenticity of the individual address. The party recognizes that other parties can attribute the original property (e.g., the domain name) to the blockchain address, thereby enabling the usage of these properties in the context of blockchain networks. We refer to such a statement as an *endorsement*. A more detailed definition of an endorsement is given in Section 3.4.

Between specific use cases and their requirements, the verification and the executing entity might deviate. In our work, we describe two suitable approaches:

- Decentralized off-chain-verification: The verification of an endorsement takes place in a decentralized manner, e.g., in the wallet of the user executing a transaction.
- Logically centralized[3] on-chain-verification: The endorsement verification takes place on-chain, and the required data is stored on-chain as well.

These methodologies deviate in their requirements and system architecture, and both are the basic designs for leveraging certificates in a blockchain context. We describe the rationale behind the design decisions and solution space in Section 3.3. Chapter 4 analyzes in detail the off-chain system design, whereas Chapter 5 covers the on-chain system design.

Despite the simplicity of the idea, several challenges occur in the design and development of such technology. As examples, we name three challenges requiring attention and potential mitigation efforts.

- As the system builds upon existing standards, such as TLS, DNS, and X.509, their properties and requirements must be considered.

[3]Whereas public blockchain networks are often decentralized, the consensus mechanism ensures that all nodes derive at the same state. Thus, for any computation that takes place on-chain, all nodes derive the same result, logically centralizing the process.

- The cryptographic key material needs to be kept secret, and our systems' existence and usage must not weaken the existing infrastructure's security properties.
- Costs are a major issue in blockchain networks and influence architectural designs and business decisions. Our system needs to cater to the high costs of single operations and therefore requires minimal costs sufficient for operation.

To conclude: We utilize the attributes of certificates managed within PKIs by leveraging their cryptographic key material and signing abilities to enable endorsements that attest relationships between the original attributes and blockchain-specific addresses. The challenges of designing a system that supports such an endeavor are manifold, stemming from the complexity of the underlying systems, their security requirements, and the restrictions of the blockchain space.

## 3.2. Requirements

This section discusses our proposed solutions' architecture and design space requirements. To this end, we provide problem statements derived from Chapter 1 and develop requirements for these. Additionally, the underlying technologies and standards, such as TLS, X.509, and DNS, pose particular requirements in their RFC specifications.

For clarity, we iterate both over functional and non-functional requirements. In subsection 3.2.1, we discuss requirements that stem from the problem statements posed earlier; in subsection 3.2.2, we generate requirements from the underlying technologies and standards. We summarize all requirements in subsection 3.2.3.

Requirements engineering is a core task of system design. Previous work revolving around leveraging TLS certificates in a blockchain context has defined requirements as well (Groschupp, 2020; Hoops, 2021; Strugala, 2020).

- (Groschupp, 2020) proposed a list of requirements for enabling an on-chain authentication framework. We briefly describe this framework in Section 5.2. The author refers only to on-chain authentication and does not differentiate between functional and non-functional requirements. We find similar requirements: FR3, NFR1, NFR3, NFR5, and NFR6 are in line with the author's findings.
- (Hoops, 2021) developed a set of requirements for the off-chain approach, which we describe in detail in Chapter 4 and which was previously published in (Gallersdörfer and Matthes, 2021a). Hoops reviewed over TLS-related RFC documents and prior

work to develop a set of requirements. No distinction between functional and non-functional requirements was made. We find similar requirements: FR5, NFR1, and NFR3 are in line with the author's findings.

- Subendorsements (see Section 3.4) and their usage are the focus of (Strugala, 2020). The author defines functional and non-functional requirements for a system enabling the usage of endorsements in an authorization context. Given the different focus of this work, the requirements do not align.

### 3.2.1. Problem-Statement Specific Requirements

In the following, we outline a specific set of problems that appear at the intersection of web and blockchain networks, namely **address replacement attacks**, **missing data authentication**, **non-human-readable names**, and **lack of access control**.

#### Problem Descriptions

We identify the following problems in the context of the interplay between Blockchain and Web 2.0:

**Address Replacement Attacks:** Web 2.0 services (such as regular web servers) often provide information on connecting to blockchain-based applications or services. However, these linkages are susceptible to attacks, as the respective services do not contain information for further verification of the authenticity or identity of these on-chain applications. Attackers use this weakness to trick users into interacting with malicious blockchain services and addresses, often resulting in lost funds or assets.

**Missing Data Authentication:** Companies and other institutions rely on sharing data, parts of data, or the integrity information of data on a blockchain. Proving the ownership and authenticity of this data to a third party requires interactive challenge-response mechanisms for which both parties must be online. Additional requirements, such as data provenance after the issuing institution ceases to exist, might be required.

**Non-Human-Readable Names:** Users obtain information about blockchain-based services via websites and other means of communication (e.g., instant messaging or mail). As there is no direct way to verify the authenticity or recognize the name of these services, directly relying on these addresses is comparable to surfing the web using IP addresses.

**Lack of Access Control:** Protecting services or blockchain networks from unauthorized access is an often-required feature. These mechanisms often involve a manual exchange of information via secondary channels such as mail or text. The owner of the

service or network is then able to safelist respective addresses. These manual processes are labor-intensive and prone to errors.

### Requirements Engineering

A set of basic requirements must be in place to solve or at least mitigate the above-mentioned issues.

At first glance, all four problems face the absence of a form of a strong identity on-chain. Replacement attacks benefit from an entity's inability to associate its domain ownership with address ownership. On-chain data authentication and the unavailability of human-readable names mean there is no way to publish content and act within the network with a recognized name. As a method of authorization, access control first needs a proper means of authentication.

We define a strong form of on-chain authentication as the first non-functional requirement.

> **NFR 1: Strong Form of Authentication**. Enable a strong form of authentication in the respective on-chain environment.

Public key authentication already exists within blockchain networks. However, similar to IP addresses in the WWW, these names are not human-readable or practical for widespread usage in the ecosystem. Therefore, we define human-readable names as a second non-functional requirement for easier adoption.

> **NFR 2: Usage of Human-Readable Names**. The names in the system should be easily recognizable by humans.

As we outline in Section 2.3, some existing systems, such as ENS, provide a form of human-readable names similar to regular domain names. A better understanding of how these names are recognized and used is needed, and they still lack adoption within the community. Further, outside the Ethereum community, these systems are not supported; rather, they face competition from cryptocurrency-specific name services[4], contributing to the ecosystem's fragmentation and highlighting the need for bootstrapping.

Therefore, we pose two non-functional requirements:

[4]For example, on the blockchain network Tezos, *Tezos Domains* (ending with `.tez`) exist. So far, they have only recorded about 140,000 domains in their system. Further information: `https://tezos.domains`, accessed on 8th February 2023.

**NFR 3: No Requirement for Bootstrapping**. The system should rely on a well-established system for name management and authentication.

**NFR 4: Blockchain Agnostic**. The system and its core concepts should be applicable in any blockchain network.

We also need to ensure that any entity can interact with the system. The system should be designed to be open, so no entity can limit access to the system. We aim to allow broad usage in the blockchain ecosystem, taking an agnostic perspective on blockchain networks, including the absence of intermediaries. Therefore, we define the following non-functional requirements:

**NFR 5: Decentralization**. Avoid centralization beyond already-existing naming services.

**NFR 6: Openness**. Allow anyone to participate within the system; thus, do not rely on intermediaries that could limit access to the system.

From a functional perspective, this form of authentication needs to support basic forms of interaction, as any similar certificate would:

- **Create:** Create the data object that later enables the verification of said object in a blockchain context.
- **Read:** Allow any interested party to obtain and read the data object that allows for authentication.
- **Update:** Allow the update of information within the data object to represent new states of said object.
- **Delete:** Delete and remove said object so that it is no longer valid within the system.

Given that we may deal with identity assertion/usage in the form of a certificate-like structure, it is recommended not to rely on a *create read update delete* (CRUD)-based approach but rather to adhere to certificate standard methodologies and to the functionalities of issuance, verification, and revocation. Therefore, we propose these three functional requirements:

**FR 1: Issuance**. An entity should be able to issue a valid form of a certificate-like object to extend the scope of existing naming rights in a blockchain-based environment.

**FR 2: Verification**. Any entity should be able to verify the authenticity of said object.

**FR 3: Revocation**. An entity that previously issued such an object should be able to revoke its validity.

One further functional requirement is necessary for *"Lack of Access Control"*. If an entity accesses a resource within a blockchain network, it actively contacts a smart contract to be authenticated. As smart contracts cannot reach out to third parties to access information about the authenticity of counterparties by themselves, we define the following functional requirement.

**FR 4: Active Usage**. The authenticity information issued within the envisioned system can be used for active authentication.

### 3.2.2. RFC-Specific Requirements

The RFCs we rely upon in our work are TLS, X.509, DNS, and CT. While these documents describe the required features, functionalities, and steps to adhere to the respective standards, the design goals and guarantees are relevant to our work. For example, we do not want to implement a TLS key exchange directly. Still, we need to understand the rationale behind the respective algorithms to properly implement our solution.

#### TLS – RFC 2246/4346/5246/8446

In RFC 2246 (TLS 1.0), RFC 4346 (TLS 1.1), and RFC 5246 (TLS 1.2), four goals of the TLS protocol are defined: **cryptographic security** (to establish a secure connection), **interoperability** (between multiple applications), **extensibility** (to enable new cryptographic signature and encryption methodologies), and **relative efficiency** (to reduce computational and network activity) (Allen and Dierks, 1999; Dierks and Rescorla, 2006; Rescorla and Dierks, 2008).

In RFC 8446 (TLS 1.3), the goals of the TLS protocol are further clarified: *"The primary goal of TLS is to provide a secure channel between two communicating peers"* (Rescorla,

2018). Further, the properties of the secure channel should be **authentication** (required for the server side, optional for the client side), **confidentiality** (data is visible only to the endpoints), and **data integrity** (data cannot be changed without detection) (Rescorla, 2018).

RFC 8446 further refers to RFC 3552: "Guidelines for Writing RFC Text on Security Considerations" which contains additional security-related requirements and goals (McFadden, 2019). Based on this, further subcategories are defined:

- **Communication Security**, targeting the above-mentioned three properties of confidentiality (2.1.1.), data integrity (2.1.2.), and peer entity authentication (2.1.3.).
- **Non-Repudiation** (2.2.), enabling one party to prove to a third party in a secure and authenticated way the existence and validity of any statement the other party has made.
- **Systems Security**, such as unauthorized usage (2.3.1.), inappropriate usage (2.3.2.), and denial of service (2.3.3.).

Given the blockchain context, we identify the following requirements to be relevant to our system:

- **Authentication**: A strong form of authentication is a requirement for a system that aims to extend the scope of TLS authentication at the intersection of Web 2.0 and blockchain. Entities must not be able to impersonate other entities given a threat model, as outlined in Section 4.3. This threat model includes partial access to one end-system on the issuing party (e.g., its web server), lowering the security goal from prevention to detection.
- **Data Integrity**: The integrity of the data exchanged in the protocol needs to be protected; given the use of blockchain as a data structure and the secure connections to other entities (e.g., protected via TLS), this requirement is implicitly fulfilled.
- **Non-Repudiation**: Entities in our system should not be able to deny their involvement later, allowing other parties to claim and prove the existence of signatures and their contents. This requirement should also hold in cases where signing parties claim the theft of their private keys.
- **Systems Security Requirements**: Unauthorized and inappropriate usage is implicitly included in authentication and data integrity requirements. Preventing

DoS attacks is essential, as the absence of name-based authentication information cannot be differentiated from the service's unavailability, as such information is standard in blockchain networks.

We remove the confidentiality of a connection as a requirement. However, we do not expect to directly expose that a cryptographic verification is taking place (protecting confidentiality at that level). The relationship between actors can be inferred by respective on-chain transactions, independent of whether verifications have occurred.

As a result, we define the following functional and non-functional requirements and omit duplicates[5]:

**NFR 7: Non-Repudiation**. The extension of the scope of an authentic certificate to a blockchain address cannot be denied after the fact.

**NFR 8: Robustness against Denial of Service Attacks**. The system is robust against DoS attacks.

### X.509 – RFC 2459/3280/5280

X.509 is a certificate standard and is part of a family of standards for PKI on the Internet. It defines the certificate and the certificate revocation list (CRL). The specification is intended for any application or system that facilitates the use of X.509 certificates, including our system.

X.509 certificates are broadly used within multiple PKIs for many use cases. To support a multitude of these PKI, our system should rely on X.509 certificates as a source for authenticity:

**NFR 9: PKI Agnostic and X.509 Support**. The system relies on the X.509 certificate standard and supports any PKI adhering to that standard.

In addition to the certificate structure (to which we implicitly adhere), the CRLs are a core component for revoking previously issued credentials. We support CRLs, given that revoking a certificate should also rescind the statements of this certificate. Our proposed system should also recognize the certificate's validity as communicated by other means, such as online certificate status protocol (OCSP).

[5]For simplicity, we subsume data integrity and additional elements of systems security under NFR 1 (Strong Form of Authentication).

**NFR 10: Adherence to Certificate Status**. The status and validity of the underlying certificates apply to statements and assertions managed within the system.

#### Further Specifications

As outlined in Section 2.1.3, CT (RFC 6962) aims to provide a protocol for a tamper-proof, append-only log of TLS certificates to verify and audit that log. CT enables the detection of maliciously issued certificates by a CA (e.g., when a CA issues valid certificates that are outside its scope[6]). Further, domain owners can recognize when someone issues a new certificate for their domain (Laurie et al., 2013).

Given the tamper-proof and append-only blockchain context, the fulfillment of this requirement in our system is implicit. Similarly to CT, the auditability of issued assertions enables the detection of malicious actors. However, we cannot automatically prevent an attack.

**FR 5: Auditability**. Any identity assertions issued within the system need to be traceable and auditable.

DNS, as outlined in RFC 1034 and RFC 1035, poses additional requirements on our system: adherence to the hierarchical naming schema, omittance of protocol information, and inclusion further details (Mockapetris, 1987a,b). If we adhere to an X.509 standard in an Internet context, we also adhere to respective naming schemes. Therefore, and for simplicity, we do not set up further functional or non-functional requirements.

### 3.2.3. Overview

To summarize the findings presented in the previous two subsections, we provide a list of all functional and non-functional requirements that the systems need to address:

- **Functional Requirements:**

  **FR 1: Issuance**. An entity should be able to issue a valid form of a certificate-like object to extend the scope of existing naming rights in a blockchain-based environment.

  **FR 2: Verification**. Any entity should be able to verify the authenticity of said object.

[6] E.g., issuing certificates without user request.

**FR 3: Revocation**. An entity that previously issued such an object should be able to revoke its validity.

**FR 4: Active Usage**. The authenticity information issued within the envisioned system can be used for active authentication.

**FR 5: Auditability**. Any identity assertions issued within the system need to be traceable and auditable.

- **Non-Functional Requirements:**

**NFR 1: Strong Form of Authentication**. Enable a strong form of authentication in the respective on-chain environment.

**NFR 2: Usage of Human-Readable Names**. The names in the system should be easily recognizable by humans.

**NFR 3: No Requirement for Bootstrapping**. The system should rely on a well-established system for name management and authentication.

**NFR 4: Blockchain Agnostic**. The system and its core concepts should be applicable in any blockchain network.

**NFR 5: Decentralization**. Avoid centralization beyond already-existing naming services.

**NFR 6: Openness**. Allow anyone to participate within the system; thus, do not rely on intermediaries that could limit access to the system.

**NFR 7: Non-Repudiation**. The extension of the scope of an authentic certificate to a blockchain address cannot be denied after the fact.

**NFR 8: Robustness against Denial of Service Attacks**. The system is robust against DoS attacks.

**NFR 9: PKI Agnostic and X.509 Support**. The system relies on the X.509 certificate standard and supports any PKI adhering to that standard.

**NFR 10: Adherence to Certificate Status**. The status and validity of the underlying certificates apply to statements and assertions managed within the system.

## 3.3. Solution Space

After outlining and defining requirements, we sketch a solution space in which the certificates of PKIs can be leveraged in a blockchain-based context. To this end, we iterate the individual elements of the hypothetical system and discuss the respective configuration options.

### 3.3.1. Fundamental Concept and Elements

The system aims to leverage certificates and their attributes managed within any PKI in a blockchain-based context. This means that anyone should be able to a) extend the scope of a certificate to any blockchain address they control and b) verify that this scope extension is valid. This means that a "sub-certificate" of the respective certificate creates this association. This association can be verified by anyone with access to the relevant OKI and, potentially, further information.

#### The Selection of Public Key Infrastructure

While limiting the system to one specific PKI is possible, an approach using any PKI that adheres to the X.509 standard is favorable. In this case, we need to define whether one instance supports multiple PKIs simultaneously or if separate instances are required.

The hierarchical structure of a PKI and its root certificates are core to its functionality. By definition and induction, the entirety of the certificates managed within one PKI exist due to the respective root certificates. This means that while we need to store the root certificates to be able to later verify any certificate or derived endorsement, the number of these certificates is limited and clearly defined. These root certificates also work on a *trust*-basis, meaning that if an entity does not trust a respective root certificate, it does not trust any of its derivatives. Instead of separately initializing instances of our proposed system for each PKI, a generalized system can be defined that manages an arbitrary number of root certificates. Instead of predefining these root certificates, entities that verify certificates or endorsements must be able to specify which root certificates they trust. The co-existence of multiple PKIs requires identical standards and the absence of conflicting objects (such as privacy or spam protection).

To conclude: Independent of other arrangements and decisions, the system is **PKI agnostic**.

### Processes

In addition to understanding the regular life cycle of the TLS certificates (creation, usage, and revocation), we need to understand the scope of the processes for endorsements and their integration within blockchain networks and the verification software. Following the TLS life cycle, we propose three critical processes for our system: the issuance, the verification, and the revocation of an endorsement. For each of the processes, we define potential arrangements. As issuance and revocation are closely coupled, we display their potential options in one section.

**Issuance and Revocation.** The issuance and revocation of an endorsement play an elementary role in the system. An endorsement is a derivative of a TLS certificate, or to be more precise, of its private key and signature. The certificate can also sign a revocation statement, rendering the endorsement invalid. This process should take place only locally. If the private key is shared or available to any other party, it might get misused for malicious purposes.

However, as we outline in Section 3.3.1, the endorsement is stored on-chain and refers to an on-chain entity. As this on-chain entity can have its own cryptographic material (in case of an EOA) or deterministic behavior (in case of a smart contract defined through its byte code), sub-endorsements could be created. In this case, the issuance and revocation of sub-endorsements could occur purely on-chain.

**Verification.** Verifying an endorsement and its certificate, including the certificate chain, is the core process of protecting the integrity of the linkage between an on-chain entity and the respective domain owner. The process can take place on-chain, and anyone can execute it. We need to decide in which context such verification can and should happen:

- **Local Execution:** In Section 3.2, we introduce the problem of loose coupling between websites, their interfaces, and the linked smart contracts, leading to the risk of malicious actors redirecting users to smart contract addresses that they control themselves. To prevent such attacks, it is necessary to verify whether an address belongs to the website in question and warn the user if it is not. Wallets must display these warnings to users. The verification can be executed securely within the user's local environment, as we assume that the attacker has no access to the user's machine[7].

[7] Our system does not aim to prevent attacks in which an attacker has access to a user's system

- **On-chain Execution:** In Section 3.2, we also introduce the problem of missing data authentication and access control in blockchain-based systems. Smart contract systems cannot authenticate other addresses on predefined rules and identity schemes. Suppose a smart contract or other on-chain application wants to enforce a rule or verification result based on our proposed endorsement. In that case, the verification must take place on-chain[8]. These processes are prone to third-party centralization if the verification takes place off-chain.

This shows that the varying problem statements, as initially defined in Section 3.2, lead to two conflicting goals: **on-chain verification** for systems that require that data to be present on-chain for decision making (e.g., who should be allowed to validate in a Proof of Authority (PoA)[9] network), as well as **off-chain verification** for processes that aim to secure the connection of Web 2.0 interfaces with the blockchain ecosystem, whether in the checkout or during other cryptocurrency-related activities.

These two approaches lead to different data storage requirements and fundamental approaches. We discuss the off-chain verification approach in Chapter 4 and the on-chain verification approach in Chapter 5.

### Data Storage

In our hypothetical system, there are different possible arrangements for the storage location of the data managed within the system. We need to define data storage locations for the following types of data:

- **Signature/Endorsement**: The cryptographic data that binds endorsers to endorsees needs to be stored publicly for later verification.
- **Leaf and Intermediate Certificates**: In a hierarchical PKI, the validity of intermediate and leaf certificates can be deduced by the root certificates, but the verifier needs to be able to access this data to verify the integrity and authenticity of any endorsement. As a certificate also contains the certificate chain, we do not differentiate between leaf and intermediate certificates.

[8] As we outlined in the previous section on requirements, we require decentralization or trustlessness within our blockchain applications. Of course, one could employ an oracle that pushes information about the validity of the endorsement on-chain, but that would introduce an unnecessary point of centralization. To avoid repetition, we do not repeat this argument in later sections of this thesis.

[9] PoA networks are usually somewhat centralized, as the entities running the network are predefined.

- **Root Certificates**: Root certificates also need to be stored for verification purposes. As these are critical to the trust, and as the validity of the verification result depends on them, they need to be securely stored.

- **User Preferences**: The user's preferences must be considered, including which root certificates to trust or any caching data speeding up the verification of endorsements.

Generally, we can store data either a) on the relevant blockchain ("on-chain"), b) in the WWW[10], or c) locally. The recommended location depends heavily on the requirements and the underlying use cases. While from a combinatorial perspective, many possible arrangements are possible, only a few make sense. We give an overview of the possible arrangements in subsection 3.3.2.

Below, we argue which potential storage solutions make sense for each type of data.

**Endorsement/Signature.** The endorsement and signature represent the binding between a certificate, its properties, and a blockchain address. As anybody should be able to access the endorsement for verification purposes, local storage is excluded. As these endorsements are newly created and did not exist previously, we cannot leverage already-existing storage within the WWW.

Two options remain: store the endorsement in the WWW or on-chain. In case of on-chain verification, the endorsement must also be stored on-chain; otherwise, the on-chain verification smart contract cannot access it. In case of off-chain verification, it is possible to obtain the data from other sources in the WWW. Only the DNS or the respective web server should be accessed to avoid introducing additional centralization. As the certificates of regular PKIs cannot be directly accessed (only via a directory service such as CT), a PKI is unsuitable as a potential place for storage. DNS itself is able to store the data, but as the unmodified DNS does not offer integrity protection and, therefore, cannot be verified in an on-chain context, it also fails as a proper means of storage. DNS could be a viable option if verification of the endorsement takes place off-chain and DNSSEC is leveraged to ensure the authenticity of data stored in DNS; ENS leverages DNSSEC to establish a binding between domains and a blockchain address. Still, the result is stored in the blockchain, as ENS functions purely on-chain. We also do not recommend storing the endorsement on the web server. While it is possible to access

[10]Potentially, the required data is already available in the WWW and we do not need to store it separately, e.g., on the respective web server.

the relevant data in an off-chain-verification context (e.g., in a `/.well-known/`-folder)[11], the web server is an often-selected attack vector in tricking users into interacting with malicious addresses (see Section 1.2). An adversary that gains access to the web server would also be able to access the respective folder and replace the endorsement. We also decide against storing additional data on the web server to eliminate this attack vector.

To conclude: All endorsements, whether leveraged in an on-chain or off-chain verification process, are stored **on-chain**.

**Leaf and Intermediate Certificates** The web server serves the leaf and intermediate certificate to the browser when it requests the domain. We must consider the possibility that multiple certificates exist for the same domain. These certificates exist because an entity might provide its services over multiple endpoints or content delivery networks (CDNs). Managing these certificates on an individual basis requires less maintenance and synchronization. Only the relevant service must be updated if a certificate needs to be replaced or revoked. Another reason for the existence of multiple certificates is that an attacker may have issued additional certificates, impersonating the web server.

Similar to the endorsement, three options exist: **Local storage** is not an option, as the number of certificates is unclear and constantly changing. Storing all relevant certificates locally is logistically neither practical nor possible. As the certificates are part of the **WWW**, serving via both the web server itself and other means, such as certificate transparency, the verification service can access them directly. Due to the determinism of blockchain networks, this is not possible in an on-chain context. Therefore, if on-chain verification occurs, the leaf and intermediate certificates must be stored on-chain.

The endorsement must refer to the relevant certificate when multiple certificates exist. In this case, the verification service accesses CT to obtain the specific certificate, as outlined in Section 4.2.6.

To conclude: Leaf and intermediate certificates are stored "inherently" in the response from the respective web server or in CT if the server does not return the matching certificate in an off-chain-verification process. These certificates must be stored in an on-chain context in an on-chain verification process.

**Root Certificates** Root certificates represent the single point of trust in a PKI. Therefore, these certificates are crucial to any verification process. In contrast to the short

[11] RFC 8615 defines well-known Uniform Resource Identifiers (URIs) for accessing site-wide metadata. This data is placed within a `/.well-known/`)-folder (Nottingham, 2019).

lifespan of intermediate or leaf certificates, root certificates have a long validity period[12]. Further, their number is limited. Google stores 140 root certificates in its Chrome browser[13], whereas the team behind Firefox accepts 168 root certificates in its browser[14].

Similar to the leaf and intermediate certificates, in an on-chain verification context, the root certificates need to be present in the on-chain environment. Otherwise, verification will fail. In an off-chain verification process, root certificates are not solely distributed via a regular HTTP request or within CT; they are part of the operating system or the browser, e.g., Google Chrome or Mozilla Firefox. If the off-chain-verifier maintains or has access to such a list, retrieval via WWW is unnecessary.

To conclude: In an on-chain verification environment, root certificates must be stored on-chain, whereas root certificate lists can be accessed locally for an off-chain verifier.

**User Preferences.** The verifier, in either an on-chain or off-chain context, must be able to define which root certificates to trust. We previously defined in Section 3.3.1 that our system needs to support an arbitrary number of PKI systems. This results in a situation in which users or verifiers do not trust all (root) certificates in the system. Therefore, they need to be able to define which certificates they trust.

Again, in an on-chain verification process, this information needs to be present at verification time; otherwise, it is impossible to decide whether a certificate or endorsement can be deemed valid. In an off-chain verification process, the list of allowed root certificates can be stored *inherently* in the respective root certificates, as the users controlling the service can decide which certificates to trust. Storage in the WWW does not apply in this case.

To conclude: In an on-chain verification environment, root certificate preferences are stored on-chain (or provided just in time), whereas for an off-chain verifier, preferences are stored locally.

### 3.3.2. Overview of Solution Space and Relevant Arrangements

In the previous section, we identified several key components and their potential design for a system that allows the leveraging of PKI certificates in a blockchain-based context.

[12] As of the 6th February 2023, the *DigiCert Global Root CA* root certificate, that is the root certificate in *example.org*'s TLS certificate, is valid until the 10th November 2031.

[13] Data taken from `https://chromium.googlesource.com/chromium/src/+/main/net/data/ssl/chrome_root_store/root_store.md`, accessed on 10th October 2022.

[14] Data taken from `https://ccadb-public.secure.force.com/mozilla/IncludedCACertificateReport`, accessed on 10th October 2022.

These components are as follows:

- Public-Key Infrastructure
- Processes
  - Issuance and revocation of endorsements
  - Verification of endorsements
- Data
  - Endorsements
  - Leaf and intermediate certificates
  - Root certificates
  - User preferences

Our system is agnostic to the type of PKI and depends only on whether the respective cryptographic key mechanisms are supported. Processes and data storage can occur locally, on the WWW, or in the blockchain network.

To account for the complexities of smart contracts and blockchain networks, we extend the on-chain context and introduce *centralized* and *standalone* smart contracts as subdivisions.

- Protocols often use **logically centralized** but technically decentralized smart contracts, as this saves execution costs and enables the proper definition of rules and guidelines. While there are smart contracts that serve as gate-keepers to their intended functionality (e.g., deciding whether a certificate is valid), the immutability of the smart contract protects the logic from changes and thus does not allow anyone to manipulate the smart contract once it is deployed[15]. This means that entities can verify the integrity of smart contracts and then decide to use them, as there is no way for them to have been manipulated. From this perspective, the system can be seen as its own protocol.

[15]This describes an ideal situation; in most cases, some form of update mechanism in form of a proxy smart contract is in place to allow enhancing the protocol or removing bugs. However, this update mechanism also introduces an attack vector. See OpenZeppelin's Proxy Upgrade Pattern for more information: `https://docs.openzeppelin.com/upgrades-plugins/1.x/proxies`. Accessed on 6$^{th}$ February 2023.

| | | | Data | | | | Processes | |
|---|---|---|---|---|---|---|---|---|
| | | | Endorsement | Leaf / Intermediate Certs | Root Certs | Preferences | Verification | Issuance / Revocation |
| Location | On-chain | Centralized | B) | B) | B) | B) | B) | |
| | | Standalone | A) | | | | | |
| | Off-chain | www | | A) | | | | |
| | | local | | | A) | A) | A) | A) & B) |

**Figure 3.2.:** Overview of all dimensions and objects of the hypothesized system. System A) allows for off-chain verification, whereas system B) allows for on-chain verification. Horizontal lines mark configurations that deplete security, and vertical lines mark technically possible configurations that add additional costs. Cells marked with a dark color are impossible configurations.

- **Standalone** smart contracts provide a way to define the structure and interfaces of a smart contract to adhere to a specific standard. This is often useful when no direct interaction with other smart contracts takes place or in cases in which centralization is neither possible nor cost-beneficial. From this perspective, the system can be seen as an interface standard (e.g., comparable with the ERC-20 token interface standard)[16].

We provide visualization in Figure 3.2 of all dimensions and elements of our hypothesized system. This enables us to discuss potential arrangements and why only two possible system designs make sense.

Our analysis shows that there are two distinct ways to leverage certificates from PKIs in a blockchain-based context: the first is **off-chain** verification with **off-chain** leaf, intermediate and root certificates, and the second is **on-chain** verification with **on-chain** leaf, intermediate and root certificates. This shows that in the case of on-chain verification, all relevant information needs to be on-chain, whereas, in an off-chain verification scheme, we can leverage off-chain data.

**Impossible Configurations.** Figure 3.2 displays impossible configurations in black. Noteworthy is the issuance and revocation of endorsements that cannot occur in a

[16]ERC-20 is an interface standard defining which functions a smart contract needs to expose to be considered as a token contract. This standardization allows other software vendors to directly integrate any token instead of adapting their code to individual types of tokens. OpenZeppelin provides a reference documentation: `https://docs.openzeppelin.com/contracts/4.x/api/token/erc20`, accessed on 15th October 2022.

blockchain-specific context or on a third-party service. As endorsements are based on the cryptographic key material of X.509 certificates, only the party that possesses the relevant certificate can issue an endorsement. The verification of an endorsement also cannot be implicitly outsourced to services in the WWW; building and enabling access to a service that verifies the endorsements is certainly possible but would require extra effort compared to leveraging existing infrastructure.

Endorsements and leaf and intermediate certificates cannot be stored locally ahead of time (e.g., as part of the software package) and therefore need to be retrieved just in time.

**Economically Unsustainable Configurations** Leaf, intermediate, and root certificates, including preferences, could be organized in standalone smart contracts that host the certificates' respective data. Also, data could be verified in standalone smart contracts redeployed for every interested entity. While technically possible, these decisions would add high costs to the system as they introduce redundancy by storing data repeatedly, as every smart contract needs to be able to access all data required for verification. Additionally, these decisions would prevent efficiency gains that would otherwise be possible due to the PKIs' hierarchical structure: The verification of intermediate and leaf certificates should be executed only once. Due to the immutable data structure, users can ensure that once-verified certificates remain valid[17].

**Depleting Security** Storing the endorsement in a WWW environment is discussed in Section 3.3.1. Insecure web servers are precisely the reason for address replacement attacks, which swap a valid smart contract address with an address under possession of the attackers. Storing the endorsement on the same machine does not provide an additional layer of security, thus depleting the system's security.

The same applies to storing or leveraging a third-party web service's root certificates and user preferences. While lists of trusted certificate authorities can certainly be leveraged (e.g., from Google Chrome, Mozilla Firefox, or the operating system), we do not recommend accessing these lists on-demand, as we see no advantage to this.

[17]The validity of a certificate depends not only on the correctness of the signature but also on ecosystem variables such as the date and revocation status. The on-chain smart contracts must cater to and consider these variables in the verification process.

## 3.4. Endorsement

In this section, we introduce the concept of an endorsement, as it is a crucial pillar of the architecture and applies to both approaches, as outlined in Chapter 4, *"Off-Chain Verification"*, and Chapter 5, *"On-Chain Verification"*. An endorsement can be defined both by its representation and functionality and from a technical perspective, i.e., the structure and data. First, we define the actors and entities in the system.

### 3.4.1. Entities

- **Endorser**: An endorser is an entity that controls and manages a certificate in a PKI. A given certificate contains properties that are attributed to an endorser (e.g., a domain name). This endorser intends to extend its property to an endorsee, which in turn can claim an association to the respective certificate and, thus, the respective property.

- **Endorsee**: An endorsee is a unique and securely[18] verifiable object on a blockchain network within (and potentially outside) a given ecosystem. An endorsee can be a regular address (e.g., `0x123...789`) controlled by an externally-owned account or a smart contract. In theory, any other type of identity or different object in these networks could be an endorsee (e.g., an ENS name or a non-fungible token). For the sake of this thesis, we limit endorsees to addresses, meaning externally-owned accounts and smart contracts.

- **Verifier**: The verifier is an entity, either within or outside the blockchain network, that intends to verify the integrity of the endorser-endorsee relationship and, thus, the endorsement itself. It relies on data from the originating PKI and the endorsement. The verifier can be a service off-chain, e.g., a regular service that connects to the blockchain to fetch data, or an on-chain service in the form of a smart contract.

[18] Secure in this context means that this object is either able to produce verifiable messages, either by cryptographic means such as signature schemes or can create messages that are the result of a well-defined deterministic computational process, e.g., such at it would be the case for a smart contract execution.

## 3.4.2. Definition

The term *"endorsement"* was first used in (Gallersdörfer and Matthes, 2020) to refer to a simple means by which a domain owner can *endorse* a smart contract, with no formal definition. In (Gallersdörfer and Matthes, 2021a), we gave a formal definition of an endorsement, consisting of a claim $C$, a signature $S$, and an endorsement $E$. (Groschupp, 2020) and (Gallersdörfer et al., 2021c) also provide definitions of *"endorsement"* which deviates slightly from the initial definition in (Gallersdörfer and Matthes, 2021a). We rely on the initial definition given in (Gallersdörfer and Matthes, 2021a).

To better understand the concept of an endorsement, we divide its definition into functional and technical definitions.

### Functional Definition

An endorsement represents an *inclusive* reference from a domain owner to a blockchain address, extending the scope of its web server identity to on-chain blockchain addresses or applications. In this way, the endorser broadens the scope from the certificate to the endorsee. That means that domain owners associate their respective identities with the addresses endorsed. Users and other entities accept the endorsee and endorser as a union and recognize both forms, e.g., the website and the blockchain address, as different means of communication, exchange of information, or value.

It is important to note that endorsements *should* be considered positive references, as an attacker could create endorsements for addresses it does not control. While an attacker cannot gain control over endorsed addresses, scenarios can be constructed in which an attacker could harm the reputation of an on-chain address by associating it with illicit activity. For example, Tornado Cash, a coin tumbler[19], is sanctioned by the US government. Miners and validators in the US need to comply with this sanction list, forcing them to censor related transactions in the Ethereum network. An anonymous user sent funds to popular addresses through Tornado Cash, which could potentially result in censoring of these addresses (Sun, 2022).

[19] A coin tumbler is an application that allows users to disguise the origins of their funds by collecting funds from multiple entities and distributing them back to the respective entities in a separate step. We use the term coin tumbler (in contrast to *coin mixer*), as the pure functionality of this application does not make a statement about the lawfulness of the respective activity (Narayanan et al., 2016).

### Technical Definition

"We [...] define three key components [...]: the claim, signature, and endorsement.

First, we define the claim $C$ in Eq. 3.1:

$$C = \{addr, domain, date_{exp}, flags\} \tag{3.1}$$

The claim $C$ contains the address $addr$ of the endorsed smart contract, the FQDN $domain$, the expiration date $date_{exp}$, and further flags defined in $flags$. [...] An FQDN $domain$ is provided as-is, and protocol information is omitted, e.g., *hq.example.org*. Furthermore, the expiration date $date_{exp}$ is defined as Unix epoch time.

We define a valid signature $S$ in Eq. 3.2:

$$S = \{sign(hash(C), cert_{privKey})\} \tag{3.2}$$

The signature $S$ contains the claim hashed with a cryptographic hash function signed with the private key $cert_{privKey}$ of the TLS certificate of the respective domain.

We further define an endorsement $E$ in Eq. 3.3:

$$E = \{S, C, [cert_{fingerprint}]\} \tag{3.3}$$

The endorsement $E$ contains the previously defined signature $S$, the claim $C$ required to validate the signature, and the optional fingerprint of the signing certificate $cert_{fingerprint}$. We include the fingerprint of the certificate to facilitate its retrieval if the certificate is not present at the web server, as described subsequently in Section 4.2.6.
" – "TeSC: TLS/SSL-Certificate Endorsed Smart Contracts" (Gallersdörfer and Matthes, 2021a).

Flags $flags$ were initially proposed in (Gallersdörfer and Matthes, 2021b,a) as a way to enhance, extend or limit the functionality of an endorsement. Given the many variables in the verification process, the idea is to allow the endorsers to configure their endorsements

to their needs. For that, endorsements can be equipped with flags. These flags are also part of the signature, such that the smart contract cannot unilaterally extend the scope of the endorsement. Although the system supports 191[20] flags, only nine have been defined so far. This allows for the extensibility of the system, such that once deployed, smart contracts can support more recently introduced features. We reiterate the previously defined flags in (Gallersdörfer and Matthes, 2021b) in Appendix B.

The following structure gives insights into each variable, including data type and exemplary values:

- *addr*: This represents an address of an on-chain entity, either an externally owned account or a smart contract. In the case of Ethereum, the address starts with `0x` and has 42 additional characters. An example of an address is `0x05a5...4d9f`.
- *domain*: This is the FQDN of the endorsing entity, including subdomains and omitting the protocol. An example of a valid domain is `subdomain.example.org`.
- $cert_{fingerprint}$: The certificate ID needs to be unique for every certificate; this is not the case for the serial number, as, in theory, two CAs can issue a certificate with the same serial number. Therefore, we rely on the SHA-256 fingerprint of the certificate as a unique identifier[21]. An example is `5E:F2:F2:14:26:0A:B8:F5:8E:55:EE:A4:2E:4A:C0:4B:0F:17:18:07:D8:D1:18:5F:DD:D6:74:70:E9:AB:60:96`[22].
- $date_{exp}$: An expiry date is provided in case the endorsement should have a limited validity. This information is given as a UNIX time stamp, for example, `1703372400` equals to December 24, 2023, 00:00 (GMT+1).
- $sign(hash(C), key_{privKey})$: The signature over the claim (including the address, domain name, certificate identifier, and expiry date) is crucial for verifying the integrity and authenticity of the claim. This information is given as `bytes`. We rely on SHA-3 and the ECDSA-signature scheme within our systems for fast and cost-efficient verification.
- $flags$: Flags are stored in a `bytes24` variable, allowing for the definition of up to 192 binary flags (Gallersdörfer and Matthes, 2021b).

[20] One flag is always set to `True` for integrity purposes.
[21] For SHA-1, attack vectors exist to generate pre-image collisions (Stevens et al., 2017). Therefore, we stick to SHA-256 despite its larger output size.
[22] Fingerprint of the certificate of `example.org`, accessed on 6th February 2023.

## 3.5. Summary

This chapter described how a PKI and its information could be utilized in a blockchain context. Further, we illustrate a solution space containing the PKI's application in different contexts.

Using attributes of X.509 certificates managed in PKI is key to the concept. Although this concept supports any given X.509-based certificate structure (e.g., enterprise-internal structures), we focus on the domain name attribute of X.509 certificates, usually utilized in a WWW setting to enable authentication and secure communication.

Given the complexity and criticality of the underlying technologies, we define a set of requirements for the system design: First, we outline use cases and include further requirements for the system design. In particular, openness and the absence of trust requirements (in addition to already-existing trust requirements, e.g., in the integrity of the certificate authorities) play a significant role in the respective systems.

Second, we analyze respective RFC documents to align with the specifications stemming from the existing infrastructure. Many of the explicitly mentioned requirements, such as **authentication**, align with the concept and goal of the proposed systems, while other requirements, such as **confidentiality**, are in stark contrast to the transparency principle of blockchains. Furthermore, the underlying infrastructure components' life cycles are considered when designing the systems.

Given these sets of requirements, we define two potential system designs, one enabling the verification of information in an on-chain context, which in turn enables the usage of certificates and respective signatures as elements of the deterministic execution environment. In contrast, the other approach allows for the verification of such information in an off-chain environment, enhancing user-facing software, such as wallets, by providing additional details on their counterparties.

Lastly, we introduce and coin the concept of *endorsements*. As the acting entities in blockchain networks are accounts that can be identified by addresses, which are controlled either by private keys or code executed on request, a binding between blockchain-based entities and the entities controlling certificates in PKI is required. Endorsements constitute a connection between the owner of the domain (as the entity that controls the X.509 certificate to the domain) and the account that resides in the blockchain network.

In the next chapter, we introduce and discuss the off-chain approach. Resolution and verification of the domain-address relationship takes place off-chain, for example, in a browser.

# Chapter 4.

# Off-Chain Verification

Entities interacting with third parties risk being deceived regarding the authenticity of on-chain addresses. The following chapter proposes the first architectural design to mitigate the issues of loose coupling between the WWW and blockchains by enabling the on-chain storage of endorsement information and establishing an off-chain verification method. In more detail, we divide this chapter into the following sections:

- Section 4.1, *"Problem Statement"*, describes the issues resulting from loose coupling between websites and blockchain addresses and discusses examples of attacks that have occurred in the real world.
- In Section 4.2, *"System Architecture and Processes"*, we establish the architecture, the individual components, and the relevant processes for enabling TLS-endorsed smart contracts and their verification.
- We provide an understanding of the threat model regarding the underlying technologies as well as the interplay between TLS and endorsements stored on-chain in Section 4.3, *"Threat Model and Security Implications"*.
- Lastly, in Section 4.4, *"Augmentation of User Wallets"*, we give insights into the augmentation of the user wallet *MetaMask* to support and recognize TLS-endorsed smart contracts. We discuss previous work, revisit fundamentals for the design, and present drawbacks and potential mitigation strategies.

The first three sections of this chapter are based partly on and rely on the material of two prior publications, namely "TeSC: TLS/SSL-Certificate Endorsed Smart Contracts" (Gallersdörfer and Matthes, 2021a) and "AuthSC: mind the gap between web and smart contracts" (Gallersdörfer and Matthes, 2020). Supplementary material in the form of

Github repositories exists in (Gallersdörfer and Matthes, 2021b). The author of this thesis implemented the interface specification of the smart contract. Under his supervision, the implementation of the smart contract was conducted by the authors in (Herrmann et al., 2021).

In Section 4.4, we reiterate prior work by (Ebel, 2021) and (Gallersdörfer et al., 2021a) for which supplementary material also exists in (Gallersdörfer et al., 2021b).

## 4.1. Problem Statement

In the following, we further examine two different problem statements that exist at the intersection of blockchain networks and Web 2.0 infrastructure, namely the address replacement attack and missing data authentication, both of which can be addressed by an off-chain-verification approach, as outlined in this chapter. Both issues have previously been raised in subsection 3.2.1.

### 4.1.1. Address Replacement Attack

In (Gallersdörfer and Matthes, 2021a), we originally coined the term *"address replacement attack"*. An address replacement attack aims to trick the user into using a different address than initially intended. Users obtain data about the address they will interact with (e.g., a smart contract or an externally owned account) from sources outside the blockchain, such as websites. Websites either directly display the address or hide the address within a *web3.js*-application[1] that connects to the user's in-browser wallet or other add-ons, such as MetaMask (MetaMask, 2023). The reason that address replacement attacks exist is that the relationship between the source and the address is not authenticated. Attackers with access to the trusted source can replace the contents and link to a different address that they control.

This type of attack can be labelled as follows:

- **Lost credentials** or **defacing** (Lazarenko and Avdoshin, 2018), as the attackers can often obtain credentials and are therefore able to manipulate contents, or
- **Man-in-the-middle attacks**, as the attacker can interfere with communication and act as a man in the middle.

While these definitions may fit specific attacks, we think that they are either too unspecific (e.g., *man-in-the-middle*) or describe only one potential attack vector (e.g., *lost-credentials*); therefore, we opt for the term *address replacement attack*.

#### Attack Vectors

There are multiple attack vectors or pathways by which to successfully execute an address replacement attack. We iterate over the most relevant scenarios, give examples of them,

[1] web3.js is a JavaScript framework that allows the website to connect to the wallet of the user and initialize interactions, such as executing transactions or signing data.

and explain whether protecting the relationship between website and smart contract would prove successful in preventing or mitigating damage.

- **Active Phishing Directed at End Users:** While all forms of deception can be seen as phishing attempts, we differentiate between phishing based on sources besides the website, as it is our main security goal to protect the relationship between the website and smart contract. Phishing attacks via social media channels or other forms of communication (e.g., via Telegram or Discord), actively reach out to users that are part of the community in question and try to trick them into interacting with a website or smart contract.
  In 2022, *Bored Ape Yacht Club* (BAYC) suffered both an attack on its Instagram and their Discord channels, in which accounts were hacked and used to promote new tokens to holders of the previous tokens, leading to a loss of tokens worth several million USD[2].
  As attackers are directly in control of the links that a user might click on, protecting the relationship between the website and the smart contract misses the goal, as both parts are in full control of the attacker. The attacker can reassure the credibility of the domain name via Typosquatting, for example, registering domain names with small changes (e.g., a lower-case `l` vs. an upper-case `I` vs. the number `1`).

- **Web Server:** The web server is the main target for attacks. Attacks often take place either on regularly visited sites (e.g., the user has to regularly interact with a smart contract) or in times of high demand, such when a new token launches. As attackers are able to gain access to either the web application itself (e.g., due to a *security exploit*) or to the file system (e.g., due to *stolen credentials*), they are able to manipulate the contents of the website, misleading users and potentially stealing funds.
  The *Coindash ICO* took place in July 2017, selling tokens to users in exchange for the cryptocurrency Ether. At launch time, the original address was replaced by hackers, resulting in 43,000 Ether being sent to an address controlled by the attackers. On Coindash's blog, there is an indication that an attacker was able to place a web shell on the server using WordPress[3].
  The attackers had direct access to the web page and were able to manipulate its

[2] `https://therecord.media/bored-ape-yacht-club-says-instagram-hacked-nfts-stolen/`, accessed 25th October 2022.
[3] `https://blog.coindash.io/coindash-tge-hack-findings-report-15-11-17-9657465192e1`,accessed 25th October 2022.

contents, leading to the loss of user funds. Protecting the relationship between the web site and the address on the blockchain might have prevented the attack. We discuss the details of the security model and mitigation strategies in Section 4.3.

- **DNS Hijack:** In case the web server cannot be directly attacked, an alternative might be to attack the DNS server that is maintaining a record of the IP to domain linkage. Such attacks can be the result of stolen credentials or actions of insiders within DNS registrars. The IP address in question is replaced by another web server that serves identical content to the original one to avoid notifying users, whereas only crucial information, such as the address, is changed.
  A number of decentralized finance (DeFi) applications were attacked in June 2022, among them *ConvexFinance*, *Ribbon Finance*, *DeFiSaver*, and *Allbridge*[4]. Apparently, a service agent at the DNS registrar *NameCheap* was able to manipulate the applications' respective DNS entries, pointing to different web servers. Besides the DNS spoofing, the attackers also created vanity addresses that contained the same four letters at the beginning and end of the addresses[5].
  An attack at the DNS level is not directly apparent, given that the original web server remains intact. Protecting the connection between the domain and the smart contract (or making that connection apparent) might not have prevented the loss of user funds but could potentially have detected it at an earlier point of time if proper monitoring were in place as we outline in Section 4.3.

### Potential Damage

Overall, the aim of the attacker is to drain the funds of the victim. This damage can be inflicted in two ways, as the owners or victims control their assets in one of two ways: either directly stored in their address (e.g., cryptocurrency that has been sent to the respective addresses) or indirectly (e.g., assets that are stored in a smart contract) but are controlled by the victims.

This creates to two potential ways for the attacker to obtain respective valuable assets:

- **Misdirecting user funds:** One way for attackers to separate users from their money is to misdirect funds. A user may want to execute a transaction that involves

[4] `https://www.trustnodes.com/2022/06/25/defi-dapps-dns-attacked`, accessed on 25th October 2022.

[5] The attackers generated a vanity address for `0xF403C135812408BFbE8713b5A23a04b3D48AAE31` and were able to generate the address `0xF403a2c10B0B9feF8f0d4F931df5d86aD187AE31`. A vanity address is an address that contains specific, predefined characters. The more characters are defined, the longer it takes to compute.

a monetary component, such as "*I want to invest in this upcoming ICO*" or "*I want to buy a specific asset*". These funds are controlled by the victims **directly**. This intention of the user is then misdirected by replacing the respective address with an illicit address, similar to the case of the *Coindash ICO*.

- **Illicit signature creation:** Another way for attackers to steal users' funds is to trick them into signing transactions that can later be used to retrieve assets that the victims control **indirectly**. Often, it is the case that users already have some connection to the website or application that they intend to use. In this case, the application might not directly redirect the users' actions to another address but change the contents of the transactions such that they benefit the attacker. For example, users could accidentally sign transactions that allow the attacker to move the tokens in an ERC20-contract away from users, basically stealing their funds. This type of attack scenario took place in the above mentioned *BAYC* case.

In both cases, we must ask whether a hardened relationship between a website and the respective smart contract could have notified the user of the attack and led to the abortion of the transaction? In the first case it is straightforward to decide that a) if a smart contract does not belong to a website and b) smart contracts with such a relationship to this website exist, then it is probably an illicit address (or the operators of this service forgot to make that relationship explicit using the proposed method in this thesis). In the second case, the detection or even prevention of this type of attack is very difficult; the address and its relationship to the website are valid, and the type of transaction can be arbitrary. Our system is not designed to verify the contents of a transaction, only its recipient. While such an attack could be facilitated by above mentioned attack vectors, our system is not capable of preventing such attacks. Helping users to understand the contents of their to-be-signed transactions is an ongoing endeavor and warrants its own dedicated research (Willmore, Joel, 2022).

### 4.1.2. Missing Data Authentication

Lack of data authentication or data provenance is a problem that exist within decentralized networks and economies without central actors able to properly distribute identity or other reliable authentication information.

To give a more practical example, which will expand the problem of data authentication which we briefly described in Chapter 1. We briefly describe the use case of digital credentials for higher education.

Multiple groups exist that aim to develop a digital credentialing standard, for example the *"Digital Credentials Consortium"* (led by Massachusetts Institute of Technology)[6], *"DiBiHo"* (led by Technical University Munich)[7], and *"W3C Verifiable Credentials for Education Task Force"*[8]. The author of this dissertation had actively participated in all listed groups.

### Digital Credentials for Higher Education

Credentials from higher education institutions, such as universities or other internationally recognized entities, are often issued in analog form. Instead of taking the form of digital certificates that can be easily verified by other entities such as employers or other universities, these credentials need to be manually scanned, transferred, and verified or even translated and notarized by trusted entities.

The endeavor of digital credentials poses a multitude of problems: standardizing diploma formats, defining course descriptions and comparable learning units, catering for longevity of certificates (some certificates must be valid for the entire lifespan of a human being), legal recognition, and bootstrapping to support a critical mass of institutions and employers. While this thesis and its proposed system architecture cannot address these problems, it might be able to evince an approach to solving a different issue: the authenticity of information and its issuing entity. To be more concrete: *"How do we know that the entity issuing a certificate is actually the entity that we think it is?"*

In discussions within these above mentioned groups, blockchain technology was suggested as a potential solution to enable longevity of certificates (or their integrity information) and data provenance. The issue of verifying the authenticity of the issuing entity remains. There are multiple proposed solutions to ensure the authenticity of issuers:

- **Permissioned:**
  - **Preselecting issuers:** A set of entities, e.g., higher education institutions, could form a group that specifies, defines, develops, and adopts a system that allows the digital issuing of certificates. By preselecting issuers, this group asserts its trust to the respective issuer and ensures that only valid issuers are able to issue credentials and that only credentials by these issuers can be

[6] See `https://digitalcredentials.mit.edu/`, accessed 30th October 2022.
[7] "Digitale Bildungsnachweise für Hochschulen" (German, "Digital education certificates for universities"), see `https://www.it.tum.de/it/dibiho/`, accessed 30th October 2022.
[8] See `https://w3c-ccg.github.io/vc-ed/`, accessed 30th October 2022.

verified correctly. While the preselection of issuers is a strong guarantee for the validity of the issued credentials, it comes with considerable downsides: A) the on-boarding and continuous management of new issuers is time-consuming and costly, B) erroneous accepted issuers might be able to erode the trust in the system, and C) other institutions would need to trust this initial group of entities, partly giving up their sovereignty over their credentialing processes.

  - **Trusted governments:** A setting sufficient to handle this problem would be for each state, which already has the competence to define institutions of higher education, to provide an infrastructure for these institutions to issue their credentials or at least provide them with verifiable identities that enable the issuance of signed arbitrary data. With that, both the competency of defining such institutions, as well as equipping them with the relevant key material, is within the hands of one entity, the state. Different states, such as the European Union, are piloting this form of infrastructure[9], although it is unclear at the time of this writing (October 2022), when such services will be broadly available.

- **Permissionless:** A permissionless approach without a gatekeeper or reliance on an identity infrastructure that is already broadly available seems to be a valid alternative to permissioned solutions, as outlined above. A pure permissionless approach might face the initial problem that we set out to solve: the lack of authentication of the issuer. However, relying on a technology that provides something like a *pseudo-identity* might mitigate the problem. At this stage, TLS and DNS might be able to provide the type of information required for certificate-verifying entities to understand whether an issuer is valid or not. Furthermore, websites and other forms of fully interactive services (that already integrate TLS and DNS) were ruled out due to the requirement of *"[Enabling] seamless verification without involvement of the issuer"* (Chartrand et al., 2020).

### Solution Approach

For reasons of clarity, we do not discuss the individual components of a system that allows the issuance, verification, and revocation of digital credentials of higher education institutions. The structure of the architecture and the processes involved, which we

[9] Compare *European Blockchain Services Infrastructure*: `https://ec.europa.eu/digital-building-blocks/wikis/display/EBSI`, accessed 30th October 2022.

define and discuss in the following sections, are generalized, meaning they are applicable to any form of use case that aims to solve or mitigate the above-described issue. Whether higher-education certificates, other forms of certifications, data in data marketplaces, or another form, the steps and processes involved in enabling a given use case are identical:

- **Definition of life cycle of underlying data:** The underlying data and its contents need to adhere to specific rules and processes. For example, credentials from higher education institutions need to be issuable, revocable (e.g., in cases of erroneous issuance), verifiable, and auditable. This type of lifecycle needs to be integrated within a smart contract functionality, such that other parties are able to retrieve and verify the respective information in accordance with predefined rules.
- **Smart contract creation and deployment:** The smart contract needs to be augmented with the functionality described in the following sections: The inclusion of the functionality ensures that later processes can verify, in addition to the rules for the data itself, the integrity and validity of the endorsement, with the ability to attribute the underlying data to the respective entity.
- **Data retrieval and verification:** The software that retrieves and verifies the respective data needs to be able to connect to the blockchain, retrieve the smart contract and its contents and verify a) the integrity and validity of the underlying data according to its predefined rules, b) the validity of the endorsement, and c) any leaf, intermediary, or root certificates that are involved in the process of issuing the data in the first place.

## 4.2. System Architecture and Processes

> "In this section, we introduce the general approach and give an overview of the architecture, [*the endorsement*], the single components, and the smart contracts structure. We further display processes of the system and design considerations.
>
> Our goal is to establish a binding between smart contracts and websites using common TLS-certificates [...]. The owner of a certificate endorses a smart contract by [...] storing an endorsement within the respective smart contract. Later, other parties can retrieve the endorsement from the blockchain and

validate it against their trusted certificate authority servers, thereby authenticating the contract. Afterward, they can engage with the specific smart contract or further evaluate data stored in the smart contract, aware of the actual identity of the counterparty."

– "AuthSC: mind the gap between web and smart contracts" (Gallersdörfer and Matthes, 2020).

"Our system consists of following components:

- The **Endorsement** ([*already introduced in*] Section 3.4) holds information about the approval of a smart contract by the domain owner signed with the TLS certificate's private key. [*This information contains relevant data, such as the smart contract address and further information such as the expiration date or additional flags.*]
- The **On-Chain TLS Endorsed Smart Contract** (Section 4.2.1) contains methods for storing and updating the endorsement. In addition, we provide the reference implementation (Gallersdörfer and Matthes, 2021b) which enables independent usage from the actual smart contract (e.g., token contract or other purpose).
- **Off-Chain Verifier** (Section 4.2.2) is an application that runs outside of the blockchain. It is responsible for verifying endorsements stored on-chain and retrieves data from the certificate issuer, website owner, smart contract stored in the blockchain, and potential certificate authorities.
- **[...]** [***Endorsed Smart Contract***] **Registry** (Section 4.2.3) prevents downgrade attacks by providing a list of smart contracts for domains and enables easy discovery of smart contracts that implement this library. [*In addition, this component manages a list of existing smart contracts and their respective domains to be used for verification purposes.*]"

– "TeSC: TLS/SSL-Certificate Endorsed Smart Contracts" (Gallersdörfer and Matthes, 2021a).

Following the structure of (Gallersdörfer and Matthes, 2021a), we discuss the endorsement of pre-existing smart contracts[10] in subsection 4.2.4, and the revocation of

[10]Smart contracts that have existed before the design and development of this system.

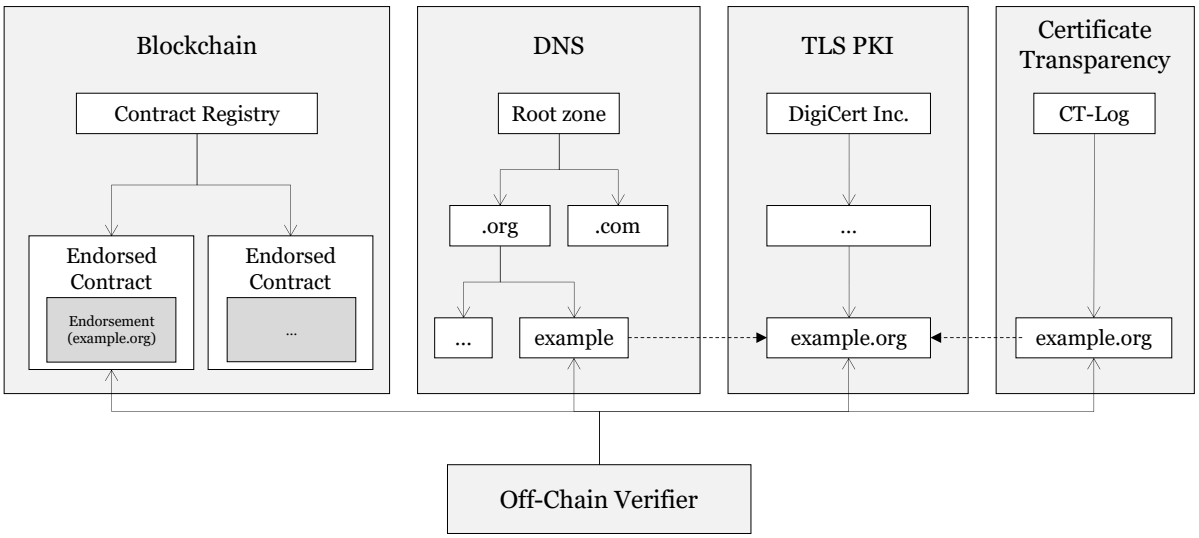



**Figure 4.1.:** High-level structure of the proposed architecture initially presented in (Gallersdörfer and Matthes, 2020, 2021b), adapted.

endorsements in subsection 4.2.5. We explain how we address the TLS ecosystems' technical specialties, such as key rotation and multiple certificates.

Figure 4.1 depicts the overall structure of our proposed architecture, initially presented in (Gallersdörfer and Matthes, 2020, 2021b) and adapted for this manuscript. We identify five main components: the blockchain containing both the endorsed smart contract registry as well as the endorsed smart contracts with the a given endorsement; the DNS, including its hierarchical naming scheme; the TLS PKI, including its hierarchical certificate structure; and CT, including its CT-log, as well as an off-chain verifier. The off-chain verifier has access to all these systems and is able to make an informed decision about the validity of an endorsement and the authenticity of the respective smart contract.

In addition to the architecture and processes, we propose an interface standard and highlight a respective reference implementation in Solidity[11], originally proposed in (Gallersdörfer and Matthes, 2021b,a). The implementation was conducted in (Herrmann et al., 2021) under the supervision of the author of this thesis.

### 4.2.1. On-Chain TLS Endorsed Smart Contract

The prospective endorsed smart contract needs to store the previously defined endorsement, including the domain name, expiration, and potential rules in the form of flags set

[11] Our reference implementation targets EVM-based blockchains. For any other blockchain network that supports the execution of smart contracts, the approach needs to be migrated to the respective language.

by the domain owner.

> "A domain owner claims the ownership of a smart contract by providing an endorsement in the respective contract. Information required to authenticate the smart contract is retrieved from external data sources. [...]
>
> Three steps are required to enable authentication of the smart contract: smart contract creation, endorsement generation, and endorsement upload. For simplicity, we omitted the intermediate or supporting activities such as creating wallet addresses, funding the accounts, and including the library in the respective smart contract.
>
> Initially, only newly created smart contracts which adhere to the interface standard can be endorsed. [...]. The to-be endorsed smart contract is supplied with the respective endorsement data. To create the endorsement, the owner of the domain retrieves the unique contract address and signs a respective claim consisting of the smart contract address, domain, expiration, and potential flags with its private key of the TLS certificate. Because the endorsement contains only one specific smart contract address, other smart contracts cannot use this information for fraudulent endorsement. The endorsement is then added to the smart contract via a regular method call."
>
> – "TeSC: TLS/SSL-Certificate Endorsed Smart Contracts" (Gallersdörfer and Matthes, 2021a).

- "**Smart Contract Creation**: [...] A contract has to be instantiated with information on the address of the owner of the smart contract and optionally the domain to which the owner has access to (e.g., the domain or any subdomain). [...]
- **Signature Generation**: The signature is still missing in the smart contract and thus, the TLS certificate has not endorsed this smart contract, as any party can execute the first step for any domain. The entity retrieves the unique contract address and signs this string with its private key of the TLS certificate. As the signature contains only one specific smart contract address, other smart contracts cannot use this information to pretend to be also endorsed by this domain.
- **Signature Upload**: The signature created in the previous step is transmitted to the smart contract as part of a regular transaction [...].

> The contract validates that the transaction is indeed created by the original owner of the smart contract, as the contract itself is not able to verify if the provided data is correct. If the owner issued the transaction, the payload is stored in the respective field of the smart contract, awaiting later retrieval.
>
> It is possible to reduce the process to two steps, as it is possible to sign the address of the contract before its initialization, providing the signature alongside the domain name. This is possible as the addresses of to-be generated smart contracts are deterministic. For simplicity and to remove possible interference with other transactions issued from the same address, we decided to first upload the contract and then actually submit the signature data, preventing that an incorrect address is signed."
>
> – "AuthSC: mind the gap between web and smart contracts" (Gallersdörfer and Matthes, 2020).

### 4.2.2. Off-Chain Verifier

"The verification of the smart contract endorsement occurs on the client-side. The software itself (e.g., a wallet in a browser) needs to access the following data sources:

- The **address** of the relevant smart contract, which is usually obtained via the Internet; alternatively it is obtained by the TeSC registry or proprietary discovery services,
- The **endorsement** of this smart contract, e.g., domain name, signature, and expiration date,
- the **signed certificate**, its public key alongside the information of the certificate authority, which is obtained by requesting the domain and retrieving the TLS certificate[12], and
- the **list of trusted certificate authorities** of the user.

The list of trusted certificate authorities is defined by the user, either by directly providing a list or by reviewing the CA list stored in the user's computer or browser.

[12] As we discuss in Section 4.2.6, the certificate can be obtained by other methods, e.g., Certificate Transparency.

Smart contract authentication involves four steps. Again, we omit intermediate steps and assume that the user journey starts on the website, such that the domain is known. Also, we assume that the contract address is known.

- **1. Smart Contract Endorsement Retrieval:** The application first retrieves the endorsement for later authentication. It contains information about the claim and the signature data of the smart contract.
- **2. Certificate Retrieval:** Afterward, it connects to the respective domain given in the smart contract (or that previously known by the website) and obtains the certificate. If the optional $cert_{fingerprint}$ is set, the certificate is directly retrieved from Certificate Transparency.
- **3. Smart Contract Endorsement Verification:** The software validates whether the private key of the certificate signed the smart contract claim and checks any additional properties of the claim, such as
  - 1) it first verifies that the address stored in the claim is identical to the smart contract address. Otherwise, the process is aborted.
  - 2) The domain in the claim has to be identical to the domain from which the information was obtained. If not, the process is aborted[13].
  - 3)The smart contract is endorsed only until the $date_{exp}$. If $date_{exp} < date_{today}$, then the smart contract is expired and should not be trusted.
- **4. Certificate trust:** After successfully verifying the endorsement, the software checks whether a trusted CA has (indirectly) signed or issued the certificate found in the smart contract and the web server. If a trusted CA signs the certificate and its public key, the identity is successfully validated. Otherwise, the program aborts with an error. It executes the certification path validation algorithm as defined in *RFC 5280* (Boeyen et al., 2008). To further verify the authenticity of the certificate, the verifier requires a proof for inclusion of the certificate into Certificate Transparency in form of a *signed certificate timestamp* (SCT) (Laurie et al., 2013)."

[13]The explicit statement of the domain is important because certificates exist that are valid for multiple domains or contain domain wildcards. If the domain would not be stored within the claim, the owner of the smart contract could decide on its own to which domain it belongs.

– "TeSC: TLS/SSL-Certificate Endorsed Smart Contracts" (Gallersdörfer and Matthes, 2021a).

The process of authenticating a smart contract is depicted in Figure 4.2.

### 4.2.3. Endorsed Smart Contract Registry

"For the client-side verification it is relevant to know 1) whether a contract exists for a given domain and 2) under which address it is deployed. To find such contracts or check its existence, we propose extending this architecture by introducing a smart contract registry that enables the user to find all domains registered on the blockchain. This registry lists all contracts that adhere to this interface standard and claim to be the identity contract for a specific domain. We allow multiple endorsed smart contracts for the same domain because one domain can endorse multiple smart contracts. Even though it is possible to search the complete blockchain for such contracts (Fröwis et al., 2019), it is easier and faster to use an on-chain smart contract.

The registry smart contract is in place to map domains to smart contract addresses by storing these relationships. The usage of such a contract involves the following steps.

1. **Insertion of Contract Information:** All parties in control of endorsed smart contracts submit information about their contract and optionally the respective domain to the registry smart contract.
2. **Domain Lookup:** A user searching for a domain queries the registry smart contract and asks for all contracts assigned to that specific domain. The contract returns all relevant contract addresses.
3. **Contract authentication:** The client can execute the previously described authentication method for each of the returned contracts, which ensures that the correct smart contract is found, if it exists.

This registry smart contract relies on owners submitting their endorsed contracts. Thus, the design for registry should strive for two properties: every endorsed contract is added, and the amount of incorrect data is minimized. First, the entities creating contracts have a strong incentive to be found because it enhances the security by preventing downgrade attacks, and invites

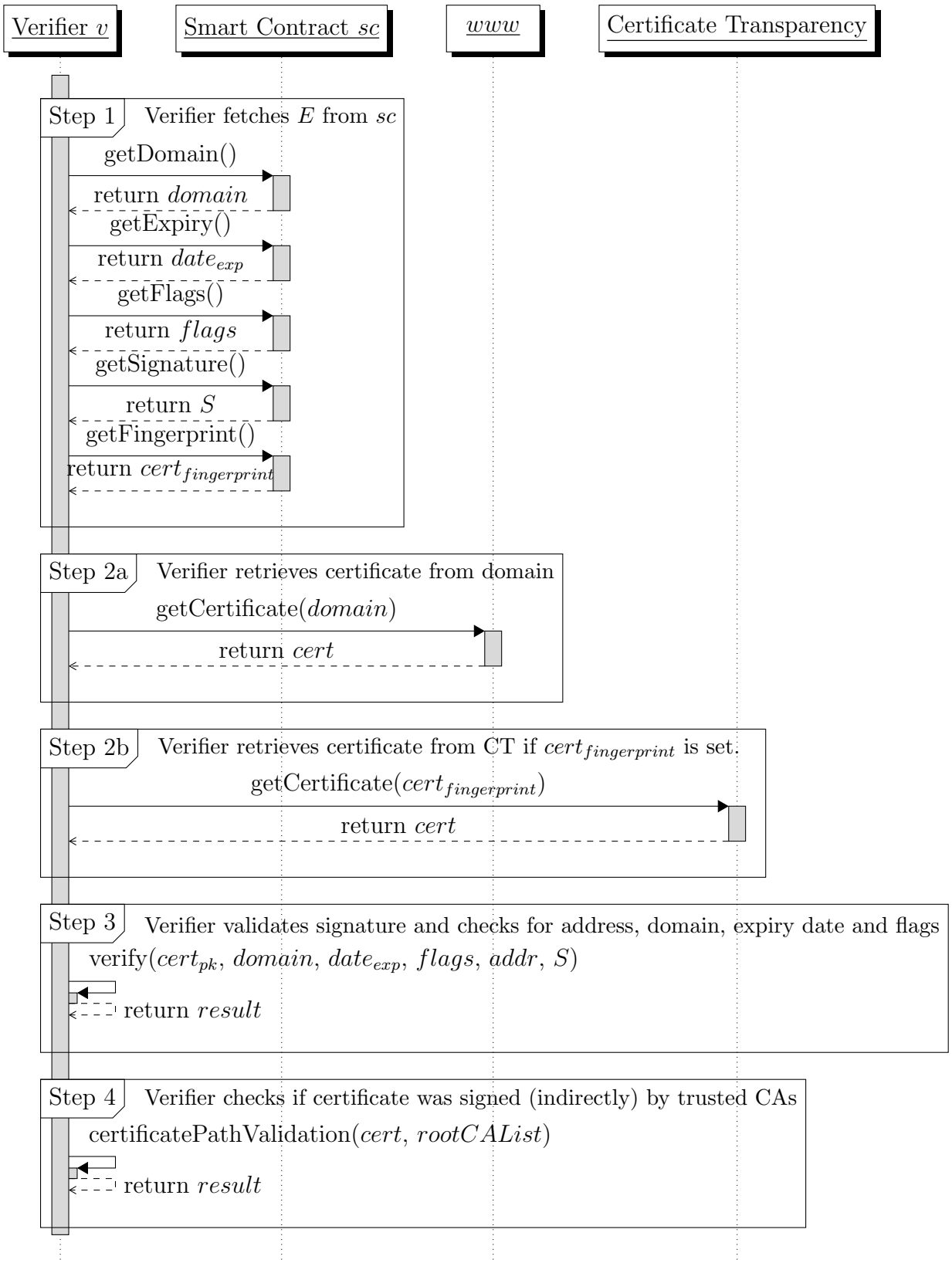



**Figure 4.2.:** Process of authenticating a smart contract, initially proposed in (Gallersdörfer and Matthes, 2020) and adapted.

> users and other parties to interact with their smart contracts. The registry smart contract enables owners to advertise their service and helps the users to find them. Second, malicious entities that do not own the respective TLS certificate should be discouraged from linking irrelevant contracts. Spamming the registry smart contract can cost a lot of money; further, spamming contracts will hardly impact the verification process as modern computation power is sufficient for processing hundreds of such contracts in milliseconds. Given this (dis)incentive structure, we have no restrictions in place for adding data to the registry."
>
> – "TeSC: TLS/SSL-Certificate Endorsed Smart Contracts" (Gallersdörfer and Matthes, 2021a).

### 4.2.4. Endorsement of Pre-Existing Smart Contracts

In principal, it is not possible to update existing smart contracts in a blockchain network. The immutability of the network and the code of the smart contract is a feature that prevents malicious entities from changing code after it has been deployed[14]. This restriction also applies to previously existing smart contracts, preventing them from adhering to our proposed interface.

To circumvent this limitation, we decide to enable endorsed smart contracts to subendorse other accounts, irrespective of whether they are smart contracts or externally owned accounts. These subendorsements are stored in the smart contract alongside the endorsement. The owner of the TLS certificate needs to consent to the subendorsement functionality by setting the **ALLOW_SUBENDORSEMENT**-flag (see Section 3.4 and Appendix B) in the endorsement. This feature can be seen as an extension of the original system without interfering with its inner workings; because it can be thought that the entity controlling the private key to the TLS certificate could deviate from the entity that controls the smart contract, we ensure the consent of the endorsing party by requiring a flag to be set.

The verification of directly endorsed smart contracts differs from the verification of subendorsements. The subendorsed smart contract does not contain any information about its endorsement. Therefore, the verifier must first find and verify smart contracts that are potentially endorsed. To obtain these parent smart contracts, the verifier retrieves all smart contracts that belong to one domain from the smart contract registry,

[14]While patterns exsit to enable updates for smart contracts, we do not cover this edge-case.

verifies them, and examine whether the subendorsed smart contract is stored in the `subendorsement` array.

A downside of subendorsements is that a discovery service is mandatory. Additionally, the smart contract cannot be authenticated if the user journey starts directly at the subendorsed smart contract and no domain information is available. In directly endorsed smart contracts, it is possible to obtain the domain information and authenticate the smart contract by retrieving certificate information from the domain or, if a certificate fingerprint is available, directly from Certificate Transparency.

### 4.2.5. Revocation

> "There are several reasons to revoke an endorsed smart contract. We describe two scenarios in which a revocation becomes necessary. First, the entity wanting to rescind the endorsement might do so voluntarily, e.g., as the smart contract is no longer in use. Second, the entity might need to involuntarily revoke the endorsement, e.g., as it has lost access to the smart contract or the respective private keys controlling the smart contract.
>
> In the first case, if the owner of the domain still has access to the smart contract or the respective account that administers the smart contract, the endorsement can just be removed by adding an invalid or empty endorsement. The off-chain verifier only accesses the latest information added to the smart contract, therefore, the verification of the endorsement will fail and users should be discouraged from interacting with the smart contract.
>
> In the second case, an entity does not have access anymore to the smart contract with the endorsement. To invalidate the contract, the respective TLS certificate has to be revoked. If the certificates are no longer valid, the validation will fail for the respective smart contract. As it is not possible to publish a contradicting statement with the same certificate in the smart contract, the certificate has to be revoked and a new one has to be used."
>
> – "TeSC: TLS/SSL-Certificate Endorsed Smart Contracts" (Gallersdörfer and Matthes, 2021a).

### 4.2.6. TLS Key Management

"Private keys used in TLS certificates have characteristics that need to be kept in mind when designing a system like [...] [*we propose*]. First, we need to address the expiry and rotation of key material. Second, we need to care for revoked TLS certificates. Last, we need to account for multiple existing TLS-certificates, thus multiple existing keys for one domain.

#### Key Expiry and Rotation

The key material in the X.509 certificates expires regularly. The *Certificate Authority/Browser Forum* has decided that TLS certificates are valid for a maximum time of 825 days[15]. Therefore, the public key and signature information must be regularly updated; otherwise, the verifying entity cannot assert the validity of the endorsed contract. To update the smart contract, a new endorsement has to be created with the respective certificate and inserted in the contract. This procedure ensures that smart contract endorsement remains valid.

This procedure bears the risk, that users get error messages for a brief time in which the smart contract endorsement does not match the certificate provided by the web server. This can be the case, when the endorsement is updated before the TLS certificate or vice versa. In case the smart contract experiences high demand, such an error message, even when only displayed for a short period of time, might not be acceptable to some parties. In this case, the usage of the certificate fingerprint variable is highly recommended. In that case, the verifying entity will look up the respective certificate in Certificate Transparency if it is not presented by the web server. With that, the certificate and endorsement can be updated independently from each other without short "downtimes" in the verification process.

#### Certificate Key Revocation

Key revocation becomes mandatory if the key material of a certificate is compromised. Such key material could be abused for creating additional

[15] `https://cabforum.org/2017/03/17/ballot-193-825-day-certificate-lifetimes/`, accessed 03rd May 2021.

endorsed smart contracts, potentially tricking users into believing that the new smart contract does indeed belong to the compromised entity. However, because our approach relies on the web server's public TLS key, revocation of the respective TLS certificate also renders potentially fraudulent smart contracts invalid. The software is able to evaluate whether the certificates not yet verified are included in CRL or if requests made through Online Certificate Status Protocol (OCSP) are valid.

#### Multiple Certificates for a Single Domain

Having multiple valid certificates assigned to one domain at the same time is common. Especially larger enterprises have many certificates, e.g., *facebook.com* has 521 valid certificates assigned to it at the time of this writing[16]. For this reason, we include the fingerprint $cert_{fingerprint}$ as a unique identifier of the respective certificate within the endorsement. This fingerprint enables us to retrieve the certificate that signed the endorsement via Certificate Transparency. Browsers such as Google Chrome require a certificate to be included in Certificate Transparency in order to be accepted. The retrieval from Certificate Transparency allows us to verify the endorsement independent of the web server's current certificate. This also allows the owner of the smart contract to update the server's certificate or the smart contract's endorsement with no concern of the smart contract suddenly becoming unverified."

– "TeSC: TLS/SSL-Certificate Endorsed Smart Contracts" (Gallersdörfer and Matthes, 2021a).

## 4.3. Threat Model and Security Implications

"In this section, we analyze the security and discuss risks [...]. Namely, we cover risks associated with TLS as a base protocol in subsection 4.3.1, examine cross-protocol attack vectors in subsection 4.3.2 and discuss other attacks on [...] [*our proposed system*] in Sections 4.3.3 and 4.3.4.

[16] `https://crt.sh/?Identity=*.facebook.com&exclude=expired&match==`, accessed 07th January 2021.

### 4.3.1. TLS as a Base Protocol

TLS certificates are often deemed unreliable and prone to fraud. Owing to the centralized nature of the system with multiple certificate authorities, potential points of failure exist: Certificate authorities might maliciously issue certificates for domains, revoke valid certificates without reason, or leak private keys, resulting in possible eavesdropping or man-in-the-middle attacks.

Nonetheless, this public key infrastructure (PKI) is a crucial part of the Internet. Without proper security mechanisms to reliably identify, connect, and communicate with a remote server, most current business and communication [*in the World Wide Web*] would not be possible. In particular, because these certificates have become omnipresent on most websites, other technologies (with potentially better security) lack the adoption for broad usage (see also subsection 2.1.3 and Section 6.1). The security model of websites assumes and relies on the proper functioning of this PKI. Thus, every system attempting to establish a secure connection between websites and smart contracts (which is the aim of this study) implicitly relies on this system. In [...] [*our work*], we make this trust assumption explicit, remove vulnerable intermediaries (such as the web application), and directly depend on this PKI by using private keys [...] [*from*] these certificates. It is also important to mention that the PKI is under active enhancement by systems such as Certificate Transparency (Laurie et al., 2013).

### 4.3.2. Cross-Protocol Attack Vectors

It is of high importance that one protocol does not endanger the functionality or security of the other. As [...] [*our proposed framework*] depends on TLS, there are two potential ways how cross-protocol attacks can occur: 1) An attacker abusing existing systems such as the web server or web application to attack [...] [*the system*] and 2) using the signature created for the specific endorsed contract to launch attacks on existing systems, e.g., the web server or users. The first attack scenario requires that a process or communication protocol unintentionally generates a valid endorsement (defined by the attacker) for a smart contract. The second scenario discusses

if the signature in the endorsement can be used to allow an attacker to impersonate the server.

In scenario 1) it is beneficial for an attacker that the TLS certificate is frequently used while communicating with the server. Potentially, every new request to the server leads to a response signed with the TLS certificate. To better understand the attack scenario, we have to look at the TLS handshake protocol, as outlined in RFC 8446 for TLS 1.3 (Rescorla, 2018). The `CertificateVerify` message *"is used to provide explicit proof that an endpoint possesses the private key corresponding to its certificate"* (Rescorla, 2018). It contains the used signature scheme and the signature that signs the hash of the data transferred previously to the handshake. From there on, two potential attack vectors exist: Either the hash function is not collision-resistant, or the same data is hashed. The first case can be discarded, as both TLS 1.3 and [...] [*our proposed system*] use cryptographic hash functions that are [*deemed*] collision-resistant. For the second case, the attacker would need to define the contents that are hashed. As the server generates 32 random bytes in the message `SERVER HELLO` with a secure random number generator, it can be assumed that it is not possible to generate a valid endorsement from the information of a TLS handshake.

In scenario 2), the identical mechanisms apply. As a difference, endorsements have a longer timespan and are not created as often as TLS handshakes. As the data signed in the endorsement creation process in no way resembles messages exchanged between client and server in a TLS handshake, they cannot be reused in the TLS handshake protocol. Therefore, the signature stored within the endorsed smart contract does not open up attack vectors on the TLS protocol or the communication with the server.

### 4.3.3. Downgrade Attacks

In a downgrade attack, an adversary tricks a party in a communication protocol to assume that the other party is incapable of adhering to newer (and more secure) versions of the communication protocol, which leads to the usage of an older and less secure version of the protocol. Because older versions are susceptible to further attacks or have no protection at all (e.g., plain text), an adversary can further exploit the communication. The same

applies to [...] [*our proposed system*]. Considering the base case and our introductory example of CoinDash, it becomes apparent that the user needs to know that a website uses [...] [*the proposed system*] to protect the users from sending transactions to malicious contracts. However, we cannot expect a user to know whether the counter-party actually implements [...] [*a TLS-endorsed smart contract*], and in the case of CoinDash, an address swap is still possible without the user noticing.

To account for downgrade attacks, the off-chain verifier (or other software implementing the verification mechanism) needs to know whether a contract exists for a given domain. If the verifier obtains a contract address from a website, it checks if this contract supports the [...] interface. If this is not the case, it checks if another valid smart contract exists for that domain by asking the [...] [*endorsed smart contract*]-registry. If such a contract is encountered without the current smart contract adhering to the interface standard, a warning is emitted to the user stating that the current contract has no additional protection against impersonation and that other contracts with such protection exist.

The [...] [*endorsed smart contract*]-registry is asked about previously existing smart contracts for a domain. If the original and endorsed smart contract is registered within this registry, downgrade attacks [...] [*can be*] prevented.

### 4.3.4. TLS Private Key Compromise

In the case of a downgrade attack (and similarly to the CoinDash incident), the attacker is assumed to have access to the contents of the webpage and can manipulate them, such as replacing an address. However, this might not always be the case. Attackers might have higher privileges on the victim's machine, allowing them to obtain the TLS certificate's private keys or the access to the file system allows them to create new certificates by using Automatic Certificate Management Environment (ACME) (Barnes et al., 2019) or Certificate Signing Requests (CSR) (Grajek et al., 2010) [...]. In such cases, the attackers are able to create new smart contracts with valid endorsements, allowing them to trick users into believing that they are interacting with a legit smart contract.

> [...] [*Our system*] is not able to prevent these attacks but allows *detecting* them. Similar to Certificate Transparency (Laurie et al., 2013), [...] [*we enable*] owners of TLS endorsed smart contracts to detect whether [...] new smart contracts are issued [*that link to their domain*]. As all contracts are publicly available on the blockchain, the network can be monitored for newly created smart contracts that contain an endorsement for a specific domain. With this information, companies are able to implement monitoring services and act accordingly by reclaiming possessions of their servers and revoking any compromised key material as outlined in 4.2.5."
>
> – "TeSC: TLS/SSL-Certificate Endorsed Smart Contracts" (Gallersdörfer and Matthes, 2021a).

## 4.4. Augmentation of User Wallets

In this chapter, we outlined the specifications of a system that supports the use of TLS certificates to endorse smart contracts in a blockchain-specific environment. Given the specifications, the outlined processes, and involved third-party entities, such as web servers, it is straight forward to develop an application that accepts a *domain name* and a *smart contract address* to verify that a) the domain and smart contract exist, b) the smart contract stores an endorsement, c) the respective endorsement links to the given domain name, d) the endorsement is signed by the public key of a valid certificate belonging to the domain, and e) the endorsement is not expired.

While this software can easily be made available to users or be provided via an API, the integration into a user interface and the regular flow of the user journey while executing a transaction in a blockchain environment needs to be carefully designed.

Previous work was done in (Ebel, 2021), *"Augmenting the MetaMask-Wallet with Domain Name based Authentication of Ethereum Accounts"*, and (Gallersdörfer et al., 2021a), *"Augmenting MetaMask to support TLS-endorsed Smart Contracts"*, as well as accompanying supplementary material in GitHub (Gallersdörfer et al., 2021b) that designed, developed, and tested an augmentation of the browser-based wallet MetaMask that verifies the authenticity of endorsed smart contracts.

In this section, we first provide an overview of the approach from (Ebel, 2021; Gallersdörfer et al., 2021a) and critically reflect on its results in subsection 4.4.1. Then, we extend the results of (Ebel, 2021; Gallersdörfer et al., 2021a) in sections 4.4.2 and 4.4.3, as follows:

- We verify and extend the insights into the design of the browser-based warnings; the initial analysis was conducted in 2020 and covered only the Google Chrome, Mozilla Firefox, and Microsoft Edge browsers. We investigate the differences from 2020 to the time of writing and extend the browser list to include Apple's Safari. Further, given the availability of mobile wallets for Apple iOS and Google Android, we also extend the analysis to the respective mobile browsers.
- We verify that limitations preventing a fully decentralized application within Google Chrome and Microsoft Edge still exist; we further analyze whether this limitation applies to any browsers, as introduced earlier.
- We discuss required changes to the underlying system to allow for further decentralization, overcoming the limitations of Chromium-based (and potentially other) browsers.

### 4.4.1. Previous Work and Results

To briefly reiterate: Users interact with blockchain networks sign transactions using their private key, which is managed within wallet software. They obtain the information about with whom to interact via websites that either include address-specific information or directly contain decentralized applications. These decentralized applications use *Web3.js* to provide the user with an easy-to-use interface in case the complexity of a transaction is higher than just sending funds to another address. Alternatively, users obtain information about addresses from other communication channels, such as text or email.

Ideally, the software verifying the endorsement does not remain standalone software but rather integrates as closely as possible to the user signing the transaction, meaning within the user's wallet. This work was conducted in (Ebel, 2021; Gallersdörfer et al., 2021a).

The following steps were conducted in that research:

1. **Analysis of browser-based warnings:** Browsers, as the oldest interface for regular users to access the WWW and integrate basic security layers, such as SSL, also face the issues of displaying warnings to the users about invalid certificates.
2. **Analysis of error-cases:** In the second step, one needs to understand which error scenarios are possible when verifying the validity of endorsements stored in smart contracts.

3. **Model-creation and implementation:** Thirdly, the model for warnings related to the proposed system needs to be conceptualized, designed, and implemented.

4. **Verification of approach:** A user study is conducted in order to verify the effectiveness of the newly augmented wallet.

In the first step, (Ebel, 2021; Gallersdörfer et al., 2021a) identified three main states of the security of a website that a browser communicates to the user: a *positive indication*, specifying that the communication with the server is authenticated and private; a *negative indication*, indicating that the communication to the server is depleted[17]; and a *downgrade indication*, indicating that no security measurements are in place at all. Upon further investigation, the browsers' warnings did not differ much; Chrome and Edge had a single-page warning design, differentiating only between underlying error messages and whether users could continue to the website. Firefox had two error pages, one coined as *overridable error page*. The second page was displayed only in cases of critical errors preventing the user from proceeding to the potentially unsafe website. Indications of protocol downgrade also differed only in small details.

In the second step, common errors within X.509, in combination with TLS and the proposed system, were analyzed (Ebel, 2021; Gallersdörfer et al., 2021a). In addition to the reasons to deem an X.509 certificate invalid, as outlined in RFC 5280 (Boeyen et al., 2008), endorsements can be invalid if the signature is wrong, the endorsement is expired, the domain is incorrect, or the root CA is not trusted. As endorsed smart contracts are not adopted within any blockchain system, the absence of an endorsement does not directly result in a dangerous state. Therefore, a distinction was made between domains that had already deployed endorsed smart contracts and ones that had not. Domains that had deployed smart contracts once were susceptible to downgrade attacks, whereas if no additional security measurement were ever in place, formally speaking, a downgrade attack could not occur.

A model was conceptualized and implemented in the third step in (Ebel, 2021; Gallersdörfer et al., 2021a). MetaMask has a two-step approach to sign and send a transaction from an address: Initially, the dialog is opened as soon as MetaMask recognizes that the user wants to initiate a transaction. This can happen for two reasons: Either the user manually opens the plugin and clicks transfer, or the website in combination with *Web3.js* indicates to MetaMask that a transaction is required. In the first stage,

[17] Security measures that are in place are configured poorly, are expired, or do not apply to the respective domain.

the plugin displays key information such as recipient and monetary and data values. The user can then "proceed" to the next screen, which summarizes the information about the transaction, further including information about transaction fees and expected execution time. The user can confirm and sign the transaction or either revert to the previous screen or edit the transaction fee settings. The augmented version of MetaMask in (Ebel, 2021; Gallersdörfer et al., 2021a) interrupts the user in a potentially dangerous state on the second screen and explains the reasoning for interruption; error messages and design decisions such as button colors are in accordance with the Google Chrome browser, such as highlighting the "Cancel" button.

During the implementation, an interesting obstacle was discovered. In contrast to Firefox, Chrome and Edge do not allow access to the X.509 certificate from the website. As the access to the certificate (to verify the endorsement) is crucial, a purely browser-based implementation is not possible in Chrome or Edge.

In the fourth step in (Ebel, 2021; Gallersdörfer et al., 2021a), a user study was conducted that simulated an address replacement attack, a situation similar to the CoinDash ICO (Initial Coin Offering) hack. The users were introduced to an investment opportunity via an email that linked to a website. In the first round, they were exposed to the regular version of the website and were expected to send money to the given address. In a second email, the users were prompted again to invest on that website, but the address had been replaced by a malicious one, resulting in the previously designed warning page, alerting the user of potential problems. It was found that displaying such a warning page could help to reduce the damage resulting from such attacks. However, how successful the system is against attacks in which the user is instructed to ignore warnings remains an open question. This scenario is not unlikely, as the attacker controls the information on the website. Google's Chromium team redesigned their browser warnings in 2015 to increase users' attention to these messages, but the results fell short of expectations (Felt et al., 2015).

### 4.4.2. Revisiting Browser-based Warnings

Browser-based warnings improved greatly. SSL 3.0 was introduced in 1996 as a way to protect the communication between client and server (Freier et al., 2011)[18]. Since then, browsers have needed to communicate the state of the connection to users. Notifying and warning users about the security of the underlying connection is a challenging task

[18]Given that no formal publication of SSL 3.0 initially existed, this citation refers to a formal publication of the historical record of the SSL 3.0 protocol published in 2011.

of multiple dimensions:

- **Frequency of Warnings**: If users receive warnings too often, they might ignore them more easily. However, if warnings appear only on rare occasions, users might not be warned in all cases of a threat.
- **Level of User Knowledge**: Warnings need to be properly worded to suit both informed and uninformed users, enabling both to make conscious decisions.
- **Visualization and Guidance:** Warnings and errors must be properly visualized. Users need to be guided in their decisions, and might need protection from potentially unsafe actions.

Despite the importance of safe web browsing, warnings in browsers do not receive much attention in research. For example, there have been proposals to standardize the collection of browser warnings (Kraus et al., 2020) and studies on the perception of IT-professionals (Ukrop et al., 2019), TLS interception in enterprise applications (Waked et al., 2020) or the redesign of browser warnings (Felt et al., 2015).

Several challenges hinder research in this area: First, the findings of research in this field are rather short-lived. We found several outdated models in previous research, as the perception of the importance of security indicators has changed. For example, extended validation (EV) certificate information is no longer displayed to the users. Second, browser-warning pages seem to change frequently, and it is challenging for researchers to obtain current and past forms of browser warnings. Third, data and methods of obtaining user behavior data are available only to the large enterprises developing the browsers. For example, researchers working on Chromium have various ways to conduct user studies with users who opt in to collecting telemetry data (Felt et al., 2016).

#### Warnings in Major Browsers

To verify the results of the analysis of browser warnings and subsequent design decisions for a MetaMask augmentation in (Ebel, 2021; Gallersdörfer et al., 2021a), we a) collect screenshots of warnings in the major browsers, b) synthesize these screenshots into a tabular overview, and c) share obtained insights. Thereby, we consider three states in the browser, as proposed in (Ebel, 2021; Gallersdörfer et al., 2021a) and (Felt et al., 2016):

- **Positive Indication:** Indication that a security protocol is active. We collect positive indications for two protocols: TLS with regular certificates and TLS with EV certificates.

- **Negative Indication:** Indication that a security protocol is active but corrupt for any reason. We collect negative indications for minor errors (such as an expired certificate) and major errors (such as having a misconfigured certificate with HTTP Strict Transport Security enabled).
- **Indication of Protocol Downgrade:** Indication that a security layer is absent. We collect indicators active for regular `http`-connections.

We analyze the following browsers (brackets indicate version number): Google Chrome (110), Microsoft Edge (109), Mozilla Firefox (109), Opera (95), Safari (16), Firefox for Android (109) and Safari for iOS (iOS 16.1.2). We analyze 1) the URL-pattern, 2) any symbols next to the URL, 3) the information page about the certificate when clicking on the symbol, 4) the warning page (when present) and 5) any information that shows up when clicking on "further information" or "advanced" on the warnings page. We use `https://badssl.com` to display warnings in the browser[19] and `https://globalsign.com` as an example with an EV certificate (King et al., 2022).

Table 4.1 gives an overview of all browser bars in their various states. Note that for iOS, the color of the website influences the color of the bar, resulting in a "red" HTTP state (which comes from a red webpage) and a "green" EV state.

In Appendix C, we give a detailed overview of the browsers' certificate information pages, warning pages, and details about the "advanced"-area of the individual warning pages.

### Insights

In contrast to the results in (Ebel, 2021) and (Gallersdörfer et al., 2021a), the **single-page warning design became the dominant form for warning pages**. Previously, Mozilla's Firefox used two warning pages to display minor and critical errors and it now uses a single-page warning design, in accordance with browsers such as Chrome, Edge and Opera. Interestingly, Safari and Safari for iOS have a very simplistic warning page for critical errors: It strongly resembles the non-security-related "server cannot be found" error page and leaves no option for the user to further interact with the website, such as "going back to the previous page" or "reload the page".

Therefore, the basis for the design and development decisions for the MetaMask plugin, such as selecting a single-page warning design, still hold up.

[19] We use the "expired"-error for the minor-error state and the "certificate-pinning"-error for the major error state.

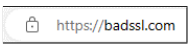
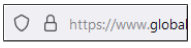
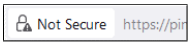
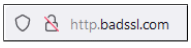
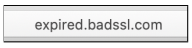
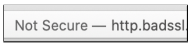
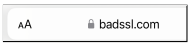
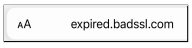
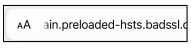
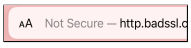
| | Secure | Secure (EV) | Minor Error | Major Error | HTTP |
|---|---|---|---|---|---|
| Chrome |  |  |  |  |  |
| Edge |  |  |  |  |  |
| Firefox |  |  |  |  |  |
| Opera |  |  |  |  |  |
| Safari |  |  |  |  |  |
| Android |  |  |  |  |  |
| iOS |  |  |  |  |  |

**Table 4.1.:** URL bars for all major browsers in four different security states, namely secured connection (with regular certificate), secured connection (with extended validation certificate), minor error in the TLS protocol, major error in the TLS protocol, and protocol downgrade (HTTP only).

In addition to the implications on the research in (Ebel, 2021; Gallersdörfer et al., 2021a), we make following noteworthy observations:

- The **differences in warning pages of browsers are marginal**. For obvious reasons, mobile browsers are limited in their display size and thus cannot use the same visuals and symbols used in desktop browsers but still communicate concisely to the user. Some browsers, such as Firefox, Chrome and Edge, use a warning symbol on their error pages, whereas other browsers, such as Opera, use their logo. We can find changes in little details in icons, but previous research shows that the effects of such changes on security are minimal (Thompson et al., 2019).

- **EV certificates have become meaningless**. No browser differentiates between regular TLS and EV certificates in its UI. Only in the active inspection pop-up for a certificate (after clicking the "lock"-symbol), browsers might show the entity to which the certificate is issued, potentially even the issuer. Further, this additional information is not highlighted in any form, but just appears as a supplement to the regular security notice (e.g., in Firefox, "You are securely connected to this site"). This also aligns with research that proposes its removal (Thompson et al., 2019).

- **The user's privacy has gained in importance**. Instead of enhancing security indicators for the connection between the browser and web server, increasing focus has been put on protecting users' privacy. Firefox displays a protection icon prominent beside the security lock, informing the user after a click about trackers

on the respective site and overall blocked trackers. Opera allows blocking trackers and ads directly within the URL-bar. In 2022, Apple introduced features that allow blocking trackers in native iOS apps[20] and support blocking trackers in their browser for mobile and desktop.

### 4.4.3. Limitations and Mitigation Strategies

While the design and user study in (Ebel, 2021; Gallersdörfer et al., 2021a) suggest that user security could be enhanced by our proposed system, an implementation problem was discovered. In most browsers, plugins are unable to access certificate information of the currently visited website. Only in Mozilla Firefox is access to the respective API fully implemented and allows access to the certificate. We investigate whether this issue still exists in the browsers' current versions and potential pathways to overcome this problem.

#### Situation in Browsers

Mozilla's web developer documentation gives an extensive overview of the compatibility of all major browsers (Chrome, Edge, Firefox, Opera, Safari, Firefox for Android, and Safari on iOS) with functions from the *webRequest*-API (Mozilla Developer Network, 2022). Table 4.2 displays the support of each browser with the desired `getSecurityInfo()`-function.

| | Desktop | | | | | Mobile | |
|---|---|---|---|---|---|---|---|
| Feature | Chrome | Edge | Firefox | Opera | Safari | Firefox | Safari |
| `getSecurityInfo()` | ✗ | ✗ | ✓(62) | ✗ | ✗ | ✓(62) | ✗ |

**Table 4.2.:** Browser support for the `getSecurityInfo()` function (Mozilla Developer Network, 2022). Brackets indicate the version in which the feature was introduced.

The situation in Mozilla Firefox is unchanged. The browser still allows access to certificate information via the `getSecurityInfo()` functionality, resulting in an identical situation to when the initial research was conducted (Ebel, 2021; Gallersdörfer et al., 2021a).

In Chromium-based browsers such as Chrome, Edge, and Opera, there is still no support for the desired functionality. However, it remains a rather actively discussed subject in the respective bug tracker since 2016, in which users and developers urge

[20] See `https://cnb.cx/3gmOmgl`, accessed on 13th February 2023.

Google to implement the respective functionality[21]. Interestingly, it is claimed that the functionality was not implemented on purpose; it seems that enabling this function leads to numerous extensions that either overpromise on their functionality or leak web browsing activity to third parties, which seems to be the case for the Firefox plugin repository[22].

The *Webkit*-based Safari did not support the entirety of the *webRequest*-API until version 14, released in September 2020, in which some features were introduced (Mozilla Developer Network, 2022). Still, the relevant functionality is missing, and there are no recent discussions or interest in extending the support for this browser.

In mobile browsers, the situation is almost identical. The Firefox browser for Android supports the function call in question, and the Safari browser on iOS lacks the entire *webRequest*-API. For Chrome on Android, the same results as for the desktop version apply.

This leaves a dire picture of the ability to access TLS-certificates from plugins within the browser directly. Only one browser supports the functionality (which makes up only 3% of world-wide browser usage[23]), and any other discussions for its support in other browsers are either stalled or do not exist. From this perspective, widespread direct access to the TLS certificate within a browser plugin seems to be unrealistic and an unfavorable design decision. The question remains of whether there are alternative avenues of system design to mitigate this issue.

### Alternative System Designs

Given the infeasibility of using the system purely within a browser plugin in browsers other than Firefox, we need to reconsider the underlying technological foundation of authenticated smart contracts and how decentralized verification can occur regardless. We outline four potential pathways for the underlying architecture to change.

**Local Installation Outside the Browser** The proposed augmented plugin is not the first that suffers from being unable to access the TLS certificates of the visited websites. *CheckMyHTTPS* helps users to verify whether their secure connection to a website has been intercepted by TLS proxies or other means, such as Firewalls or security

[21] See `https://bugs.chromium.org/p/chromium/issues/detail?id=628819`, accessed on 10th February 2023.

[22] See `https://bugs.chromium.org/p/chromium/issues/detail?id=628819#c33`, accessed on 10th February 2023.

[23] See `https://gs.statcounter.com/browser-market-share`, accessed on 10th February 2023.

suites (Rey et al., 2023). The browser plugin also accesses TLS certificate information, which works adequately within Firefox. Given that they cannot access TLS certificates within Chrome, the developers decided to rely on a locally deployed *Python* script[24] that communicates with the browser via `NativeMessagingHosts`, an API that allows the browser to communicate to locally deployed applications.

While this could work in theory, it poses a considerable challenge to the user and an obstacle to adoption. A user might lack the skillset or the trust to install a full Python suite alongside the script on their machine. This might result in a similar user experience to having a native application that integrates the functionality but would hinder adoption significantly. This is further highlighted in one of the two reviews in the Chrome Web Store in which a user complains about being required to install Python[25].

We deem requiring the installation of a second application alongside the browser plugin not favorable.

**Centralized Service** An alternative approach is the usage, as initially suggested, of a centralized service. This would allow for easy integration, as a centralized API server would be provided. Each time the user wants to initiate a transaction, the receiver and currently visited website are sent to the server. This, in return, obtains the certificate, verifies whether the respective protection is in place or a downgrade pattern is detected, and sends back the result to the plugin. The plugin then displays the result to the user. It can be decided whether the verification should occur within the service or the browser plugin; nonetheless, the domain name must be shared for the service to work. In addition to the privacy concerns, this centralized service can be at aim for cyber attacks or DoS attacks.

Given the centralization, privacy, and security concerns, a centralized service is out of the question to solve the underlying issue.

**Certificate Transparency** CT, as an append-only log, stores all newly issued certificates; therefore, it should also include any certificate used within our proposed system. A browser plugin could access CT, retrieve the respective certificate, and verify whether the certificate's private key correctly signs the endorsement. While this seems to be in theory a feasible approach, in practice, it is not. CT is an append-only log; SSLMate's

[24] See `https://github.com/checkmyhttps/checkmyhttps/blob/master/Chromium/native-app/checkmyhttps.py`, accessed 13th February 2023.

[25] See `https://chrome.google.com/webstore/detail/checkmyhttps/jbnodnfpdcegpnflleanllmiihkinkio`, accessed 13th February 2023.

*Cert Spotter Stats* highlights that over 8 billion certificates are managed within CT while using about 28 TB of data storage (SSLMate, 2023). This data size is not manageable within a browser plugin and puts significant demands on any infrastructure provider. Services such as `crt.sh` allow for a simple search within the logs, resulting in the same issues as with any centralized service that retrieves the certificate from the server on behalf of the user.

Given the data size and complexity of CT, its usage is not feasible within our proposed system.

**On-chain Authentication** Shifting the burden of obtaining certificate information to a local script, external providers, or CT have their considerable downsides. One potential avenue, which has not been explored yet, is the shift of certificate storage and verification on-chain. In this case, owners of domains and issuers of endorsements are required to put their certificates alongside their signatures in a controlled on-chain environment, in which proper verification is ensured. Wallets interacting with these smart contracts, could verify the integrity of respective endorsements using these smart contracts and displaying the resulting information to their users.

As this comes with significant changes to the system and smart contract design, we discuss the purely on-chain enabled authentication in Chapter 5.

## 4.5. Summary

In this chapter, we provided insight into our system, which supports users and enterprises alike to mitigate the risks of loose coupling between websites and blockchain addresses by enabling a decentralized verification mechanism for TLS certificate signatures.

Loose coupling of information between the WWW and blockchain networks is a significant risk to enterprises, start-ups, and users in the blockchain ecosystem alike. Numerous hacks and incidents have led to the loss of millions of USD.

The architectural approach of storing endorsements on-chain in individual smart contracts and tracking issuances in a logically centralized smart contract allows the end user to verify that a smart contract address indeed belongs to a website, as its owner has previously issued such an endorsement.

Thereby, we showed that the interplay between TLS and endorsements is minimal and that the issuance of endorsements does not increase the attack surface. Further, we outlined how, given the adversary model, attacks that replicate the endorsement can

still take place. However, due to the transparency of the blockchain network, malicious issuances can be detected and user risks mitigated.

Lastly, we reiterated the integration of the endorsement verification mechanism in the browser wallet MetaMask conducted in (Ebel, 2021) and (Gallersdörfer et al., 2021a). We revisited browser-based warnings and verified that assumptions about the browsers and their features still hold. Given the limitations of modern browsers, we discussed alternative technological avenues for enabling the verification of endorsements in browsers. We conclude that an off-chain approach is not feasible at this time.

In the next chapter, we describe an additional architectural approach to enable the verification of endorsements in an on-chain context, enabling decisions within smart contracts based on the validity of the endorsement. Thereby, we cover both on-chain authentication utilizing TLS certificates and on-chain authentication utilizing DNSSEC, as seen in ENS. Further, we develop a dataset of all domains that have been bridged to ENS using DNSSEC.

# Chapter 5.

# On-Chain Verification

This chapter discusses different architectural designs to enable the use of certificates and their signatures in a blockchain-based environment. Different problem statements and use cases require alternative certificate storage, management, and verification approaches. The approach enables a consensus about the validity of single certificates and their signatures, in turn enabling their usage in a deterministic environment, e.g., smart contracts. We further analyze ENS, as it follows a similar approach utilizing DNSSEC. This chapter is split into the following sections:

- Section 5.1, *"Problem Statement"*, discusses the pain points and potential use cases that require a separate approach for the on-chain usage of TLS-certificates and their signatures.
- In Section 5.2´, *"System Architecture and Processes for On-Chain TLS-Certificate Usage"*, we describe the system architecture design comprising a certificate database and an endorsement database and mechanisms to validate certificates and their signatures entirely on-chain as proposed in (Groschupp, 2020) and (Gallersdörfer et al., 2021c).
- In Section 5.3, *"DNSSEC Integration in the Ethereum Name Service"*, we analyze the architecture and integration of DNSSEC within the ENS protocol.
- We discuss architectural design decisions for both ENS' DNSSEC approach as well as the proposal described in (Groschupp, 2020) and (Gallersdörfer et al., 2021c), as well as alternatives and their rationales in Section 5.4.
- Lastly, in Section 5.5, *"ENS DNSSEC Domain Dataset"*, we develop a dataset of the ENS DNSSEC ecosystem. We describe how we gather, filter and apply data in

order to derive valuable conclusions for Chapter 6, *"Evaluation and Comparison of Approaches"*.

Section 5.2 is based on "Exploring the Use of SSL/TLS Certificates for Identity Assertion and Verification in Ethereum" (Groschupp, 2020) and "Mirroring Public Key Infrastructures to Blockchains for On-chain Authentication" (Gallersdörfer et al., 2021c).

## 5.1. Problem Statement

In this section, we discuss two additional problem statements that exist at the intersection of WWW and blockchain networks. Both issues have briefly been raised in subsection 3.2.1. We discuss the problems of *Non-Human-Readable Names* and *Lack of Access Control* and why these problem statements, besides the drawbacks of the off-chain approach, lead to a deviating architectural approach.

### 5.1.1. Non-Human-Readable Names

Users that intend to interact with other parties on blockchain networks often obtain the information of counterparties by means of the WWW. In case information is accessed via this approach, the binding between a website and blockchain address can simply be verified; the required information (the address) arrives via the website and the wallet (as outlined in Section 4.4) checks whether the domain of the website belongs to the respective endorsement stored within the address. Verification takes place within the wallet, and no further information is required from external sources.

Suppose the user receives information via other means than the WWW. In that case, verifying the endorsement within the address suddenly raises a problem: Which domain does the smart contract belong to, and is it the domain the user intends to interact with? Given that no browser context is available, a wallet cannot verify the relationship between a website and a smart contract.

To summarize: The problem exists in case a user obtains address information without the given context of the browser.

The need for a diverging approach stretches to further situations in which address information is expected to be used on-chain. On-chain smart contracts cannot, as previously outlined, access data from external sources without additional help from oracles. Surely, the validity of respective endorsements and domains could be provided by an oracle network incentivized to do so; however, this would introduce third parties that would be in control of this process, resulting in potential harm ranging from a simple time delay to potentially wrong statements about the validity of specific endorsements, given high enough stakes.

An oracle would impose further design questions: Who are the operators of this oracle network? How are they incentivized? How are disputes between single entities of the oracle network resolved? What are the expected time delays, and what happens if an answer from the oracle is missing? Are funds locked indefinitely? Who defines the trusted

root CAs?

When intending to leverage these domains in an on-chain environment, a sufficiently decentralized and open system must be put in place. Ideally, the verification and authentication could occur within a single transaction, removing the need for complex callback structures, as expected in an oracle-enabled system.

### 5.1.2. Lack of Access Control

Access control is an essential feature within blockchain networks. In public permissionless networks, some form of access control exists (e.g., in token-gated communities)[1]. Further, access control becomes more relevant in public or private permissioned blockchain networks. Often, these networks are set up so that only specific entities can participate in the consensus or even join the network.

Access control or similar means always require an exchange of information before allowing an entity to pass some form of access control successfully. These forms of information exchange often require secondary channels, such as emails or chat systems. Anecdotal evidence suggests that no additional entity verification is required beyond the brief exchange of contact information. Further, manual labor is required to update access information, which can also be prone to errors. This leaves the question of how well-protected these access control mechanisms are for specific communities.

To alleviate the problem of these simple access control mechanisms, using domain names (and, thus, endorsed smart contracts) could enhance security properties and increase the share of automation in the system, as novel "request for access" methods could be provided. For example, a two-step method could be in place in which entities could apply to a community or system using their associated domain name and, the owner or a respective panel could decide whether to accept or reject the applicants, while also ensuring that the applicants are the entities that they claim to be.

However, similar to the previously outlined problem, if someone were to use domain names for access control to their systems, the endorsement verification would have to take place in an on-chain environment, requiring a system capable of handling this information purely on-chain.

[1] Users are required to obtain a token or a NFT before joining the community.

## 5.2. System Architecture and Processes for On-Chain TLS-Certificate Usage

In (Groschupp, 2020) and (Gallersdörfer et al., 2021a), a new architecture for storing, verifying, and retrieving the verification results of TLS certificates alongside any potential endorsements in a purely on-chain setup was proposed. In this section, we reiterate the components and give an overview of their functionalities. We discuss any rationale that led to this specific architecture and alternatives in 5.4. First, in subsection 5.2.1, we describe the first of two databases: the database for storing X.509 certificates. Then, we present the smart contract endorsement database in subsection 5.2.2. Lastly, we give an overview of the relevant processes and considerations in subsection 5.2.3.

### 5.2.1. X.509 Certificate Storage Database

First, we discuss the functionality of the database. Then, we present the rational for the design decisions.

To begin, one needs to store any X.509 certificates. The validity of the certificates is the main objective of such a database. While it seems apparent, certificates need to be syntactically and semantically correct. Certificates must follow the X.509 certificate standard, including adherence to any variable types. Signature information, issuer names, and respective hashes must be consistent to name only a few additional properties. Further, certificates must either be self-signed (potentially acting as a root certificate) or signed by a CA certificate.

We discuss the X.509 certificate storage databases' functionality by highlighting respective *CRUD*-operations:

- **Create**: The X.509 certificate is added to the certificate storage database by verifying its semantic and syntactic integrity. If the certificate is valid, it is added to the database alongside any identifying information and its public key.
- **Read**: Any certificate stored within the database can be retrieved by identifier, which is the SHA-256 hash in `DER`-format.
- **Update**: The contents of a certificate cannot be updated, as new parameters or variables (such as validity) change the certificate and its fingerprint and require a new signature from the respective certificate authorities. However, it is possible to update the state of the certificate: If CRL or OSCP data is pushed to the smart

contract, the certificate's validity can be confirmed or, if it is included in a CRL, revoked.

- **Delete**: It is impossible to delete a certificate; a revocation can change the certificate's status to *revoked*, rendering the certificate useless.

(Groschupp, 2020) and (Gallersdörfer et al., 2021c) propose to allow the storage of any certificate that adheres to the X.509 protocol. The system is not limited to a specific list of certificate authorities to keep the system open and allow for any PKI. In contrast, any certificate can be added, and anyone can create a root certificate to sign respective sub-certificates. To prevent malicious entities from signing arbitrary data and endorsing domain names, users must specify which root certificates they want to trust when verifying endorsements. Harm is thus prevented, as users will not trust arbitrary certificates but rather a predefined list. Nonetheless, selecting this list of valid issuers can be a problem, as we outline in subsection 6.2.2.

### 5.2.2. Smart Contract Endorsement Database

Similar to the certificate database, (Groschupp, 2020) and (Gallersdörfer et al., 2021c) propose an endorsement database. The specification for endorsements partly follows [2] the description outlined in Section 3.4. Nonetheless, the verification of an endorsement follows the same specifics initially proposed in Section 4.2. The only difference is that endorsements and certificates must be verified only once; the verification result for the endorsement and its respective certificate are stored in the respective databases. If one of the certificates is invalid, the respective endorsement is also invalid.

In addition to tracking, the database also keeps track of **root stores**. Root stores are a list of trusted root certificates, which we informally describe in Equation 5.1:

$$RootStore = \{CertificateList, Owner\} \tag{5.1}$$

Root stores are a way of conveniently storing information about trusted root certificates. These lists can be created, and new root certificates can be appended or removed. Users that verify an endorsement can select which root store to trust. As an owner is defined, these ownerships can be decentralized so that they can be destroyed (e.g., by transferring

[2] (Groschupp, 2020) and (Gallersdörfer et al., 2021c) use other terminology for parts of the endorsements and ignore flags set in the endorsements.

the ownership to `0x0`), or some form of governance can be set up to decide which certificates to add and remove.

For endorsements, the functionality is analogous to the certificate database:

- **Create**: Endorsements are created off-chain using respective key material and signature information and are then added to the endorsement database. Endorsements link only implicitly to a root certificate by relying on an intermediate certificate and do not need to link to root stores.
- **Read**: Endorsements can be read by their index in the database, associated domain, or account addresses. The index of a root store needs to be provided to ensure that the endorsement is valid for the user. The smart contract returns the respective information.
- **Update**: Similarly to the certificate database, an update is not possible; instead, a new endorsement needs to be added to the endorsement database.
- **Delete**: To revoke an endorsement, a specific signature can be pushed to the smart contract, which updates the state of the endorsement and sets it to invalid.

The specific smart contracts are described in (Groschupp, 2020) and (Gallersdörfer et al., 2021d). While these programs provide a proof of concept, they suffer from exhaustive gas consumption and potentially unreachable states (e.g., if the list over which to iterate with a `for`-loop grows too long). In Section 6.3, we provide preliminary cost structures.

### 5.2.3. Processes and Rationale

In the following, we give insights into the processes of the system. Further, we discuss the rationale for logically centralizing the smart contract system.

#### Processes

Several processes are involved in the systems proposed in (Groschupp, 2020) and (Gallersdörfer et al., 2021c). In the following, we discuss the processes for a) issuance, b) verification, and c) revocation of endorsements.

**Issuance** The issuance of a new endorsement to the system involves several steps.

- **Creating the Endorsement**: In the initial step, the endorsement is created. Similarly to the off-chain approach outlined in Chapter 4, the endorsement follows a specific structure and contains information about the domain, the address, the certificate, the expiry date, and more. The data is then signed with the private key of the respective TLS certificate and stored for later submission to the blockchain network.

- **Submitting Certificates**: Issuers must verify which certificates are already present in the blockchain context and to what extent they cover the certificate chains of the TLS certificates they used to generate their endorsement. It can be expected that at least one certificate has not yet been submitted to the blockchain and needs to be stored in the certificate storage database. The issuer needs to submit each missing certificate via a single transaction to the contract. With the given implementation, a batch submission of multiple certificates is not supported. The smart contract verifies that each certificate is signed by a previously introduced certificate. Root certificates can be submitted at will but need to be trusted later by users.

- **Submitting Endorsement**: After all relevant certificates have been stored in the certificate storage database, the endorsement can be submitted to the smart contract. Again, the issuer needs to create a separate transaction that contains the endorsement; a batch submission is not implemented. The smart contract verifies the integrity of the endorsement and checks if its contents are correct, and the provided signature belongs to a prior added and verified TLS certificate. After the submission, the endorsement is sufficiently verified and can be used. Additionally, a binding between the endorsement and the respective root certificate is created to make verifying the endorsement cheaper.

**Verification** The verification of an endorsement took already place in the previous step. The verification of the endorsement needs to take place only once; as a result, the verification can be stored and subsequently accessed by anyone interested in the endorsement. However, user preferences in the form of different trusted root certificates can differ, requiring an additional step to verify whether to trust a given endorsement or not.

- **Creating a Root Store:** Initially, root stores containing a list of trusted root certificates need to be created. One transaction is required to create a root store,

and a subsequent transaction is required to add root certificates to the respective list. Potentially, one root store could be predefined and accepted by the majority of the community, as otherwise, everyone would need to manage their own root store.

- **Verifying an Endorsement Against a Root Store:** The root store is the main denominator for users to understand whether an endorsement is valid. A valid endorsement can be created at any time for any domain as it is possible to add its own root certificates, so the trusted list of root certificates is crucial to correct verification. The user submits a transaction to the system to ask whether an endorsement for a given account, domain, and root store is valid; given that it is only a read operation, gas costs are not required. This information can subsequently be used to facilitate transactions.

**Revocation** There are two processes for revoking an endorsement: Either the endorsement itself is revoked or the underlying certificate used for creating the endorsement is revoked. Further, the certificates can be revoked by either CRLs or OCSP. For each method, the process looks similar:

- **Endorsement Revocation**: The endorsement can be revoked by submitting a special endorsement that contains additional information. This additional information cannot be found in a regular endorsement and highlights that the certificate owner does not further endorse the domain-account relationship. This transaction does not need to be sent from the same address used to create the endorsement, enabling control over the endorsement even if there is no control of the related account. The revocation takes place in the endorsement store database.

- **Certificate Revocation via CRL**: CRLs are lists of certificates that a given CA has revoked over time. These lists contain all certificates for the given CA. Therefore, they can become very long. Any entity can submit a transaction containing the CRL and the certificate fingerprint of the revoked certificate. If the certificate is stored in the certificate store database and has not been revoked previously, its state is updated to "revoked". The revocation takes place in the certificate store database.

- **Certificate Revocation via OCSP**: OCSP is a protocol that returns the information about the validity of a specific certificate, run by the CA that has issued the certificate in question. These responses are signed and can be verified similarly

to the certificate itself. Suppose the OSCP message claims that a specific certificate hash been revoked. In that case, that message can be sent to the smart contract, which in turn verifies that the message was issued by the relevant CA and updates the validity state of the certificate. The revocation takes place in the certificate store database.

Nonetheless, revocation faces a key issue in the system proposed in (Groschupp, 2020) and (Gallersdörfer et al., 2021c): The information about the revocation status is not available in real-time, as someone needs to submit the relevant data to the blockchain. Monitoring systems and individual incentive structures (e.g., rewards for withdrawn certificates) need to be implemented to ensure a secure on-chain environment.

#### Logical Centralization of Smart Contracts

"Logically" centralizing the storage and verification of both X.509 certificates and endorsements provides several advantages:

- **Reduced Overhead**: The code for parsing X.509 certificates, public key cryptography mechanisms, specific byte operations, and such are not deployed multiple times but only once for the entire ecosystem. This centralization also reduces costs, as (Gallersdörfer et al., 2021c) shows. We compare the respective costs with other approaches in Section 6.3.
- **Compatibility with EOAs and Smart Contracts**: As the register and system link to addresses and provide data on the authenticity of respective endorsements, an address to does not need to support a specific interface. Only in the case that an address wants to authenticate itself actively at a smart contract, this smart contract must be able to ask the system about the authenticity of the sender.
- **Endorsing Pre-Existing Smart Contracts**: Centralizing the endorsement and certificate database allows smart contracts to be endorsed later. A decentralized approach requires smart contracts to be redeployed.
- **Trustworthy Deployment**: Given that the system is deployed only once, entities that need to hardcode respective addresses (e.g., wallet developers) only need to verify the system's integrity once, protecting against malicious deployments of the system.

## 5.3. DNSSEC Integration in the Ethereum Name Service

In contrast to the approach above, the ENS leverages DNSSEC to utilize domain names in an on-chain context.

DNSSEC (as described in (Rose et al., 2005b,c)) ensures the authenticity of the records managed within DNS. Thereby, a hierarchical signature scheme, similar to a PKI, is established to allow the signing and verification of any records, such as `A`, `AAAA`, `CNAME` or `TXT`. As a trust anchor, only one key exists that signs respective subordinate zones (e.g., the zones responsible for the TLDs).

As only the owner of the domain can introduce changes to the DNS record, statements of the owner of the domain can be cryptographically verified. For that, a `TXT`-record can create an arbitrary text message that does not affect the intended functionality of DNS. To claim their domains within ENS, domain owners use a `TXT`-record to associate their domain names with Ethereum-specific addresses.

In the following, we give insight into the overall architecture of ENS. First, we look at the system in general and the *ENS registry* that manages all names. Then, we look at the DNSSEC-specific parts, namely the `DNSRegistrar` and the `DNSSECImpl`. Afterward, we discuss design decisions, their rationale, and their limitations. We develop a dataset containing all domain names bridged to ENS in Section 5.5. An extensive analysis using this dataset takes place in Chapter 6.

Within this section, we refer to the ENS documentation (Ethereum Name Service, 2023b) and the respective smart contracts, hosted on Etherscan (Etherscan, 2023a,b) and on the project's GitHub page (Ethereum Name Service, 2023a). Given that terminology is not used consistently within the ENS documentation and system, we adjust terms if required for consistency. All addresses used in our analysis are displayed in Appendix D for transparency.

### 5.3.1. Architecture Overview

ENS is structured similarly to DNS. It follows a hierarchical structure managed within the *ENS registry*. Each record[3] contains four relevant parts: a) a controller[4], b) a resolver, c) a registrant, and d) additional data (such as TTL). Controller, resolver, and registrants are Ethereum addresses and, therefore, can be smart contracts or EOAs.

[3] ENS refers to domains as *names* that consist of dot-concatenated *labels*, such as `alice.eth` being a name, whereas `alice` and `eth` are labels.

[4] ENS sometimes refers to a controller as an owner. The smart contract interfaces expose the term *owner*. We use *controller* as the web application uses *controller* as well.

ENS does not directly store the name of each record but instead uses *namehash*, a proprietary hash function that leverages *Keccak-256*. This enables the use of a fixed-length storage variable and provides efficiency gains. Name hashes have an interesting side-effect: ENS does not know which names are managed within their systems. ENS uses dictionary attacks to find the pre-image to the respective hashes, which is not always successful. In our analysis in Section 5.5, we can obtain all domain names bridged using DNSSEC, as the domain name is required for verification purposes.

The **controller** can transfer ownership, set the resolver, set further information, or create subdomains. The *registrant* owns the domain, can move the registration, and can redefine the controller if needed. The *resolver* is the smart contract providing additional information to the respective domain, e.g., links to cryptocurrency addresses, IPFS hashes, avatars, and more. With this approach, multiple goals can be achieved:

- **Separation of Ownership and Control**: The controller of an ENS domain must not be identical to the entity that registered the domain. With that, the private key to the registrant account can be kept in a cold wallet, whereas the controller's private key is kept in a hot wallet for active usage.
- **Upgradability**: Given that the resolver can be replaced anytime, its functionality can be extended as desired. If a user requires additional fields of storage within its resolver or requires some advanced functionality, the resolver is replaced.
- **Composability**: Smart contracts that were not part of the initial design process can be integrated easily. For example, if a DAO wants to own and manage an ENS name, it can introduce a governance mechanism by setting the controller to a respective smart contract.

In this hierarchical system, one entity, *root*, owns the entire namespace and delegates domains (in this case, TLD) to other entities in the system. For the TLD `.eth`, *root* handed over the rights to further distribute subdomains (e.g., `alice.eth`) to the ENS registrar.

The ENS registrar is a complex system allowing anyone to register `.eth`-domains. Domains need to be renewed every year, and depending on the length of the domain, prices vary. Given the ENS registrar's complexity and extensive analysis in (Xia et al., 2022), we refrain from further discussing its contents or functionality.

### DNSRegistrar

The DNSRegistrar is equivalent to the ENS registrar for the traditional DNS system. Instead of having an auction system that enables anyone to claim a subdomain of the respective TLD, the DNSRegistrar requires proof from the DNSSEC world to assign the domain to the necessary address. The DNSRegistrar needs to be bootstrapped: it a) needs the right to set up domains in the respective TLD[5], and b) needs to have the DNSSEC root key stored in its contract. The right to set the respective name in the ENS registry is automatically granted when the ENS root assigns the right to the DNSRegistrar.

The following steps are taken if an entity wants to register a domain within the DNSRegistrar.

1. The domain owner in DNS makes specific preparations and create a proof of ownership via DNSSEC. We provide a detailed description in Section 5.3.1.

2. The owner sets up an Ethereum address with sufficient funding (see 6.3 for evaluation of costs).

3. The owners enters the smart contract through one of either three functions: `claim`, `proveAndClaim`, or `proveAndClaimWithResolver`, accompanied by relevant information, such as which domain they want to claim and additional information.

4. The hierarchical key information, including signatures stored in RRSets, is sent to the `DNSSECImpl`-contract for inspection and verification. The `DNSSECImpl`-contract returns the proof to the registrar smart contract.

5. The contract verifies that the TLD is a *PublicSuffix*. For that, it calls a contract `PublicSuffixList` that returns whether the TLD is valid[6].

6. The contract recursively checks whether it already possesses the TLD by calling `enableNode()`. If not, it calls the root ENS contract and transfers the ownership of the TLD to itself. The DNSRegistrar has the right to assign new TLDs within the Root smart contract.

[5] That means the ENS root needs to grant the DNSRegistrar the right to manage the respective domain.
[6] This is currently the case for any TLD. It appears that the ENS developers are not interested in managing a list of ICANN-approved TLDs but rather accept any TLD as valid, given that ENS only manages `.eth`, which ICANN does not claim. Therefore, no naming conflicts exist, and any TLD that has not been registered before is granted to the DNSRegistrar.

7. Then, the registrar extracts the endorsed address from the proof and sends the information to the ENS registry, which sets the record and allows the owner to set respective resolvers.

### DNSSECImpl

The `DNSSECImpl` smart contract was initially deployed in August 2021. In its constructor transaction, the hash of the root public key was also set. Given its structure, it also requires the linking of several other smart contracts that implement algorithms to adhere to the DNSSEC specification. They were created shortly before the DNSSEC oracle contract. Table 5.1 gives an overview of these algorithms and their support by ENS. Although it appears that `NSEC3-SHA1` is supported, we cannot verify the correct functionality, as no details about it can be found in the documentation. Also we could not observe any transaction that called respective functions. Not all algorithms are relevant, but ENS supports the major algorithms; in particular, algorithm 8 is supported by all entities that support DNSSEC in general (see subsection 6.1.1).

| Number | Algorithm | Supported by ENS |
|---|---|---|
| 01 | RSA/MD5 | ✗ |
| 02 | Diffie-Hellman | ✗ |
| 03 | DSA/SHA-1 | ✗ |
| 05 | RSA/SHA-1 | ✓ |
| 07 | RSA/SHA1-NSEC3-SHA1 | ✓ |
| 08 | RSA/SHA-256 | ✓ |
| 10 | RSA/SHA-512 | ✗ |
| 12 | GOST | ✗ |
| 13 | ECDSA-256/SHA-256 | ✓ |
| 14 | ECDSA-384/SHA-384 | ✗ |
| 15 | ED25519 | ✗ |
| 16 | ED448 | ✗ |

**Table 5.1.:** DNSSEC cryptographic algorithm numbers and their ENS support. Numbers available in (ICANN, 2023)

In theory, there are two functions to enter the smart contract; `submitRRSet` and `submitRRSets`, but only the latter is called in practice. Both functions work almost identically, given that the latter iterates only over a list of RRSets whereas the first function processes one RRSet.

Generally, the approach is identical to verifying a record using DNSSEC. The proof mechanism iterates over the RRSets and their respective proofs, starting from a well

known DNSKey and iterating over RRSets until the last RRSet is reached. This proof is then returned to the sender; in the case of the verification process, it is the DNSRegistrar.

Let us be more concrete and consider an example for the domain `eth.limo`, registered on the 24th February 2023. The transaction called the `ProveAndClaimWithResolver`-function. The sender provided five RRSets that we inspect in detail. They are displayed in Table 5.2. In addition to these RRSets, the sender also provides the initial proof as a public key.

| # | Type | Algo | Inception | Expiry | KeyTag | SignerName |
|---|---|---|---|---|---|---|
| **1** | DS (43) | 8 | 2023-02-24 | 2023-03-09 | 951 | . |
| **2** | DNSKEY (48) | 8 | 2023-02-17 | 2023-03-10 | 39862 | limo. |
| **3** | DS (43) | 8 | 2023-02-17 | 2023-03-10 | 61575 | limo. |
| **4** | DNSKEY (48) | 13 | 2023-02-24 | 2023-02-25 | 21598 | eth.limo. |
| **5** | TXT (16) | 13 | 2023-02-24 | 2023-02-24 | 39519 | eth.limo. |

**Table 5.2.:** Five RRSets provided at the registration of `eth.limo` in ENS.

Now we can iterate over each row:

1. Contains a DS record, uses the algorithm 8 and refers to the keytag 951, which is the zone singing key for the root (.) zone. Given that this key was newly created[7], the entity registering this address was required to prove the validity of this RRSet with the initially provided proof. Including this RRset within the registry allows the creation of a new proof that can be used consecutively for the next RRSet.

2. Contains the DNSKEY record which includes the key signing key for the .limo-zone (given the keytag 39862). Its value is proven by the previous line and allows us to prove the validity of the next line.

3. The DS, again, referring to one of the two zone signing keys, is then leveraged for further proofs.

4. For the following zone (eth.limo), a DNSKEY is recorded again. Here, another algorithm (13) is leveraged, which is still supported by ENS.

5. Lastly, the TXT record for the subdomain (_ens.eth.limo) is introduced and verified in the storage.

[7] `eth.limo` was the first domain registered since the key is available.

If the DNSRegistrar wants to verify that the domain is owned, it uses the address provided by the sender and builds the record itself, as displayed in Listing 5.1. Then, it verifies that this record is authentic, meaning the proof for the respective record exists. If a record exists, it was previously confirmed and stored within the database.

```
1 _ens.eth.limo. 60 IN TXT "a=0x989A...29d3F"
```

**Listing 5.1:** TXT-record for _ens.eth.limo. Address shortened for readability.

That concludes the verification. The registrar grants the domain to the address in the record.

## 5.4. Rationale in System Architecture Designs and Decisions

In ENS, two systems coexist. The traditional DNS system has over 2000 TLDs, whereas ENS manages only the TLD `.eth`. This creates a need for a **separation of concerns**. The ENS root grants any TLD that has not been registered previously, and a proof can be presented to the DNSRegistrar; the only other TLD `.eth` remains in control of the ENSRegistrar, enabling anyone to purchase respective domains. Given the setup, we do not think a different approach would make sense or allow the replacement of individual components if updates were available. In contrast, the on-chain approach we outlined does not co-exist with a proprietary system, and therefore, we do not expect interplays with existing systems on-chain.

**Key storage** is a rationale within both systems. ENS stores the RRSets, and the on-chain approach stores TLS certificates and their respective keys. Given that both solutions profit from pre-existing keys that do not need to be validated again, key storage makes sense in both cases. We take a closer look at the problem of storing these records. If the volume of registrations is low, storing the sets might incur more costs than just pushing them on-chain every time, as these records are very short-lived.

The **validity of the initial endorsements** differ in both approaches: Whereas in ENS, the registration never expires, the on-chain system leveraging TLS certificates adheres to the validity of the respective certificate. In ENS, this is a design decision that we believe is driven by economic necessities. For TLS certificates, the validity can be a maximum of one year; Let's Encrypt even issues certificates valid for only 90 days. While it can be reasonable to refresh the TLS certificate on-chain every three months, the validity periods of the RRSets in DNSSEC are much shorter. For the last record in

Table 5.2, validity was a mere 121 minutes. This timeframe is too short to force entities to re-validate. Whether to periodically force re-validations after a specific time to prevent too many abandoned domains within the ENS, is discussed in the next section.

The **proof of ownership** also deviates between the two approaches. In ENS, control over the subdomain `_ens` (e.g., `_ens.example.org`) must be proven. In the on-chain TLS approach, a valid certificate for the actual domain is required. We did not find any rationale for why the `TXT` record must be set in the specific sub-domain instead of the actual domain. A potential reason could be that including the `TXT` record within the regular domain could largely inflate proof sizes; often, other entities that require ownership proof of domains also place `TXT` records within the domain. As RRsets include all records of one type, respective proofs become more complex. However, this also leads to a potential security issue, as a controller of a subdomain (albeit being the specific `_ens.` subdomain) could masquerade as the parent domain. We discuss this security issue in subsection 6.4.3.

The subsequent **control over records** is worthy a discussion. In the TLS on-chain (and off-chain) approach, the owner of the domain sets all rules and requirements: address, validity, rules, flags, and more, limiting the scope of action for the on-chain address. In ENS, the address receives complete control over all concerns; it can change, replace and update any information associated with the domain in the blockchain context. It can even register sub-domains, overriding the domain owner's decisions. We believe that there are economic reasons for this. Verifying that the sender equals the previously defined owner is easy, and thus, subsequent changes are cheap. If one had to issue a new cryptographic proof from DNSSEC to make a change, costs would rise quickly. This design decision also limits the issuance of subdomains in the on-chain context. Using the TLS on-chain approach, subdomains can be issued, whereas using ENS, the address needs to issue respective subdomains.

Also, **revocation** is more straightforward in the TLS on-chain approach. A signed statement of the certificate can be published to the chain, and the respective smart contract considers the endorsement invalid. For ENS, the situation is different. While in theory, DNSSEC can prove that a record does not exist, practically, the functionality is either unavailable or untested (see Section 5.3.1). Revocation is a rare scenario for TLS certificates, and in DNSSEC, key revocation does not exist. In a system where records are indefinitely valid, users should be instructed to revoke issued domains if they stop using them.

Lastly, a difference between the two approaches is the existence of **user configurations**.

In contrast to TLS, DNSSEC has the advantage of having a single point of trust, the root entity. In the TLS certificate ecosystem, multiple entities exist that can be considered trustworthy. Also, new, reliable entities (e.g., in an enterprise context) need to be considered. ENS, which follows a similar approach to DNS, needs one single point of truth and cannot deviate or allow users to choose which entities they trust; relying on DNSSEC is a helpful approach.

## 5.5. ENS DNSSEC Domain Dataset

To our knowledge, no extensive analysis of the domains transferred via DNSSEC in ENS has occurred. (Xia et al., 2022) have analyzed ENS to a great extent for regular `.eth`-domains, but they only partly covered domain names transferred from DNS. They analyze only domains with the endings `.xyz`, `.club`, `.luxe`, `.art`, and `.kred` that rely on a custom implementation with the registrar[8]. They did not include the DNSSEC registrar that can migrate any TLD.

Therefore, we develop a dataset of all domains migrated from DNSSEC to ENS as of 27$^{\text{th}}$ February 2023. We employ the following steps: 1) data collection, 2) data cleanup, 3) data enrichment, and 4) data verification.

### 5.5.1. Data Collection

Given that we focus only on the DNSSEC registrar, we need to focus only on a single address within the ENS ecosystem: `0x58774bb8acd458a640af0b88238369a167546ef2` is the address that manages all TLDs from the "traditional" DNS system. Its source code can be obtained from Etherscan.io (Etherscan, 2023a). It was created on 17$^{\text{th}}$ August 2021. While the proposal for leveraging DNSSEC exists for much longer, this is the first occurrence on the main net, to our knowledge. We analyze the domain names up to the 27$^{\text{th}}$ February 2023; the last domain in our dataset was recorded on the 26$^{\text{th}}$ February 2023. The smart contract exposes three methods that are used to prove the ownership of a domain:

- `proveAndClaim(name, input, proof)`: Can be considered the standard functionality. A user submits information to the DNSRegistrar, including the domain name, RRSets as an input, and proof of ownership.

[8] See `https://medium.com/coinmonks/an-overview-of-ethereum-name-service-ens-e736d0b946ba`, accessed 27$^{\text{th}}$ February 2023.

- `proveAndClaimWithResolver(name, input, proof, resolver, addr)`: This function extends the initial function by providing the same interface as `proveAndClaim` but additionally allows a resolver address as well as a deviating owner to be set.
- `claim(name, proof)`: This function provides just the basic claim functionality, which requires the user to submit the RRSets in the DNSSECImpl contract beforehand.

It also supports the event `Claim(bytes32 indexed node, address indexed owner, bytes dnsname)` that gives insight into the respective claimed domains. However, as we want to gain additional insights into the transactions (e.g., how many failed), we access all transactions that call the functions mentioned above.

We use Dune Analytics to crawl all transactions for the three calls (Dune Analytics AS, 2023). Dune conveniently provides decoded transaction information, such that we can easily query the data required for analysis. As we want to get relevant transaction data alongside the actual call, we join the table of the respective call with the transaction table. We employ SQL-queries as shown in Listing 5.2, Listing 5.3, and Listing 5.4.

```
1 SELECT call_block_number, call_block_time, call_success,
    call_tx_hash, gas_price, gas_used, nonce, value, from, to, name,
     proof
2 FROM   ens_ethereum.dnsregistrar_call_claim AS ens,
3        ethereum.transactions AS tx
4 WHERE  ens.call_tx_hash = tx.hash
```

**Listing 5.2:** SQL-Query for fetching all transactions calling the `claim` function on the `DNSRegistrar`.

```
1 SELECT call_block_number, call_block_time, call_success,
    call_tx_hash, gas_price, gas_used, nonce, value, from, to, name,
     input, proof
2 FROM   ens_ethereum.dnsregistrar_call_proveandclaim AS ens,
3        ethereum.transactions AS tx
4 WHERE  ens.call_tx_hash = tx.hash
```

**Listing 5.3:** SQL-Query for fetching all transactions calling the `proveAndClaim` function on the `DNSRegistrar`.

```
1 SELECT call_block_number, call_block_time, call_success,
    call_tx_hash, gas_price, gas_used, nonce, value, from, to, name,
     input, proof, resolver
```

```
2 FROM   ens_ethereum.dnsregistrar_call_proveandclaimwithresolver AS
    ens,
3        ethereum.transactions AS tx
4 WHERE  ens.call_tx_hash = tx.hash
```

**Listing 5.4:** SQL-Query for fetching all transactions calling the `proveAndClaimWithResolver` function on the `DNSRegistrar`.

We give an overview of the respective data in Table 5.3.

| Method | `claim` | `proveAndClaim` | `proveAndClaimWithResolver` | Total |
|---|---|---|---|---|
| Call Count | 16 | 190 | 904 | 1,110 |
| Success | 16 (100%) | 169 (88.9%) | 798 (88.3%) | 983 |
| Failed | 0 (0%) | 21 (11.1%) | 106 (11.7%) | 127 |
| Unique FQDN | 12 | 169 | 833 | 877 |

**Table 5.3.:** Preliminary insights into the data as of 27th February 2023. For each method, we display the total transaction calls, how many of them were (not) successful (including shares), and their respective unique domains. The total number of all unique FQDNs for the three methods is 925. After accounting for duplicates across the methods, the number of unique domains is 877.

## 5.5.2. Data Cleanup

In the next step, we need to clean the data we obtained from Dune. First, we remove all transactions that have failed. Although the share of failed transactions is relatively high, we attribute them to the general issue of hard-to-set gas fees in Ethereum and do not consider them further in our domain name analysis.

Second, we must translate the domain names obtained in the dataset to human-readable names. We display the respective encoding scheme in Figure 5.1.

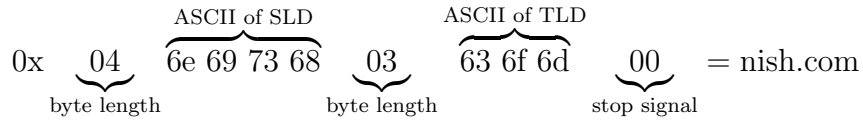



**Figure 5.1.:** Encoding scheme for domain names within the DNSRegistrar. This example translates to `nish.com`.

With that, we can look more closely at the actual domain names and the TLDs used. We display the list of the top 10 TLDs in descending occurrence in Table 5.4. Interestingly, we can find 124 different TLDs in ENS. We provide a treemap in Appendix E to better

understand the used TLDs. We use data from IANA to label the TLDs with the types *generic*, *generic-restricted*, or *country-code* (IANA, 2023b).

| TLDs | Occurrences |
|---|---|
| .com | 234 |
| .xyz | 185 |
| .io | 66 |
| .dev | 43 |
| .org | 27 |
| .net | 21 |
| .id | 20 |
| .me | 17 |
| .wtf | 16 |
| .app | 10 |

**Table 5.4.:** Top 10 top-level domains and their frequency in the ENS system.

For each RRSet, a single hexadecimal string contains all the necessary information of the respective set. However, we do not find a library that decodes these strings and instead decide to reverse-engineer the respective format from the solidity smart contract. We display the encoding scheme in Figure 5.2. After we decode this information, our transaction dataset is complete.

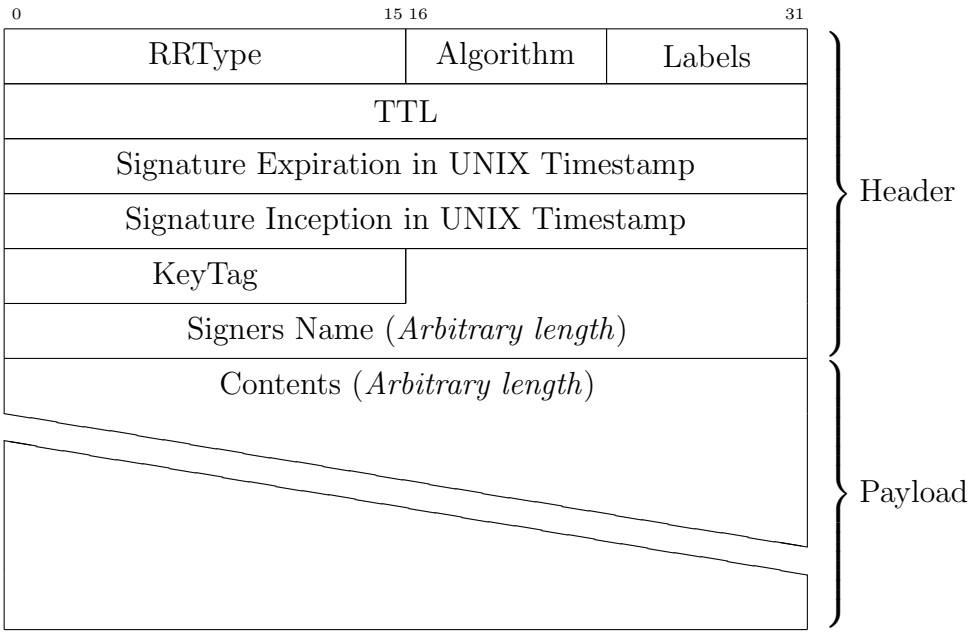



**Figure 5.2.:** Data structure of RRSets used in ENS.

### 5.5.3. Data Enrichment

The previous steps lead to a dataset that contains all domain names that have been bridged successfully to ENS from DNSSEC. While this dataset provides preliminary insights, we are interested in additional information for further analysis in this chapter. More precisely, we are interested in three variables:

- **Hosted Website**: Is there a website hosted on the domain? This indicates whether the system might be used to prevent address replacement attacks, which we initially set out to address. To check whether a website is available, we call the site and record the `HTTP`-response.
- **DNSSEC Record**: Users need to set the DNSSEC record only once for an `_ens.`-subdomain, which proves the entry to the DNSSEC oracle. Afterward, they can, in theory, remove the entry. However, removing the entry could have other reasons, such as change of ownership. Therefore, we are interested in whether the record is still set and identical to the initially associated address. We use *DNSPython* (Dnspython, 2023) to crawl the `TXT`-records of the individual subdomain.
- **Domain Availability**: In rare cases, entities might abandon their domains and allow them to be registered by anyone. If the domain is still used for incoming funds by other users, attackers could register the domains, claim the domain in ENS, and redirect funds to themselves. We bulk-submit all domains for which we do not find a website and a DNS record to *UnitedDomains*[9] to understand whether they are free to register. We further investigate this security issue in Section 6.4.

Table 5.5 gives an overview over the **Hosted Website** variable. Only slightly more than half of all domains even responded to our requests, and most of them responded with a Status Code 200, meaning that the website is reachable. One domain responded with an unspecified HTTP code (999). Upon manual inspection, it turns out that the domain directly forwards to *LinkedIn*, which blocks Python's `requests`-package and returns the specific 999 error (http.dev, 2023).

Further, Table 5.6 provides data on the specific **DNSSEC record** in the `_ens.`-subdomain. We define three states $s_i = \{0, 1, 2\}$, whereas $s_0$ means that no record is available, $s_1$ a record is available, and is identical to the sender of the transaction and $s_2$ a record

[9] See `https://www.united-domains.de/`, accessed 27th February 2023.

| HTTP Status Code | Message | Occurrences |
|---|---|---|
| 200 | `OK` | 430 |
| 400 | `Bad request` | 1 |
| 401 | `Unauthorized` | 2 |
| 403 | `Forbidden` | 17 |
| 404 | `Not Found` | 18 |
| 406 | `Not Acceptable` | 5 |
| 444 | `No Response` | 1 |
| 500 | `Internal Server Error` | 1 |
| 502 | `Bad Gateway` | 8 |
| 503 | `Service Unavailable` | 3 |
| 522 | `Connection Timed Out` | 2 |
| 526 | `Invalid SSL certificate` | 1 |
| 999 | *Undefined* | 1 |
| n.a. | *No Response* | 387 |

**Table 5.5.:** Response codes when requesting the domain contents.

is available and is not identical to the sender. $s_2$ is not uncommon, given that an entity can easily own two addresses and use one address to set up a record for another address.

| Status | $s_0$ | $s_1$ | $s_2$ |
|---|---|---|---|
| Count | 115 (13.1%) | 666 (75.9%) | 96 (10.9%) |

**Table 5.6.:** Status of the DNSSEC-records as of 27$^{\text{th}}$ February 2023 for all DNS domains registered on ENS. ($s_0$ : No record available. $s_1$ : Available. $s_2$ : Available, but address deviates from sender.)

### 5.5.4. Data Verification

To verify the integrity of our data, we compare it to the online application of ENS[10]. We verify that a sample of domains in our dataset are identical to the data displayed in the application. Further, we confirm that single domain names displayed in the application also appear in our dataset. Last, we compare the count for TLDs in the application to our dataset. We find no deviation from our dataset, therefore we assume it to be complete for the given timeframe.

[10]See `https://app.ens.domains/`, accessed 27$^{\text{th}}$ February 2023.

## 5.6. Summary

In this chapter, we discussed system designs that facilitate the storage and verification of certificates and their signatures in an on-chain context, both for DNSSEC and the TLS PKI. Both approaches remove the need for any interactions or processes outside the blockchain to assert the authenticity of addresses on-chain. Furthermore, we showed:

- A system design that mirrors domain names to an on-chain environment is a worthwhile goal. For example, the system allows for address-independent payments, abstracting the correct mapping from addresses to identities.
- Creating a usable representation of TLS certificates (or any X.509 certificates) is feasible within the capabilities of today's blockchain network. While the system is more complex than the architecture proposed in Chapter 4, it allows for advanced use cases.
- Issues that stem from an off-chain verification approach as outlined in Chapter 4 are no longer present, as all information is verified in an on-chain context.
- ENS leverages DNSSEC within its existing infrastructure and uses `TXT`-records equivalent to the endorsement, leading to specific deviating choices for system design.
- Domains that have already been transferred from traditional DNS to ENS can be extracted, analyzed, and contextualized in our research. We find a total of 1,110 transactions sent to the respective contract.

In the next chapter, we evaluate and compare all techniques outlined in this dissertation for different metrics, including the suitability of ecosystems, applicability, security, costs, and fulfillment of requirements.

# Chapter 6.

# Evaluation and Comparison of Approaches

The system architectures outlined and discussed in Chapters 4 and 5 mitigate loose coupling between on-chain entities and real-world entities by creating a cryptographically verifiable binding between these two worlds. To fully understand strengths, impact, opportunities, and limitations, we evaluate the off-chain, on-chain, and ENS-approach in distinct dimensions in more detail:

- In Section 6.1, *"Suitability of Ecosystems"*, we evaluate the suitability and fitness of the TLS/X.509 certificate ecosystem and the DNSSEC ecosystem to enable the secure mapping between domain names and on-chain entities.
- We discuss the applicability as well as the advantages and limitations of the practicability of these systems in Section 6.2, *"Applicability and Practicability"*. Further, as data on the DNSSEC integration for ENS is available, we analyze its adoption within the Ethereum community.
- As the costs of usage of layer 1 blockchain networks have increased, we discuss key metrics of gas usage and the resulting costs for the Ethereum mainnet in Section 6.3 *"Costs"*.
- In Section 6.4, *"Assessment of Security"*, we analyze and evaluate the security assumptions of all approaches.
- Lastly, we analyze the fulfillment of the requirements initially posed by use cases as well as RFC documents in Section 6.5, *"Requirements"*.

This chapter relies partly on material and insights from prior publications. In Section 6.2, we briefly take up and expand discussions from (Ebel, 2021) and (Gallersdörfer et al., 2021a). In sections 6.3 and 6.4, we use the information on gas prices and security considerations of the off-chain approach developed in (Gallersdörfer and Matthes, 2021a) and of the on-chain approach developed in (Groschupp, 2020) and (Gallersdörfer et al., 2021c).

## 6.1. Suitability of Ecosystems

The three systems we describe in this thesis rely on the TLS or the DNSSEC ecosystem. Both deviate in history and functionality. Nonetheless, they can be leveraged for the same target. In this section, we discuss the implications of the ecosystem properties on the respective systems. Thereby we differentiate two concepts:

**6.1.1 Availability**: Is the underlying technology (TLS or DNSSEC) available for the respective domain?

**6.1.2 Usage**: Is the underlying technology (TLS or DNSSEC) actively leveraged by the respective domain?

We provide a summary and implications on using TLS and DNSSEC in subsection 6.1.3.

### 6.1.1. Availability

**TLS certificates**, including respective private keys, were initially safeguarded by entities that charged fees for their provisioning. Depending on certificate type (e.g., extended validation) and scope of validity (e.g., wildcard-certificates), prices still vary today[1]. Users also needed technical expertise to set up web servers with certificates.

This state has changed in the past. Given the increasing surveillance of web connections, encryption and secure authentication of counterparties became increasingly necessary. In 2015, the company *Let's Encrypt* was founded. Since then, their motto has been *"encrypt the entire web"* (Let's Encrypt, 2023). They provide TLS certificates upon request to any domain and without charge. *Let's Encrypt* was also the proposer for the ACME protocol, which we introduced in 2.1.2, which enables the automatic request and retrieval of TLS certificates. According to their latest report in November 2022, they issued over 3 billion certificates and have actively provided about 240 million websites with certificates (ISRG, 2023). Given that the total count of registered domains is about 651 million (Domain Name Stat, 2023), *Let's Encrypt* serves about one-third of the entire WWW with TLS certificates.

To our knowledge, *Let's Encrypt* supports any TLD registered in IANA's root zone database. That means TLS is available for every domain free of charge. Beyond these numbers, it is hard to gauge how easy it is for novices to obtain a certificate, e.g., what share of web hosters directly provide a certificate or simple means of getting one.

[1] See `https://www.digicert.com/tls-ssl/compare-certificates`, accessed 28th February 2023.

**DNSSEC** is a relatively novel technology compared to TLS; therefore, its support is not as widespread. Compared to the initial phase of TLS, DNSSEC does not come with additional registrar costs. Large providers such as AWS provide DNSSEC free of charge within their products[2].

However, given its hierarchical structure and direct management within the DNS, DNSSEC support can break at multiple places. First, the TLD needs to support DNSSEC for the underlying name servers to support DNSSEC. If a third-party entity manages the underlying name servers, it must also support DNSSEC. Lastly, the domain owner must set the respective records.

Public lists for DNSSEC support in TLDs exist (Openprovider, 2023). Of the 2532 listed TLDs (including TLDs like `.co.uk`), 248 do not support any algorithm for DNSSEC, which is about 10% of all TLDs. However, the most popular TLDs support DNSSEC, whereas TLDs that do not support DNSSEC are relatively uncommon. To our knowledge, no precise number exists. Therefore, we combine two sources, namely (Openprovider, 2023) for the list of TLDs that support DNSSEC and (Domain Name Stat, 2023) for the number of registered domains per TLD. Of the initial 2532 TLDs, (Domain Name Stat, 2023) could provide data for 1302 TLDs[3]. With this approach, we have data for 51% of TLDs. Still, we cover about 97% of all domains in this calculation[4]. Table 6.1 gives an overview of the respective sum of live domains under a TLD that either supports DNSSEC or not. We find that only 1.7% of all domains currently cannot enable DNSSEC support. This number is much smaller than the initially outlined 10%, given that almost all large TLDs support DNSSEC. Only `.ga`, as the 11$^{\text{th}}$-largest TLD, does not support DNSSEC. However, 98.3% is an upper bound and is likely lower given that not all name servers or hosters support DNSSEC for their customers. We provide a treemap of all TLDs, including their DNSSEC support in Appendix F.

| | DNSSEC ✓ | DNSSEC ✗ |
|---|---|---|
| Number of Domains | 625,085,556 (98.3%) | 10,621,345 (1.7%) |

**Table 6.1.:** Number of domains managed by TLDs supporting DNSSEC, as of 28$^{\text{th}}$ February 2023.

[2]See `https://aws.amazon.com/route53/pricing`, accessed 28$^{\text{th}}$ February 2023.

[3]This is due to the initial dataset containing not only TLDs, but also Second-layer domains, which we do not have domain numbers for. The number deviates only slightly because the respective TLD is included in its higher level TLD.

[4]Our calculation considers 635 million domains, (Domain Name Stat, 2023) has 651 million domains in their dataset.

### 6.1.2. Usage

The analysis of the availability of the technology under discussion is extensive and covers large parts of the DNS. Regarding usability, the numbers are harder to obtain; they must be directly crawled or observed to understand whether a domain uses TLS or DNSSEC.

**TLS certificates** are certainly widely adopted. Google provides in their transparency report insights into encryption on the web and browsers (Google, 2023). Given that they have access to statistics that their users share via the Chrome browser on all major platforms, they have a general overview of the landscape. Their report finds that a) depending on the platform, 80% (Linux) to 99% (Chrome Platform) of all requested websites were served via HTTPS, and b) all top 100 websites support HTTPS.

The adoption of encryption and TLS certificates is omnipresent. *HTTPS Everywhere*, a browser plugin by the Electronic Frontier Foundation (EFF), automatically switched to the HTTPS protocol when present. EFF announced the deprecation of HE, as HTTPS can be considered fully adopted (EFF, 2021).

**DNSSEC** has not yet reached full adoption on the WWW. Usage statistics are available on a region and world basis in (APNIC Labs, 2023). Given that there is already a strong adoption of TLS, the necessity for DNSSEC is not immediately apparent, given the additional overhead in properly managing all key material (Huston, 2023). The DNSSEC validation rate as of 27$^{\text{th}}$ February 2023 is 31.6%. It is unclear whether adoption will significantly rise, given that it has been higher in the past.

### 6.1.3. Implications for Usage

Overall, both systems are widely accepted in the WWW. TLS has almost full adoption and support worldwide, with potentially only limitations coming from hosters or providers (such as access to the certificate's private key). DNSSEC also has a significant share of support and supports all major domains. Parties interested in developing or designing systems that migrate domain names to blockchain networks must consider other factors besides adoption (such as intended use case) to decide on TLS or DNSSEC as an underlying technology.

## 6.2. Applicability and Practicability

Understanding the applicability and practicability of the systems described in this thesis is highly important. Regardless of technological capabilities, security enhancements,

audited code, or else: If the application has no ease of use for both the end users that interact with smart contracts and the entities that set up respective smart contracts, no significant number of users will adopt the solution. To understand the usability implications of the full scope of the application, we structure this section as follows:

**6.2.1** *Technical Challenges*: We outline the implications of technical challenges on the applicability of the systems, such as access to TLS certificates in respective contexts.

**6.2.2** *Autonomy and Simplicity*: The described systems enable autonomy for their users; however, this is at the cost of simplicity and comprehension.

**6.2.3** *Ecosystem Bootstrapping*: Despite relying on pre-existing TLS certificates, the systems still need to find adoption by user-facing software and institutions alike.

**6.2.4** *ENS DNSSEC Bootstrapping*: Given that data is available on the adoption of DNS-rooted domains in ENS, we give an overview of these bootstrapping efforts.

**6.2.5** *Further Complexities*: Further complexities arise for users and issuers alike.

## 6.2.1. Technical Challenges

A multitude of technical challenges can hinder applicability and usability.

**Major browsers'**, such as Chrome, Edge, Opera, and Safari, **lack of support for the retrieval of TLS certificates** of recently established connections to web servers **render the off-chain verification approach questionable**. As outlined in Section 4.4.3, there are ways to utilize the system design in which the verification of certificates takes place on the users' machine. However, we find that they are unsuitable, either from a security or privacy perspective (like a centralized approach), or that the usability suffers (like with the installation of a local script). Only moving the verification of endorsements on-chain allows the plugin to access and assess the certificate's information. As we later see in Section 6.3, this is also a question of costs.

Further technical complexity awaits when moving to a purely on-chain-focused system. For example, **blockchain networks often support only specific public key cryptography** algorithms while omitting others, e.g., RSA. This leads to the problem that these algorithms must be implemented in smart contracts and deployed on the respective chains, potentially leading to security risks and additional costs. To properly roll out a system that a) supports many cryptographic algorithms, b) is well-tested, and

c) properly audited is a resource-intensive endeavor that would only ensure the bare minimum requirements for such a proposal.

The system's complexity would require setting up a **governance scheme for the smart contract ecosystem**. It can be expected that the system and its components need updates and extensions over time, for example, to support new features or cryptographic algorithms. In such cases, smart contracts would need to be updated. For these update processes, proper governance mechanisms need to be in place to prevent fragmentation of the smart contracts that are deployed for the system. Furthermore, backward compatibility must be ensured such that once-deployed contracts can still access the system's services. Update mechanisms can become a security issue or introduce new problems previously unknown to the system (Parity Technologies, 2017).

### 6.2.2. Autonomy and Simplicity

The autonomy of users and the simplicity of a system and its applications can be conflicting targets. For every decision the engineer leaves up for the user to decide, the user must make a conscious choice. At first, this approach strengthens users' autonomy; however, it comes with issues:

- More decisions will likely result in reduced usability,
- Users need to be educated to make well-informed decisions,
- A pre-selection of choices needs to be made, and
- Decisions can come with unintended consequences.

The contrast between autonomy and simplicity in the context of our proposed system applies to the selection of trusted root certificate authorities. Both TLS systems are designed so the end-user can decide which certificate authorities to trust, allowing them to use any PKI or even deploy their own. If users have the autonomy to decide, then the question remains: How will this autonomous decision-making be available in practice? Selecting individual CAs to trust is too complex for the regular user, whereas proposing standard configurations can be seen as biased. A governing body would be required to introduce new CAs to this predefined list or to remove untrustworthy entities. If the list is not moderated, the incentive for illicit activity rises. There is a comparable situation in the decentralized exchange Uniswap. Uniswap allows to list any token, potentially creating duplicates and fakes of original tokens. Preliminary research suggests that

over 95 % of all listed tokens are intended to scam users (Mazorra et al., 2022). The same situation can be envisioned in cases where the list of acceptable root CAs is not moderated.

### 6.2.3. Ecosystem Bootstrapping

One of the reasons for the systems to rely on TLS certificates is that they are already a commodity: There is widespread usage (as outlined in subsection 6.1.1) of the certificates, they are easy to obtain, they cost next to nothing, and almost all browsers and software vendors in the WWW acknowledge these certificates. This property makes them compelling to use. Users are already familiar with domain names and know how to use them. They also might have some idea of the security protocol, even if it does not go further than understanding that a regular closed lock in their URL bar is a good sign. Leveraging these certificates in the context of blockchains should be simple, given their already widespread adoption.

However, these certificates are not a means to an end by themselves. We intend to leverage them in a new context, namely in an on-chain environment. Leveraging these certificates requires us to introduce a new data structure that deals with on-chain environment specifics. That data structure (the endorsement), although it can be directly generated from a TLS certificate, has not been adopted so far.

Not only does the endorsement have no adoption but any related software also does not support the system:

- **Wallet software**: Ideally, any wallet software should support our proposed system. Not only does this require lots of programming work, but it also might lead to similar problems we found when augmenting the browser plugin MetaMask. Other types of wallets could face similar issues. Although TLS certificates are widely available, their access might be non-trivial. For example, website-based wallets[5] (such as MyEtherWallet[6]) might only be able to access TLS certificates if the requested host allows cross-domain requests, requiring solutions, as outlined in Section 4.4.3.

- **Development environments**: Software engineering tools for blockchain environments are broadly available and enhance the teams' performance. Integration or support for the described systems is required to allow the responsible entities to

[5]Wallets that purely rely on Javascript provided by a website without relying on a backend.
[6]See `https://www.myetherwallet.com/`, accessed on 16th February 2023.

publish smart contracts that adhere to the respective standard. While the target audience is smaller, the integration increases the overall complexity of the respective smart contract systems.

- **Automation**: Given the short lifetime of TLS certificates, they and their respective endorsements need to be updated periodically. While for TLS certificates, systems such as ACME exist that deal with automatic renewal, they also need to integrate the renewal of the endorsements. Such automation leads to either having both keys (for the certificate and for the blockchain address) on the same machine or more error-prone processes to sign-and-transfer schemes between the two machines. From a security perspective, it makes sense to separate keys, but it further increases complexity and maintenance.

- **Monitoring**: Monitoring must be developed and deployed to leverage the system's full capacity. While services exist that leverage CT for certificate discovery and retrieval, they potentially do not support blockchain networks. Given the high requirements for storage[7], a fast and efficient monitoring system is costly to build.

Indeed, a similar system that does not rely on TLS certificates would also be required to bootstrap a recognizable and accepted naming scheme. Still, the advantage of bootstrapping is not as significant as initially expected.

### 6.2.4. ENS DNSSEC Bootstrapping

Given the data set that we generated in Section 5.5, we can lay out the adoption of the DNSSEC approach of ENS. We further provide an overview of total ENS registrations. We also briefly overview how wallets and other software use real domains besides `.eth`-domains.

#### Registrations

We observe that in the timeframe of 17$^{\text{th}}$ September 2021 to 28$^{\text{th}}$ March 2023, about 2.67 million `.eth` domains have been registered (Dune Analytics AS, 2023). In comparison to the number of domains managed in traditional DNS TLDs, `.eth` would rank 36$^{\text{th}}$ between `.se` (2.70 million domains) and `.loan` (2.64 million domains) (Domain Name Stat, 2023). It is interesting to see the high demand for ENS domains. However, further

[7] A Geth node, storing the entire Ethereum blockchain, requires over 13 TB storage as of February 2023 (Etherscan, 2023c).

research is warranted, as it is unclear to what extent ENS domains (and regular domains) are leveraged for speculative purposes. In contrast to the registration number of 2.67 million domains, the system counted only about 665,000 unique participants and only 495,000 set primary names[8]. Therefore, only about 17.8% of ENS domains are actively used. However, it is unclear to what extent this also applies to regular domain names in the WWW.

Further looking into domain registrations that leverage DNSSEC, adoption looks negligible, especially compared to `.eth`-domains. During the same timeframe in which 2.67 *.eth* domains were registered, we found only 877 domains registered using the DNSSEC approach. Figure 6.1 displays the accumulated domain registrations over time with a logarithmic scale.

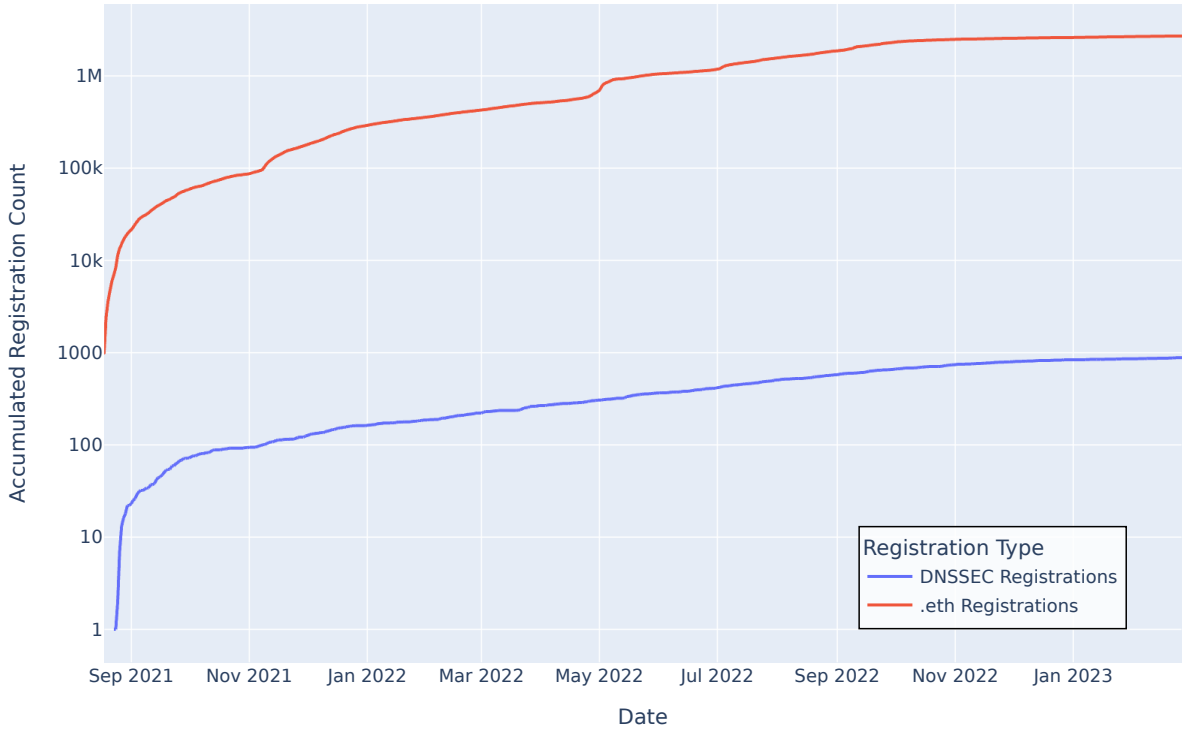



**Figure 6.1.:** Accumulated number of registrations for `.eth` domains and DNSSEC-enabled domains from 17th August 2021 until 27th February 2023. Registrations that took place before the 17th August 2021 are not considered.

We can only speculate why the adoption of domain names using DNSSEC is so low in ENS. Multiple reasons can be considered:

[8] *Primary Name* is the main resolution from domain name to address. Compare `https://dune.com/makoto/ens`, accessed 15th March 2023.

- **Too Expensive:** Transaction fees can be a considerable problem, as we outline in Section 6.3. Paying a lot of money compared to buying an ENS domain can be an argument, especially as use cases for blockchain networks are scarce. Leveraging a new domain instead of an established one might not bring the expected benefits.
- **Not Advertised as Not Revenue-generating**: It seems that ENS does not heavily advertise its solution for bridging existing domains to ENS. A reason could be that it is not generating any profits, as only Ethereum's transaction fees apply. Investing in a technology that does not directly yield monetary benefits can be a difficult decision to make.
- **Too Complicated**: The users intending to transfer their domains to ENS must understand varying concepts and handle interfaces for setting DNS records. Setting, gathering, and submitting information can be tiresome, and users might stop their attempts to transfer a domain.

We highlight this issue again in Section 7.3.

#### Wallet and Software Address Support

ENS claims 530 integrations into wallets, browsers, and other software (Ethereum Name Service, 2023c). We install and test selected tools and make the following observations:

**No DNSSEC verification** While the ENS app[9] displays a warning in case of a DNSSEC error, other applications were unaware that a) the domain is a DNSSEC-enabled domain and b) that caution should be exercised in case a DNSSEC record is no longer set. This can be a sign of an attack or an abandoned domain, as outlined in Section 6.4. Obviously, the application should communicate the state to the user; however, developers might be reluctant to integrate the functionality into their system, as this increases the complexity tremendously. For now, the applications only need to access on-chain information to find out about the address behind a domain but would be required to make off-chain requests to check on the validity of the DNSSEC entry.

**TLD Confusion** In two instances, we noticed confusion about the actual TLD of a domain. For example, Etherscan.io automatically appends the TLD `.eth` to any domain name, even if it is a domain name from the traditional DNS. This creates confusion, as

[9]See `https://app.ens.domains`, accessed 15th March 2023.

`example.org` and `example.org.eth` are two entirely separate domain names. Further, "Top-Level-Domains" are claimable below the `.eth` TLD and can be abused by malicious entities to trick people into sending funds to different addresses. We see that these domains can be held by separate entities that have respective control over the subdomains. ENS should clarify how their system works, protect TLD-related names such as `com.eth`, and instruct their integration partners to clear up issues with name resolutions.

**Conflicts in Name Resolution** ENS and DNS partly raise identical claims, as they both intend to be responsible for the name resolution of websites, similar to the `vitalik.eth`-example in subsection 2.3.2. This is not an issue as long as both systems are distinct, meaning that each system has its TLDs and does not interfere with the other. However, when ENS allows regular DNS TLDs existing in their system, two records to set a website exist: one in DNS as an `A`-record, and another in ENS as a `contenthash`-record. As of this writing, we cannot produce resolution errors, as, for example, the browser *Opera* properly resolves ENS domains to IPFS and other domains to the respective DNS-records, but also does not failover from DNS to ENS in case a record is not set in DNS. This has two effects: There are no conflicts in the name resolution of Opera, but the `contenthash`-record for DNSSEC-bridged domain names in ENS is useless, as it finds no consideration. We do not see any guidance in the ENS documentation on how to deal with potential conflict resolutions. If both systems become more and more merged, resolution errors will arise at the cost of security and usability.

### 6.2.5. Further Complexities

The user is the primary focus of the systems. One intent is to help users make conscious decisions about signing transactions and protecting their funds. While the user study displayed in (Ebel, 2021) and (Gallersdörfer et al., 2021a) found that an augmented MetaMask might protect users from address replacement attacks, it only did so in a partly adverse situation: The attacker did only change the address, invoking a security downgrade scenario in the browser plugin.

Suppose an attacker decides not only to replace the address but also to communicate to users to ignore any warning messages. In this case, the chance that users will ignore the warning in the browser plugin increases. Countermeasures and protections such as our system will likely lead to further countermeasures from attackers, fighting for the user's attention. It remains up to future work to better understand a) how users behave in increased adverse situations, b) how such a system will play out in the long run, and

c) whether it brings the intended security protections or provides a security layer that is hard for users to understand, resulting in diminished value in the long run.

Another issue comes with the growing ecosystem of blockchain networks. They are not only the respective blockchain networks but other technologies such as layer 2 networks, roll ups, and such arise. It remains an open question how these technologies can integrate our system and leverage the properties of TLS certificates in their respective contexts. Potentially, systems need to be redeployed on individual layers, resulting in ecosystem fragmentation and poor user experience. Some domain names might be available on one layer but not on the other.

Issuers that want to leverage the proposed system in their applications face several issues. First, they are impacted by the lack of bootstrapping for respective tooling, as outlined in subsection 6.2.3. Additionally, they face additional issues: Accessing the TLS certificate of their web server seems trivial but can be a complex endeavor. Smaller enterprises could be locked out of their servers, as they might only have obtained a web hosting package with included management; accessing the TLS certificate in this scenario, especially periodically, might not be economically possible. For larger enterprises, the problem might be similar: Given their complex structures and protections in place to ensure the integrity of their IT systems, periodically accessing TLS certificates might also be problematic.

## 6.3. Costs

Costs and transaction fees are a significant concern for users within blockchain networks. For example, in Ethereum, about 736,000,000 USD are spent on transaction fees alone in 2022[10]. Especially in high-demand situations, prices for transactions can skyrocket. For that, we analyze the influencing factors and costs of all three approaches. In subsection 6.3.1, we give an overview of the influencing factors relevant to any blockchain application. Then, we analyze the influencing factors of gas usage for all three approaches in subsection 6.3.2. We summarize our findings and give an overview of theoretical and actual costs in subsection 6.3.3.

[10]Own calculations based on (Blockchair, 2023).

### 6.3.1. Costs in EVM-based Blockchain Networks

In Ethereum and any EVM-based blockchain, three main components contribute to the price of a transaction. We display the formula in Equation 6.1. First, the computational intensity of a transaction is described in **usedGas**. All operations in the blockchain are priced in units of gas. The transaction sender has to pay for all computations, including a markup for the initial transaction of 21,000 units. A second factor is the incentivization of validators to include the transaction in their blocks, which comes as a **GasPrice**. The GasPrice determines the price the sender is willing to pay for a single gas unit priced in the respective blockchain currency (e.g., in Ethereum, Wei). For better readability, GasPrices are often given in *GWei* (GigaWei), that is GWei = Wei $*$ 1$e$9. Also, Ether = Wei $*$ 1$e$18. The GasPrice determines how fast a transaction is included in the block. The third factor is the **TokenValue**. It describes the value of the respective blockchain token in USD.

$$Tx_{Fee}[\text{USD}] = \text{UsedGas} * \text{GasPrice} * \text{TokenValue} \tag{6.1}$$

For a concrete example, Etherscan's Gas tracker tracks blockchain transactions and recommends different GasPrices for low, average, and high-speed transactions (Etherscan, 2023d). On the 2$^{\text{nd}}$ March 2023, at 08:00:47 UTC, the website recommended 20 Gwei for low, 21 for average, and 22 for high-speed transactions. For a regular transaction with average speed, we display the cost calculation in Equation 6.2, reaching about 0.73 USD for a single transaction. All three variables are prone to change, as the execution of code results in varying gasUsed. Ether prices and transaction throughput change as well.

$$Tx_{Fee}[\text{USD}] = 21,000 * 21 \text{ GWei} * 1,644.07 \text{ USD} = 0.725 \text{ USD} \tag{6.2}$$

Therefore, we initially compare all approaches using the variable *GasUsed*. In the second step, we plot the actual costs using the historical average daily gas prices and the Ether/USD exchange rate. We gather average gas prices and the value of ether since the inception of Ethereum. We obtain this information from Blockchair's full node block dumps, which contain the respective information (Blockchair, 2023).

### 6.3.2. Gas Cost Analysis

In all three approaches, separate computational steps contribute to the overall gas consumption of the respective approach. In general, three main actions within smart contracts consume gas.

- **1. Proof and Endorsement Verification:** The submission of proofs, signatures, certificates, RRSets, and endorsements, including their cryptographic verification, is computationally intensive and requires large amounts of gas.
- **2. Data Storage:** Storing any information, including cryptographic proofs, is also expensive, as it bloats the blockchain state.
- **3. Usage:** Lastly, using the information to retrieve addresses from domain names can contribute to gas costs.

Table 6.2 gives a preliminary indication of which operation applies to which approach.

| | Off-chain TLS | On-chain TLS | On-chain ENS |
|---|---|---|---|
| 1. Verification | ✗ | ✓ | ✓ |
| 2. Storage | ✓ | ✓ | ∼ |
| 3. Usage | ✗ | ∼ | ∼ |

**Table 6.2.:** Sources for gas consumption including applicability for each approach.

Given this preliminary indication, we decide to focus on the key elements of each approach:

- For the **off-chain TLS approach**, we analyze only the storage and deployment transaction of the smart contract, as this is the only interaction with the blockchain. The verification and usage of the endorsement, including its certificates, takes place off-chain and does not cause any gas costs. Therefore, we skip it in the individual part.
- The **on-chain TLS approach** puts heavy weight on verifying and storing leaf, intermediary, and root certificates, including the endorsement itself. Analyzing the storage and verification costs separately makes no sense, as one cannot exist without the other.

- The same applies for the **on-chain ENS approach**: The verification and storage of signed RRsets, including the proof for the address, are by far the most gas-consuming operation within the smart contract system. Both on-chain approaches use similar cryptographic algorithms, allowing for a more nuanced discussion. As we have access to real-world data, we can provide additional insights.

We do not focus on usage activities; accessing verified address storage elements is cheap within smart contract systems. Further, any optimization for address retrieval applies to both on-chain systems, as they can be implemented without changing other parts of the architecture (e.g., the concept of namehashes is interchangeable). Furthermore, comparing an optimized and live system to a proof of concept yields no additional insights.

In the following, we iterate over all three approaches and give an overview of expected costs. The unsuspecting reader might assume that determining the amount of gas in a transaction can be an exact science. However, many random factors (e.g., the character composition of the receivers' addresses) or circumstances can hardly be predicted (such as how many certificates have been submitted beforehand). For example, people brute force addresses for leading zeros, as it saves tiny sums of gas for a transaction, leading to significant savings in the long run[11]. Therefore, we indicate gas consumption for each case, and if justified, we analyze the costs in more detail.

### Off-chain TLS Approach

The costs of the off-chain approach have already been analyzed in (Gallersdörfer and Matthes, 2020, 2021a). We reiterate and revise the essential information.

Certificate information storage does not occur on the blockchain, as certificates such as leaf, intermediary, or root certificates are retrieved just in time for verification. However, the storage of endorsements requires gas. In contrast to the other two approaches, the endorsement's storage is decentralized, which means that with every deployment of a smart contract, the respective endorsement interface needs to be stored, incurring costs from the structure and storage allocation of the smart contract.

(Gallersdörfer and Matthes, 2021a) find that the deployment of a smart contract costs about 1.55 million gas, and a subsequent update on the signature (e.g., due to replacing an expired certificate with a new one) comes with costs associated with about 100,000

[11] See a short example here: `https://twitter.com/jconorgrogan/status/1623463549447335936`, accessed 3rd March 2023.

gas. Adding the contract to the registry costs an additional 124,000 gas, putting the entire initialization at 1.68 million gas. The main costs come from the smart contract creation; changing variable lengths or contents did not significantly change gas costs.

Overall, the code in (Gallersdörfer and Matthes, 2021b) is not optimized; indeed, room for improvement exists.

### On-chain TLS Approach

(Groschupp, 2020) and (Gallersdörfer et al., 2021c) give detailed insight into the costs associated with the on-chain TLS approach. We use key information and apply it in the context of this work.

Verification is a key element in the gas costs of the on-chain approach. In contrast to the off-chain approach, the smart contract is deployed only once, and we do not consider deployment costs, similar to users in ENS not paying for the main contract deployment. A fully functioning domain needs four elements: A working root certificate, an intermediate certificate, a leaf or domain certificate, and the endorsement. (Gallersdörfer et al., 2021c; Groschupp, 2020) found median gas costs of 1.1 million, 780,000, 790,000, and 580,000 for storing each element, respectively. To even the playing field, we assume that all certificates are issued in a single transaction, similar to ENS. Therefore, the 21,000 base gas for a transaction only has to be paid once. As the root certificates are the trust anchor, they are already deployed and covered by the deployers. That results in the total costs of $(780,000 - 21,000) + (790,000 - 21,000) + (580,000 - 21,000) + 21,000 = 2,108,000$.

In (Groschupp, 2020) and (Gallersdörfer et al., 2021c), the authors highlight two noteworthy insights: First, the choice of the algorithm influences the gas cost, whereas the prices for SHA-1 are higher than SHA-256. Second, the size of the signed data has a considerable influence on costs; the verification of a domain certificate with 225 alternative domain names resulted in costs of 4.5 million gas.

It is not surprising that different algorithms result in a deviating gas consumption. However, it is noteworthy that root certificates still use SHA-1: Despite SHA-1 being practically broken (Stevens et al., 2017), using SHA-1 in root certificates does not endanger the system's functionality, as they are not verified by browsers (TBS Internet, 2021). With this knowledge, the proposed approach in (Groschupp, 2020) and (Gallersdörfer and Matthes, 2021b) should refrain from verifying the root certificate's signature, saving a lot of gas in the process. We do not include root certificates in our calculation, regardless.

The storage costs are included in the verification process; it is hard to differentiate the costs of pure storage vs. other operations (e.g., cryptographic algorithms), but as the

results from the off-chain approach suggest, they are minor. Nonetheless, the on-chain TLS approach certainly requires the most data to be stored on-chain and thus resulting in the comparatively highest cost structure.

### ENS DNSSEC Approach

We leverage the dataset gathered in Section 5.5 to gain insights into the gas consumption of ENS DNSSEC approach. For an initial overview, we create a boxplot for each of the three functions an address can use to enter the contract. We display the plots in Figure 6.2.

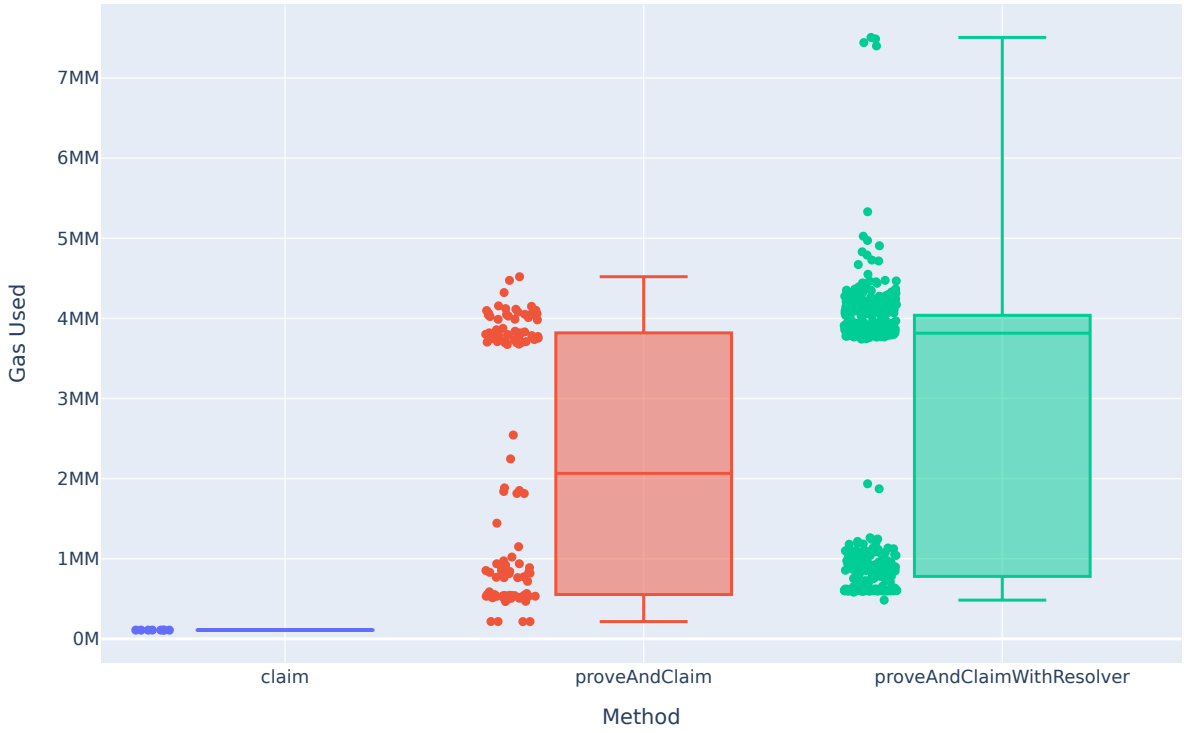



**Figure 6.2.:** Boxplots and individual points displaying the gas consumption for all three methods.

We make the following observations:

- The `claim`-function requires a very homogeneous amount of gas (108,000 to 109,000). As it only completes the last step of the verification, it requires previous proof submission, and only 16 domains are claimed that way, we do not further consider it.

- There appear to be several different clusters: For PAC and PACWR, one cheap and one expensive cluster exist. For PACWR, an additional more expensive cluster exists.
- PAC has a slight cost advantage, potentially the "WithResolver"-part is responsible for the marginally increased costs.

We further investigate the RRSets included as arguments in the respective transactions. We decode the RRSets and explore the respective counts and the algorithms used. We find that two algorithms are mainly in use, 8 (RSA-SHA256) and 13 (ECDSA-SHA256). Figure 6.3 provides an overview of the findings. Each dot represents a single domain, and we display the count of RRSets on the x-axis and the gas consumption on the y-axis. The color describes the number of RRSets that use algorithm 13. For example, the green dots with RRSet Count 3 all have in total 3 RRsets, two with algorithm 13 and one with algorithm 8.

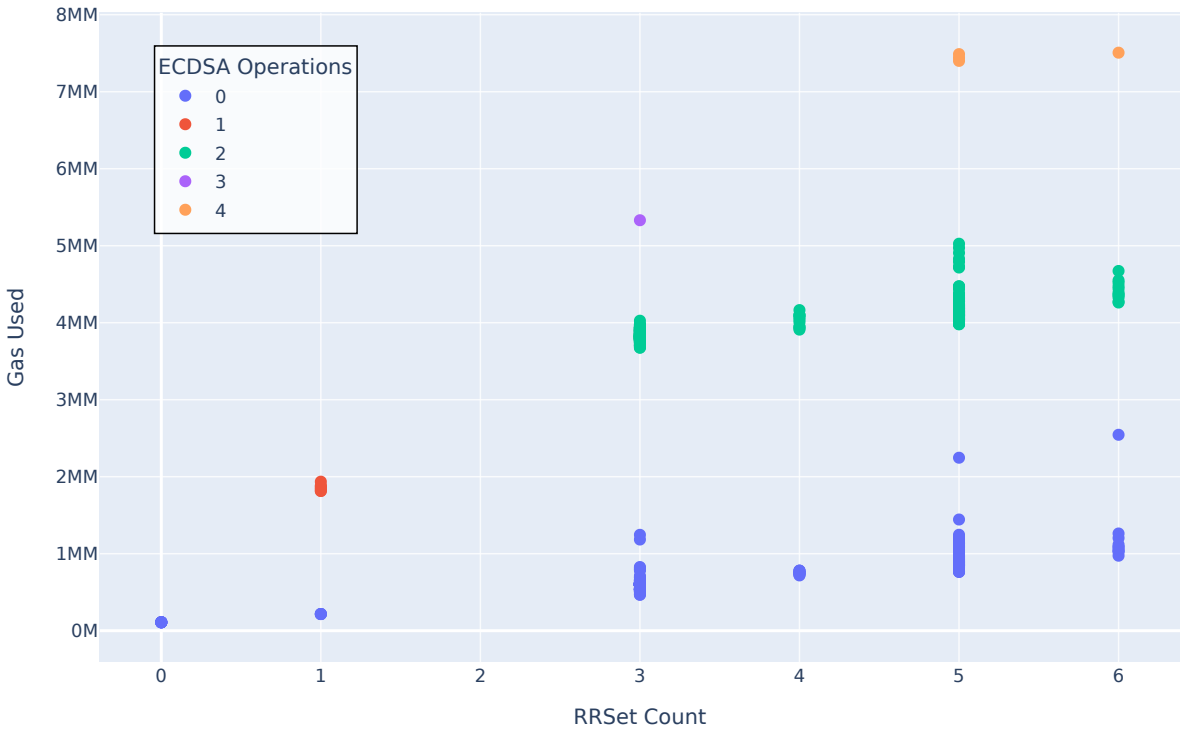



**Figure 6.3.:** Overview of the gas consumption of registrations compared to the number and type of the respective RRSets.

We find a clear correlation between leveraged cryptographic algorithms and the gas consumption of the respective transaction. RSA is much cheaper than ECDSA in terms

of gas costs. Thereby, we find that:

- On average, ab RSA-signed RRSet adds about 150,000 gas to the total costs of the transaction.
- On average, an ECDSA-signed RRSet adds about 1,500,000 gas to the total costs of the transaction, which is about ten times that of an RSA-signed RRSet.

We can confirm the initial thought from (Gallersdörfer et al., 2021c; Groschupp, 2020): The costs of cryptographic algorithms are the main determining factor in the overall picture.

For the overall comparison, we use the mean value of all successful transactions excluding transactions that call the `claim`-function and come up with a mean value of 2,595,415 gas spent.

### 6.3.3. Overview of Costs

We give an overview of all three approaches and their respective costs per issuance over the last three years, applying the gas price and token value of the Ethereum network. We use the mean value for the two on-chain approaches and the fixed value for the off-chain approach. Besides the overall DNSSEC approach value, we also introduce a fourth data point: an RSA-only DNSSEC point. The reason is that the on-chain TLS approach leverages only RSA, whereas the DNSSEC approach also leverages ECDSA. We limit ENS to the RSA algorithm only to enable a fair comparison. The mean value for all RSA-only issuances on the ENS is 724,466. We overview all four gas costs, including respective median, Q1, and Q3 USD costs in Table 6.3.

| | Off-chain TLS | On-chain TLS | ENS | ENS (RSA only) |
|---|---|---|---|---|
| Gas | 1,680,000 | 2,108,000 | 2,595,000 | 724,000 |
| Q1 [USD] | 65.34 | 81.99 | 100.92 | 28.16 |
| Med [USD] | 204.47 | 256.57 | 315.84 | 88.12 |
| Q3 [USD] | 501.68 | 629.48 | 774.91 | 216.20 |

**Table 6.3.:** Overview of gas costs for all four approaches.

Figure 6.4 provider an overview of the respective costs. Ethereum is a volatile network, and given the charts, prices can rise to 3,064.32 USD[12] while being 7.49 USD as the lowest. We develop the following insights:

[12] 3,000 USD is not much higher than the most expensive domain proof that took place on ENS. On 11th November 2021, the domain `game.com` was claimed for 2,605.56 USD.

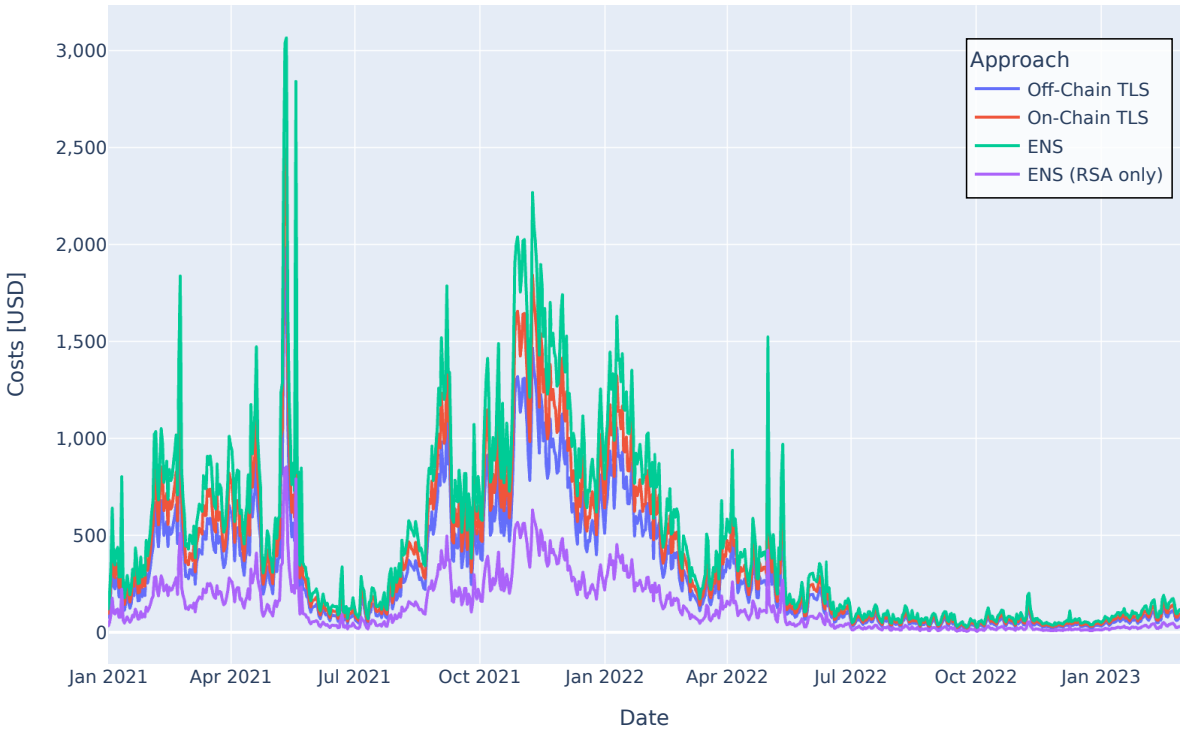



**Figure 6.4.:** Chart depicting the costs for each day from 1st January 2021 till 28th February 2023 in USD. We apply the median gas costs of each approach.

- Given the demand for such solutions, it becomes clear that neither approach is feasible for large-scale adoption. The median cost for registering a domain using the on-chain DNSSEC RSA-only approach still costs 88 USD. This is 8-fold the actual registration of a `.com`-domain[13].
- The usage of cryptographic algorithms matters. Unfortunately, users have no idea, and the application does not share why the user has to pay 3- to 4-fold more than other users.
- Timing matters: The more valid RRSets are included in the smart contract, the higher the chance that one has to submit fewer RRSets to the smart contract. A potential cost-enhancing way could be to batch these operations. We describe this idea in Section 7.3.
- The network should be urged to provide precompilations of contracts for cryptographic algorithms. These contracts could be much more cost effective than their Solidity-based counterparts.

[13]Compare to `https://www.domain.com/domains/domain-name-pricing`, accessed 4th March 2023.

## 6.4. Assessment of Security

The security of systems that rely largely on the correct functioning of underlying cryptographic systems and their algorithms warrants a dedicated discussion about potential security implications. We divide the following section into dedicated parts, namely:

- **Section 6.4.1**: All systems leverage cryptographic material and signatures from TLS or DNSSEC. It needs to be ensured that crafting these signatures does not harm the underlying systems.
- **Section 6.4.2**: The security of the system must be ensured. We discuss key elements for considering these approaches and shed light on blockchain-specific parameters.
- **Section 6.4.3**: Users, both entities issuing and using domains, need to be understood as primary sources of risk. We give an overview of the respective considerations.

Given the importance of security, much work has taken place. An initial assessment of the security of the off-chain TLS approach is done in (Gallersdörfer and Matthes, 2020, 2021a). The user-specific perspective of whether to consider warnings in wallets has been discussed extensively in (Ebel, 2021; Gallersdörfer et al., 2021a). In (Gallersdörfer et al., 2021c; Groschupp, 2020), security considerations of the on-chain TLS approach have been discussed extensively. Additional discussions and assessments have taken place in (Hoops, 2021) and (Käslin, 2020). Given this body of knowledge, we provide a generalized overview of security considerations and specifically cover elements of ENS, if warranted.

### 6.4.1. Interference with Underlying Systems

The interference with underlying systems can be discarded in the TLS-based approaches (Gallersdörfer et al., 2021c; Gallersdörfer and Matthes, 2020, 2021a; Groschupp, 2020). Generating signatures using cryptographic mechanisms and key material that is leveraged in another context would require adherence to semantic and syntactic specifications. These specifications are complex and do not align with any specifications we make in endorsements. In addition, the goals of the signatures are very different:

- In regular HTTPS communication, the signatures are very short-lived to authenticate the counterparty (often, only the server) and to exchange a symmetric key between the browser and the web server.

- In the on-chain or off-chain TLS approach, the signatures have a shelf-life identical to the validity of the certificate that was used for creating the signature; therefore, the signature only changes when a new TLS certificate is issued. In this case, the signatures are for authentication purposes only.

In addition to these two approaches, the ENS approach leverages DNSSEC. Here, the signatures and contents' format does not even deviate from its original intentions. `TXT`-records are intended to be used within DNS for plenty of reasons, so leveraging them, alongside respective signature information, is valid within a blockchain-specific context. A risk of interference with the underlying system can be ruled out.

### 6.4.2. Systems Security

Building secure systems on blockchain networks is inherently hard and has led to many hacks and billions in lost funds (Kushwaha et al., 2022). The reasons for this are manifold:

- Code deployed once is **immutable**. This means that smart contracts installed on the blockchain cannot be changed, and any error or bug introduced remains there forever. Patterns exist to allow "updating" smart contracts (see subsection 3.3.2 for details), but they come with their problems. Simply fixing a bug is not possible and requires complex routines.

- **Source code** is always **available**. Cryptocurrency communities consider it best practice to publish the source code to their deployed smart contracts so that people can look up the contract they intend to interact with. While this is certainly a good practice, the transparency allows for an extensive analysis (similar to the analysis in Section 5.5) from bad actors. In theory, regular people look into the code and spot errors before interacting with the code. In practice, they are hardly incentivized to do so. In contrast, malicious entities see 'inherent bug bounty programs' in smart contracts. They can withdraw all contract funds if they can exploit the smart contract.

- **Data is transparent** as well. While there are complexities in accessing all relevant data of a smart contract, it is still possible to log, trace, and debug all transactions that a smart contract has received. This information can subsequently be leveraged to find bugs or vulnerabilities in the system. Especially in systems that interact with outside entities, such as bridges, or that require proofs from external systems are prone to attacks (Duong, 2022).

- Smart contracts are **programmed in novel languages** such as Solidity. While these were designed to adhere to the developer and leverage concepts similar to famous languages, concepts are not understood or improperly applied. Additionally, these programming languages might face issues that are not apparent. For example, Solidity allowed for the over- or underflow of integers until version 0.8.0, released five years after the first release (Solidity Team, 2023). Certainly, many developers starting to work with Solidity were surprised by this feature.
- **New attack vectors** surfaced in blockchain systems. As smart contracts can call each other, novel concepts such as reentrancy attacks are enabled. Reentrancy leads to a situation in which a smart contract is partly called multiple times, and the execution of relevant parts is done multiple times (e.g., *"withdraw funds"*). In contrast, the check (*"user has sufficient balance"*) took place only once. This kind of attack led to the largest heist of Ether in 2016, when 3.6 million Ether was stolen, leading to the rollback of Ethereum and the consequent fork into Ethereum and Ethereum Classic (ConsenSys, 2023).

There is an entire industry focusing on auditing smart contract code. Often, pricing and timeline are contrary to the teams' goals. Additionally, extensive funding might be available only once an application becomes successful, often too late for a proper security audit. This can also be seen in the `DNSSECImpl`-code of ENS. During our smart contract analysis, we find several test variables and structures in live code, which are displayed in Listing 6.1. While they do not harm the system, it is questionable how much attention has been paid to auditing or verifying the correct functionality. To contextualize: the entire ENS network is ranked 126$^{th}$ with a market capitalization of 285 million USD on CoinMarketCap as of this writing[14].

```
53      event Test(uint t);
54      event Marker();
```

**Listing 6.1:** Actively used `DNSSECImpl`-code on Ethereum (Etherscan, 2023b).

### 6.4.3. User-Specific Risks

There are three user-specific risks that we discuss: *typo squatting* (also briefly discussed in (Gallersdörfer et al., 2021c; Groschupp, 2020)), *right to claim ownership* and *domain expiry*.

[14]Compare `https://coinmarketcap.com/currencies/ethereum-name-service/`, accessed on 4$^{th}$ March 2023.

### Right to Claim Ownership

A system should not affect entities that do not interfere with it. The systems we analyze in this manuscript should be designed so that the attack surface of uninvolved third parties is not enlarged. For the TLS-based approaches, this holds true. No one can leverage certificates or certificate signatures from subdomains to claim to be the main domain in the respective system; however, this is not true for the DNSSEC approach.

The specific question is: Does an entity own a domain if it owns the `_ens`-subdomain? We raised this question in Section 5.4 and revisit it here again.

Granting the domain in ENS to the entity that owns the subdomain does not seem to be an issue initially. However, one has to recognize that services that hand over subdomains exist primarily as a way to enable dynamic DNS. This includes setting any records for the respective host name.

Underscores in domain names are used in the wild. Using an underscore in the second-level domain name (e.g., `_example.org`) is prohibited, but it is not in host names. While we often refer to `_ens` as a subdomain, it is a host name of the domain. More specifically, RFC 8552 exists with the title *"Scoped Interpretation of DNS Resource Records through "Underscored" Naming of Attribute Leaves"* (Crocker, 2019). This RFC describes a best current practice: The host names of "well-known" services start with an underscore. It further *"defines the "Underscored and Globally Scoped DNS Node Names" registry with IANA"*. Its aims to highlight usage of "well-known" services and thus avoid collisions.

Two questions arise from this RFC:

- Does ENS adhere to this registry and has ENS registered `_ens` within this registry?
- Do providers of dynamic DNS services recognize the underscore as problematic for host names, specifically for `_ens`?

We can briefly answer the first question with *no*. IANA hosts the list, and while there are several entries, `_ens` is not one of them (IANA, 2023a). ENS should follow the procedure outlined in section 3 of RFC 8552 to register the usage.

The second question is more difficult to answer. Instead of conducting a field study to register the respective subdomain with as many providers as possible, we took a sample size of several providers. We found that none allowed us to claim the individual address because the underscore was considered illegal. One operator of such a service rejected the request with the notice that only the domain owner is allowed to claim host names starting with an underscore.

### Typo-Squatting

Typo-squatting is a known attack vector in which users are tricked into believing that they are on the intended website. In contrast, they are actually on a website controlled by an attacker. They are tricked into believing that the domain name is identical while exchanging single characters for similar looking, for example, an uppercase i (I) for a lowercase l (l). This security issue warrants an entire research field (Szurdi et al., 2014).

Given that only 877 domains exist in our dataset, we can list all domain combinations and their Levenshtein distance. We find four domain combinations with a distance of 1 and 31 with a distance of 2. The list of domain name combinations with a distance of 1 is displayed in Table 6.4.

| First Domain | Second Domain |
|---|---|
| z80.lol | **a**80.lol |
| chi.xyz | ch**a**i.xyz |
| aaronbomb.xyz | aaronbomb**s**.xyz |
| collective.xyz | **e**collective.xyz |

**Table 6.4.:** All domain combinations (Levenshtein distance of 1) in the ENS DNSSEC dataset. Changes in the second domain name are marked in bold. Data as of 27th February 2023.

Upon manual inspection, domain combinations with a distance of 2 appear irrelevant. For example, domain combinations such as `pwn.xyz`/`ibn.xyz` or `mty3.io`/`mex3.io` do not hint at typo-squatting attempts, as they are fairly easy to distinguish between. However, the existence of domains with a distance of 1, such as `collective.xyz` and `ecollective.xyz`, is interesting. In the following, we investigate them further:

- `a80.lol` and `z80.lol` were registered in October 2022 two days apart from each other. It appears that `z80.lol` was registered as a demo to showcase the DNSSEC feature of ENS[15]. Both websites are not accessible and their domain names are owned by different entities. A typo-squatting attack can be likely ruled out.

- `chi.xyz` belongs to a private person and was registered in December 2021, whereas `chai.xyz` was registered in April 2022. Little information can be found on the second domain, which appears to be some form of decentralized finance hub according to AwesomeNEAR[16] which seems to be offline for some time. A typo-squatting attack can be likely ruled out.

[15]See `https://twitter.com/0xz80/status/1584687805015687170`, accessed 15th April 2023.
[16]See `https://awesomenear.com/chai-xyz`, accessed 15th April 2023.

- `aaronbombs.xyz` and `aaronbomb.xyz` are likely not typo-squatting attempts. Both were registered at the same day in November 2022. Additionally, they are owned by the same Ethereum address.

- `collective.xyz` describes itself as *"the home of NFT communities"* and seems to have some relevance[17]. The first domain was registered in 2021, whereas the second domain was recently registered in January 2023 and transferred to ENS in February 2023. As of this writing, the second domain did not host any content, and a respective Twitter account does not seem to exist. With this information, we cannot decide whether a typo-squatting attack has taken place or will take place.

The analysis shows the difficulty to discern between typo-squatting attacks and similar names without a malicious attempt. Developing algorithms that compare and detect such attacks can be considered very challenging.

### Domain Expiry

Domain expiry can be an issue if the validity of a bridged domain is infinite. This is not an issue for the off-chain and on-chain TLS approaches, given that the verifier, in this case the registry, honors the validity of the underlying TLS certificate and does not further verify the authenticity of the respective address. In ENS, however, as the validity is indefinite, this can result in a problem from two perspectives:

- **Domain Expires but is Still Valid On-Chain**: While long-expired domains hint at a low or non-existent usage, the domain can still be used in a blockchain context, especially when no website is involved in the interaction.

- **Domain Expires and is Reclaimed Quickly**: Often, it happens that access to domains is lost for a short period, allowing anyone to claim the domain name (e.g., someone was able to register `google.com` for a minute in 2015 (CNET, 2015)). Even if only possessed for a brief time, it would allow the temporary owner to generate proof that lasts a lifetime on the blockchain. We find examples in which domains were re-registered with different sender addresses but cannot confirm or deny whether these are attacks or registrations with no connection.

[17]For example, `collective.xyz` has 29.3 k followers on Twitter, compare `https://twitter.com/collectivexyz`, accessed 4$^{th}$ March 2023.

In Section 5.5, we analyzed how many of the domains transferred to ENS were up for registration. Of the 62 domains that did not respond to HTTP requests and had no DNS records, 16 domains can be registered, as partly seen in Figure 6.5. For privacy and security reasons, we blur the domain list.

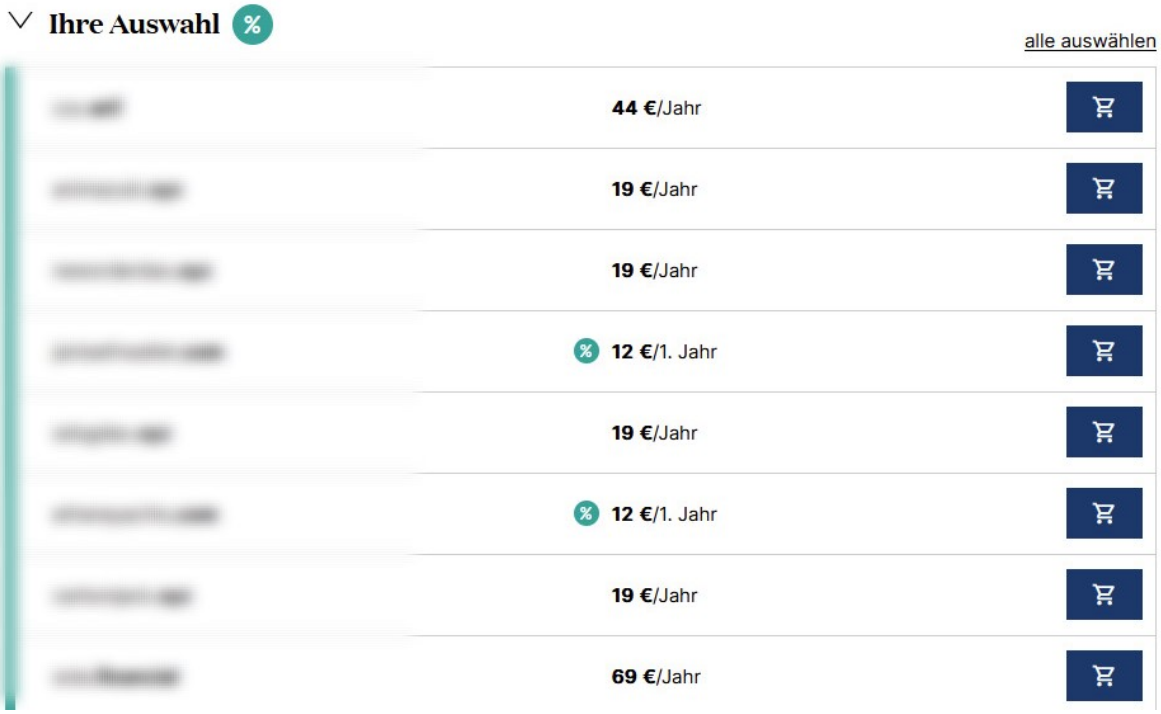



**Figure 6.5.:** Screenshot of `united-domains.de` of domains up for registration that have a valid registration in ENS. For security reasons, we blur the domain names.

Upon manual inspection, we find that the list partly includes personal websites, NFT projects, and DAOs. It is concerning to see that about 1.8% of all DNSSEC domains on ENS are free to register. Less than two years ago, the smart contract was created on 17$^{\text{th}}$ August 2021. We expect this number to rise, given that projects become irrelevant or abandoned. It needs to be decided case by case, whether the existence of such domains poses a security issue.

## 6.5. Requirements

In Section 3.2, we proposed functional and non-functional requirements to which the systems described in this thesis should adhere. At the end of this chapter, we discuss whether the requirements are fully fulfilled, partly fulfilled, or not fulfilled. We extend this discussion to ENS and its DNSSEC implementation, as it provides valuable insights

into the comparability of all three systems. We discuss functional requirements in subsection 6.5.1 and non-functional requirements in subsection 6.5.2. To conclude this section, we summarize our findings in subsection 6.5.3.

For simplicity, we refer to the approach outlined in Chapter 4 as the *off-chain approach*, the approach outlined in Section 5.2 as the *on-chain approach* whereas we refer to the approach outlined in Section 5.3 as the *ENS approach*.

### 6.5.1. Functional Requirements

> **FR 1: Issuance**. An entity should be able to issue a valid form of a certificate-like object to extend the scope of existing naming rights in a blockchain-based environment.

Both the off-chain approach and the on-chain approach can issue a certificate-like object to an address and define the respective rules and scope of this object, for example, whether the issuance of subdomains is allowed or how long the endorsement is valid. In ENS, however, the entity that provides the signed DNSSEC entry hands over all power to the respective address. The individual address can then decide on setting information in its registry, such as addresses for receiving funds, social media URLs, and more. These two approaches result in different designs: We propose keeping the right to decisions within the hands of the certificate owners, as they are the factual owners of the domain name. ENS hands the rights over to the address, which allows for cheaper updates to any information, as no signature needs to be verified again.

> **FR 2: Verification**. Any entity should be able to verify the authenticity of said object.

In all three approaches, off-chain entities can verify the authenticity of the endorsement or signature. Only the on-chain approach and the ENS approach allow for the verification of the endorsement or signature on-chain, which is the main reason for the two deviating approaches.

> **FR 3: Revocation**. An entity that previously issued such an object should be able to revoke its validity.

Revocation is currently only supported by the off-chain and on-chain approaches. In the off-chain approach to revoke an endorsement, the entire certificate needs to be revoked.

An updated endorsement containing a revoked flag can be published in the on-chain approach to rescind the endorsement. In the ENS approach, the owner needs to reassign the owner of the controller and delete any assigned information. Technically, there is no way to delete or revoke ownership.

> **FR 4: Active Usage**. The authenticity information issued within the envisioned system can be used for active authentication.

The *Active usage* requirement strongly correlates to the on-chain verification requirement. Within the off-chain approach, we cannot authenticate active or passive. Only the on-chain approach and the ENS approach allow the use of information in logically centralized smart contracts to decide whether a domain actively endorses an address.

> **FR 5: Auditability**. Any identity assertions issued within the system need to be traceable and auditable.

All three systems are auditable. Given that all three systems have some form of central registry in which endorsements must be either stored or linked, anyone can access these registries and verify their contents. We generate a dataset for the ENS DNSSEC registry in Section 5.5, which could also serve auditing purposes.

### 6.5.2. Non-Functional Requirements

> **NFR 1: Strong Form of Authentication**. Enable a strong form of authentication in the respective on-chain environment.

The on-chain approach and the ENS approach both provide a strong form of authentication on-chain, but the off-chain approach does not.

> **NFR 2: Usage of Human-Readable Names**. The names in the system should be easily recognizable by humans.

All three systems rely on domain names as a means and therefore provide human-readable domains. However, from a UX perspective, domain integration in ENS is not perfect. While in their application, names are correctly displayed as `example.org`, other applications that support ENS display them as `example.org.eth`[18].

[18] For example, `nft.com`, which is bridged to ENS, resolves in their app correctly to `nft.com`, whereas Etherscan resolves it to `nft.com.eth`. See `https://app.ens.domains/name/nft.com/details` and `https://etherscan.io/enslookup-search?search=nft.com`, both accessed on 28th February 2023.

**NFR 3: No Requirement for Bootstrapping**. The system should rely on a well-established system for name management and authentication.

While all three systems rely on well-established systems for name management, adoption within the software, wallets, and more has not happened for the off-chain and on-chain approaches. In contrast, ENS is widely adopted in many systems and block explorers.

**NFR 4: Blockchain Agnostic**. The system and its core concepts should be applicable in any blockchain network.

All systems are blockchain-agnostic. The ENS approach could also be leveraged in any other chain. Still, given its close collaboration and association with Ethereum, it might be harder to migrate to other chains and gain adoption.

**NFR 5: Decentralization**. Avoid centralization beyond already-existing naming services.

All systems are somewhat decentralized and rely on a third party. The management of ENS and the ENS system relies on the ENS DAO to ensure further development. The off-chain and on-chain approaches would face similar dependencies to ensure continuous growth.

**NFR 6: Openness**. Allow anyone to participate within the system; thus, do not rely on intermediaries that could limit access to the system.

All systems allow any entity to participate in their systems. The only requirement to participate is domain ownership, which comes with additional costs.

**NFR 7: Non-Repudiation**. The extension of the scope of an authentic certificate to a blockchain address cannot be denied after the fact.

Given the immutability of blockchain networks, all systems support non-repudiation, as any data submitted to the blockchain can be retrieved later.

**NFR 8: Robustness against Denial of Service Attacks**. The system is robust against DoS attacks.

No approach introduces additional centralized systems that could be subject to a DoS attack.

> **NFR 9: PKI Agnostic and X.509 Support**. The system relies on the X.509 certificate standard and supports any PKI adhering to that standard.

The on-chain and off-chain approaches are PKI agnostic and support X.509 certificates. The ENS approach supports only DNSSEC and is incompatible with any other form of PKI or certificate.

> **NFR 10: Adherence to Certificate Status**. The status and validity of the underlying certificates apply to statements and assertions managed within the system.

The off-chain approach and on-chain approach honor the certificate's status. However, the on-chain approach cannot directly retrieve the certificate's status, but it needs to be provided on-chain. The off-chain approach can directly access the status of the certificate. ENS ignores any change to the DNSSEC information and does not allow any proof that the entry was removed.

### 6.5.3. Overview

In Table 6.5, we give an overview of the fulfillment of the requirements of the three systems.

## 6.6. Summary

The evaluation of all three systems in varying dimensions shows that specific trade-offs were made to leverage the underlying infrastructures of the WWW in a blockchain context and achieve the use-case-specific goals. When using and implementing any approach, developers and system engineers must be aware of the characteristics of these systems as well as their advantages and drawbacks, in particular:

- **Suitability of DNS / TLS / X.509 Certificate and DNSSEC Ecosystem**: TLS has been fully adopted by the WWW, and there is no major website that does not support TLS inherently. Even if a website does not support TLS from the start, the owner can easily obtain the respective certificates at no cost. Systems such as Let's Encrypt have paved the way for a fully encrypted WWW. There are no reasons why TLS cannot be leveraged as a base system for the outlined use cases. Almost the same can be said about DNSSEC. Its adoption is not as widespread as

| | Off-chain approach | On-chain approach | ENS approach |
|---|---|---|---|
| **FR 1** | ✓ | ✓ | ✓ |
| **FR 2** | ∼ | ✓ | ✓ |
| **FR 3** | ✓ | ✓ | ∼ |
| **FR 4** | ✗ | ✓ | ✓ |
| **FR 5** | ✓ | ✓ | ✓ |
| **NFR 1** | ✗ | ✓ | ✓ |
| **NFR 2** | ✓ | ✓ | ∼ |
| **NFR 3** | ∼ | ∼ | ✓ |
| **NFR 4** | ✓ | ✓ | ∼ |
| **NFR 5** | ∼ | ∼ | ∼ |
| **NFR 6** | ✓ | ✓ | ✓ |
| **NFR 7** | ✓ | ✓ | ✓ |
| **NFR 8** | ✓ | ✓ | ✓ |
| **NFR 9** | ✓ | ✓ | ✗ |
| **NFR 10** | ✓ | ∼ | ✗ |

**Table 6.5.:** Fulfillment of requirements of the three systems described in this thesis. (Fully fulfilled: ✓/ partly fulfilled: ∼ / not fulfilled: ✗.)

that of TLS. However, from a practical perspective, it can still be considered fully functional for the intended use cases. Only 1.7% of domains exist under TLDs that do not support DNSSEC, the most notable being `.ga`. It can be expected that most entities intending to leverage DNSSEC for similar applications should not face any issues.

- **Requirement for Bootstrap**: The broad availability and adoption of TLS and DNSSEC allow almost anyone to link domain names to addresses quickly, mitigating the requirement for a complex bootstrapping process of creating and assigning a naming scheme for anyone to adhere to. Regardless, a bootstrapping process is still required, as wallets, monitoring, and blockchain-specific software need to be augmented with the respective functionality. Users and vendors also need to adopt the system. Just because a naming scheme is available, the need for bootstrapping is not absent. ENS has already been implemented in many wallets for the Web3 community. Nonetheless, the number of adoptions shows that ENS users mostly focus on claiming new `.eth`-names, not on bridging their regular domains. Why adoption is still lacking remains a key question.

- **Costs of deployment and operation**: If these systems are deployed in a public permissionless setting, the respective sender must pay for the resulting gas usage.

The average costs for all approaches from 2021 to March 2023 lie between 88 and 316 USD. Signature algorithms are a key contributor to gas costs, resulting in high fees for the entities transferring domains. As ENS leverages ECDSA, prices are slightly higher. Nonetheless, private or public permissioned systems do not require monetary payment. Thus, these systems can be leveraged more easily within these networks. Potential ideas for moving parts of the deployment and verification process to second-layer networks, further reducing the costs, are outlined in Section 7.3 *Future Work*.

- **Security**: The proposed systems rely on the security of the underlying systems and therefore inherit well-known issues such as typo-squatting. However, the complexity and additional signature material that needs to be managed do not contribute to the attack surface of the underlying systems. ENS should inform ICANN that they are using subdomains to prove ownership of the to-be-claimed domain; respective protections can then be set in place. Any approach can contribute to the security of the underlying system by spreading awareness of its functionality, potential downsides, and countermeasures.

- **Fulfillment of Requirements:** No system satisfies all requirements. While the off-chain approach suffers from missing usability, the on-chain approach faces high costs, whereas ENS misses crucial features, such as the revocation of once-endorsed Ethereum addresses. The individual application, its requirements, and its users must be considered when deciding which approach suits best.

In the next chapter, we summarize the findings, answer the research questions, and conclude with an outlook for future work.

# Chapter 7.

# Conclusion and Future Work

## 7.1. Conclusion

Identifying and authenticating counterparties in blockchain networks is essential to properly engage in commercial activities for businesses and end users. Naming services try to solve the existing issues in blockchain networks, but still face issues of missing adoption and high bootstrapping efforts.

In this dissertation, we proposed a framework for leveraging the TLS PKI or any other form of X.509 PKI and its contents in the form of certificates and endorsements to assert a relationship between the properties of the certificates (in case of TLS, domain names) and on-chain addresses.

While other systems exist that partly rely on PKI to leverage certificate attributes, this dissertation closes the gap in understanding potential ways to leverage the certificates alongside their signatures in both an off-chain and on-chain verification context, resulting in different feature sets, security assumptions, and execution costs, targeting different problem statements and use cases.

Furthermore, we investigated a system that leverages DNSSEC with the same goal of enabling the usage of domain names in an on-chain context. We analyzed the infrastructure, gathered data, developed a set of key differences in system designs, and analyzed costs and usage.

This dissertation shows that using TLS and DNSSEC in an on-chain context is feasible and can be integrated with new and preexisting smart contracts. Verifying endorsements in an off-chain context is limited by browser support. The hierarchical structure of PKIs and DNSSEC allows for the one-time verification of intermediary certificates and keys, reducing the cost structure for all approaches. Using TLS certificates or DNSSEC partly minimizes the need for bootstrapping a novel identity management system.

Nonetheless, trade-offs exist for all described systems. While the need to bootstrap a naming scheme is eliminated, bootstrapping from a software-centric perspective is still required. Website owners, wallets, and users must adopt a software stack that allows the issuance, verification, and revocation of such endorsements, including all technical complexities, such as accessing certificate information, setting DNS records, or generating respective signatures. For the TLS-based approaches, the software for the automatic renewal and update of endorsements is still missing. A "chicken-or-egg" problem still exists: Website owners will not set up an enhanced security protocol if no user can profit from it. The same applies to the other side, as users will not benefit from it. While there is no need to bootstrap an identity management and naming system from the ground up, these systems still require bootstrapping efforts from users and companies.

From an ideological perspective, all described systems undermine the purpose of the respective blockchain network, using a decentralized network while introducing centralized entities that control the naming or identity management system. Nonetheless, these systems are optional, and no one is forced to use them. Further, the systems using X.509 certificates are generalizable, so any PKI can be leveraged if valid use cases exist.

## 7.2. Answer to Research Questions

In the following, we answer the research questions initially posed in Chapter 1. While the initial questions are *italicized*, the answers are written in regular font styling.

### RQ1: To what extent can one leverage domain names and related Public Key Infrastructures in a blockchain environment?

**RQ 1.1** *In which ways have domain names and related PKIs been utilized in a blockchain context?*

We identified that a form of PKI, namely DNSSEC, is actively leveraged in the ENS ecosystem, allowing the usage of respective domain names in the ENS ecosystem. Further, other systems, such as TLSNotary and Town Crier, leverage signature information of TLS certificates to prove on a blockchain that a website once displayed respective information.

**RQ 1.2** *What are the technical pathways to enable domain name usage in a blockchain context?*

We identify two technical pathways to utilize PKIs in a blockchain context; if

verification occurs locally on the users' device, only the endorsement must be stored alongside the smart contract. If someone wants to verify the endorsement in an on-chain context, all respective certificate information, including its path, must be available on-chain. We refer to these approaches as off-chain verification and on-chain verification, respectively.

## RQ2: What are ways to verify the connection between a domain name and a blockchain address in a local context?

**RQ 2.1** *How can cryptographically verifiable assertions between blockchain addresses and domain names be made?*

We leverage the cryptographic key material in the form of the private key of the respective certificate of a domain to create a signature over data that uniquely describes the relationship of the certificate and its properties to the address. We refer to this signature, including the data, as an endorsement.

**RQ 2.2** *How does the life cycle of cryptographic key material influence the system's functionality?*

The life cycle and validity of the cryptographic key material significantly affect the functionality of the proposed system. Invalid or expired certificates will impact the functionality of the respectively generated endorsements, requiring a re-creation of any endorsements.

**RQ 2.3** *What are the limitations of verifying assertions in a local context?*

Off-chain verification limits the functionality to preventing or detecting address replacement attacks and data authentication. Additionally, certificate retrieval and verification are issues in browser-based add-ons, drastically limiting usage.

## RQ3: How can we verify the connection between a domain name and a blockchain address in an on-chain context?

**RQ 3.1** *What are the technical pathways for enabling the usage of certificates and endorsements in an on-chain context?*

We identify two possible ways of verifying certificates and endorsements on-chain; with the support of oracle networks, we can compute the results of verifications off-chain and bridge them on-chain for decision-making. Further, it

is possible to predefine the root certificates on-chain and verify the contents of the certificates and endorsements using the root certificates as a trust anchor. From a decentralization standpoint, the second option is more desirable.

**RQ 3.2** *How does the life cycle of cryptographic key material influence the functionality of the proposed system?*
The life cycle of the cryptographic key material significantly influences the proposed system. In contrast to the off-chain verification approach, blockchain networks cannot access revocation information directly. Therefore, proper mechanisms must be developed to incentivize entities to publish this information on-chain and invalidate respective certificates and endorsements.

**RQ 3.3** *What are the limitations of verifying assertions in an on-chain context?*
While on-chain verification allows for elaborate use cases and integration in access control schemes, the usage and verification of assertions on-chain come with a significant cost increase in on-chain fees. Furthermore, no direct access to revocation information might delay the invalidation of respective on-chain endorsements.

**RQ 3.4** *How can the interactions of users of Ethereum Name Service (ENS) regarding DNS Security Extensions (DNSSEC) be analyzed?*
All domain names that users bridge to ENS interact with a single smart contract, exposing three distinct functions for submitting and proving respective information. We can analyze these transactions and extract relevant information, such as domain name, RRSets, and additional metadata such as creation time and transaction fees.

**RQ 3.5** *How do existing approaches for leveraging domain names in a blockchain context compare to each other?*
Several key differences exist between the approaches. While the differences in their leveraged systems, such as TLS and DNSSEC, are obvious, design decisions regarding the proof of ownership, key material longevity, and costs became apparent.

## 7.3. Future Work

Leveraging PKI systems in a blockchain-specific context requires additional research to fully understand the complexities and enable efficient ways to issue and verify endorsements for blockchain addresses.

We identify the following areas for further research:

- **Expanding Scope:**
  - *Application in Alternative Contexts:* The feature set resulting from any of these systems allows the exploration of novel use cases that partly rely on information from the DNS. For example, bootstrapping communities or entities that belong to a specific group becomes possible, as outlined in (Strugala, 2020).
  - *Decentralized Identifiers*: Decentralized Identifiers (DIDs) are a novel form of identity mechanism proposed by the W3C (Sporny et al., 2022). Knowledge gathered by leveraging PKI information in a blockchain context could be helpful to develop DID methods that do not rely on centralized parties beyond existing naming schemes, such as the `DID:TLS` method proposed in (Käslin, 2020).
  - *Investigating Alternative Base Layers*: Other technologies exist that protect the integrity of naming schemes but that are not PKIs, such as CT. In particular, CT resembles, from a technological perspective, a blockchain-like approach, creating a form of audit log. The root hash is singed and all certificates are part of it. Relying on CT, the inclusion and verification of certificates on-chain could be much cheaper than alternative approaches. Furthermore, ENS opted for DNSSEC as their primary building block for bridging domain names to their systems. Understanding the implications of leveraging alternative methods such as TLS or CT could yield enhancements to their systems.
  - *Investigating Layer 2 Usage*: Blockchain networks consist not only of the respective base layer but also integrate different technologies such as layer 2 networks, rollups, or other scaling solutions. One needs to understand how these naming systems can be used in these novel scaling solutions or if a redeployment for the respective layer is required.
- **Enhancing Cost Structures:**

  - *Precompilation of Cryptographic Algorithms*: Cryptographic algorithms are omnipresent in blockchains. However, only a limited set of these is implemented in any blockchain. Some cryptographic algorithms are already provided in a pre-compiled form (Wood et al., 2014), avoiding the overhead of the EVM language. However, the list of such contracts should be extended to fit more use cases, including verifying TLS and DNSSEC proofs on-chain.
  - *Batch Issuance of RRSets in ENS*: One could trustless batch multiple RRSets and domains in a separate smart contract. Users could set up their DNSSEC correctly and submit an intention to claim their domain in ENS to this separate smart contract, including some funds to cover the costs. When enough entities joined the pool, anyone could batch-submit all RRSets and prove their respective inclusion to the ENS DNSSEC oracle. As DNSSEC "automatically renews" the signatures, a third party can use the proofs without further interaction from the actual domain owner.
  - *Leveraging Secure Off-Chain Data Retrieval*: ERC-3668 specifies a way to enable regular wallets to supplement a standard transaction with information from an off-chain source that, in turn, is verified by the on-chain smart contract (Johnson, 2020). This could help to circumvent directly issuing the domain on-chain but using information just in time. Layer 2 networks in particular could benefit from this approach, as an issuance on the respective network is no longer required. However, further work is necessary to understand the implications of ERC-3668 on the usability and security of individual systems.

- **Understanding Human Behavior:**
  - *The Influence of Warning Messages on Users*: Generally, warning messages in browsers and other software and their influence on user behavior are poorly understood. The same applies to the software described before; for example, it is unclear how users perform if they are informed of the security warning beforehand and instructed to ignore it. In addition, as any domain can endorse any smart contract, it remains unclear whether an assertion can have a negative effect on an address (e.g., if from a domain associated with illicit content).
  - *Enhancing Protection against Typosqatting*: Typosquatting remains an issue with no apparent solution, existing in novel systems described in this thesis and

in regular WWW environments such as browsers. It needs to be understood which information is helping the user to decide whether to trust a website or how systems can be developed that better find and mark such typosquatting attempts, as outlined in (Hoops, 2021).

  - *Adoption of Domain Names in ENS*: While ENS is reasonably large and has issued millions of domain names in their `.eth` TLD, adoption of common domain names remains very low, with below 1,000 domains bridged as of this writing. Understanding the reasons for low adoption is essential to help design better systems. Potentially, the high costs and complexity of issuance can deter users from doing so, but the solution is potentially not well-known within the community.

- **Abandoning Existing Systems**:
  While TLS or DNSSEC serve as the backbone of the respective systems, one should explore ways to cut ties to the underlying system after a specific time and bootstrapping, potentially changing the identity management system from a hierarchical structure to a web of trust structure. ENS has introduced *NameWrappers* (Ethereum Name Service, 2023d) which allow the domain owner to abandon the control of a subdomain. Still, it remains unclear which long-term effects such approaches have when viewpoints from the WWW deviate from on-chain viewpoints. Nonetheless, these approaches could also serve as a basis for bootstrapping an entirely different DNS.

# Appendix

# Appendix A.

# Prior Publications and Student Work in the Context of this Thesis

In the following, we display prior publications by the author of this dissertation, in detail the title, authors, conference or proceedings, and year of publication.

- Ulrich Gallersdörfer and Florian Matthes (2020). AuthSC: mind the gap between web and smart contracts. *arXiv preprint arXiv:2004.14033*. URL: `http://arxiv.org/abs/2004.14033`
- Ulrich Gallersdörfer and Florian Matthes (2021a). TeSC: TLS/SSL-Certificate Endorsed Smart Contracts. *The 3rd IEEE International Conference on Decentralized Applications and Infrastructures*
- Ulrich Gallersdörfer, Friederike Groschupp, and Florian Matthes (2021c). Mirroring Public Key Infrastructures to Blockchains for On-chain Authentication. *5th Workshop on Trusted Smart Contracts In Association with Financial Cryptography 2021*
- Ulrich Gallersdörfer, Jonas Ebel, and Florian Matthes (2021a). Augmenting MetaMask to support TLS-endorsed Smart Contracts. *5th International Workshop on Cryptocurrencies and Blockchain Technology*

In addition, following student work in the form of theses and semester projects was executed under the technical and scientific advisory of the author. We highlight manuscripts relevant to this dissertation. They are separately referenced in the bibliography and relevant sections. Thank you to all the students for supporting this research project.

- Friederike Groschupp (2020). Exploring the Use of SSL/TLS Certificates for Identity Assertion and Verification in Ethereum. *Master's Thesis*. Technical University of Munich. **(Groschupp, 2020)**
- Kilian Käslin (2020). Establishment of a Minimum Viable Self-Sovereign Identity Network. *Master's Thesis*. Technical University of Munich. **(Käslin, 2020)**
- Jan-Niklas Strugala (2020). Leveraging TLS/SSL-based Identity Assertion and Verification Systems for On-chain Authentication and Authorization of Real-world Entities. *Master's Thesis*. Technical University of Munich. **(Strugala, 2020)**
- Jonas Ebel (2021). Augmenting the MetaMask-Wallet with Domain Name based Authentication of Ethereum Accounts. *Master's Thesis*. Technical University of Munich. **(Ebel, 2021)**
- Jan Felix Hoops (2021). Threat Analysis, Evaluation, and Mitigation for Smart Contracts Endorsed by TLS/SSL Certificates. *Master's Thesis*. Technical University of Munich. **(Hoops, 2021)**
- Pascal Herrmann, Tuan Anh Ma, Metodi Manov, David Stübing (Winter Term 2020/2021). TLS-based Authentication Management on Blockchain. *Project Work in Software-Engineering for Business Applications - Lab (IN2106, 2129, 4077)*. Technical University of Munich. **(Herrmann et al., 2021)**

# Appendix B.

# Endorsement Flags

In Chapter 3, we briefly introduce the concept of flags as a way to further configure an endorsement. This appendix gives an overview on all previously defined flags with their definition.

The following flags have been initially proposed in (Gallersdörfer and Matthes, 2021b):

> "Flags enable additional functionality or restrictions in handling an endorsement. We display a list of all available flags and reasoning. In the smart contracts, we store flags in a `bytes24` variable. This allows us to store up to 192 flags, addressing them from $f_1$ to $f_{191}$. Each flag can be set either to true or false, resulting in $f_i = \{0, 1\}$.
>
> - $f_0$ **SANITY**: The sanity flag is always set to 1 to check if the flag variable is uninitialized or if all flags are actually set intentionally to 0.
> - $f_1$ **DOMAIN_HASHED**: This flag is set if a domain is stored as a hash for privacy reasons. The hash is constructed as $h = hash(domain)$. This flag is set if an owner does not want the smart contract to be easily attributed to the domain by crawling the blockchain. We rely on `keccak256` as hash function.
> - $f_2$ **ALLOW_SUBENDORSEMENT**: This flag is set if a smart contract is able to endorse further addresses such as contracts or externally owned accounts. The referenced smart contracts are stored in an array of the respective endorsing smart contract. If this flag is not set, the verification of the subendorsements fails.
> - $f_3$ **EXCLUSIVE**: If this flag is set, one contract equipped with a valid endorsement can exist; if multiples exist, no contract is considered to be

valid as long as the owner resolves the issues by either invalidating the endorsements, or removing the EXCLUSIVE flag from the contracts.

- $f_4$ **PAYABLE**: If a domain owner wants to allow users to send funds to the domain (owner), the owner sets this flag to let users know that this contract accepts funds.
- $f_5$ **ALLOW_SUBDOMAIN**: If this flag is set, smart contract addresses that are displayed in a subdomain context (the smart contract only being endorsed by the regular domain) can be verified. This is similar to a wildcard in TLS certificates and requires the certificate being issued for the respective domain.
- $f_6$ **TRUST_AFTER_EXPIRY**: Data that has been entered while the endorsement was valid can still be considered as valid after the endorsement or the TLS certificate expires. Because this information is time-stamped and no one can add or modify the data (without being noticed), the data is considered to be valid if this flag is set. This flag is especially useful for cases in which the blockchain is used to store data for public verification.
- $f_7$ **STRICT**: If this flag is set, the certificate returned via the web server must be identical to the certificate that signed the endorsement; otherwise, the verification fails.
- $f_x$ **reserved**: All other flags are reserved.

" – *TeSC: TLS/SSL-Certificate Endorsed Smart Contracts – Supplementary Material* (Gallersdörfer and Matthes, 2021b).

# Appendix C.

# Browser Warning Pages

In Chapter 4, we analyze TLS-related warning pages for application within our approaches. This appendix contains the details of the warning pages for all browsers in question.

Table C.1 gives an overview of all warning-related information that is displayed to the user when visiting a) a certificate-protected web page (case *"Secure"*), b) an extended validation certificate protected web page (case *"Secure EV"*), c) a web page with a minor certificate error (e.g., expired certificate) (case *"Minor"*), d) a web page with a major error (e.g., malfunctioning certificate pinning) (case *"Major"*), and e) an unprotected web page over HTTP (case *"HTTP"*). Unreachable states are marked with a dash (-).

We include the following metrics and observe respective characteristics:

- **URL**: How the URL's text is displayed to the user. Either the protocol (*HTTPS*) is shown, omitted, or crossed out.
- **Lock-Symbol**: The way the lock symbol is represented in the URL bar. We observe black-colored locks, crossed-out locks, absent locks, or additional warning messages "not secure" instead of the lock.
- **Cert Info**: Information displayed about the certificate after clicking the lock symbol in the URL bar. We observe information about the validity, issuer, receiver, or information about the certificate error.
- **Warning**: Contents of the warning page in case of any certificate error, such as error details and options such as "reload", "go back", or "close website".
- **Advanced**: Contents of the advanced section after clicking "Advanced" on the warning page in case of a minor or major certificate error. We receive further detailed information about the error and potential options such as "Continue" or "go back".

| | | **Chrome** (110) | **Edge** (109) | **Firefox** (109) | **Opera** (95) | **Safari** (16) | **FF Android** (109) | **Safari iOS** (iOS 16.1.2) |
|---|---|---|---|---|---|---|---|---|
| **Secure** | *URL* | No HTTPS | HTTPS | HTTPS | No HTTPS | No HTTPS | No HTTPS | No HTTPS |
| | *Lock-Symbol* | Black | Black | Black | Black | Black | Black | Black |
| | *Cert Info* | Valid | Valid | Valid, Issuer | Valid | Valid | Valid, Issuer | - |
| | *Warning* | - | - | - | - | - | - | - |
| | *Advanced* | - | - | - | - | - | - | - |
| **Secure EV** | *URL* | No HTTPS | HTTPS | HTTPS | No HTTPS | No HTTPS | No HTTPS | No HTTPS |
| | *Lock-Symbol* | Black | Black | Black | Black | Black | Black | Black |
| | *Cert Info* | Valid, Issued to | Valid, Issued to | Valid, Issued to, Issuer | Valid, Issued to | Valid, Issued to, Issuer | Valid, Issuer | - |
| | *Warming* | - | - | - | - | - | - | - |
| | *Advanced* | - | - | - | - | - | - | - |
| **Minor** | *URL* | HTTPS crossed out | HTTPS crossed out | HTTPS | No HTTPS | No HTTPS | No HTTPS | No HTTPS |
| | *Lock-Symbol* | "Not secure" | "Not secure" | "Not secure" | "Not secure" | None | Crossed out | None |
| | *Cert Info* | Warning | Warning | Warning | Warning | - | Warning | - |
| | *Warning* | Warning, "Go Back" | Warning, "Go Back" | Warning, "Go Back" | Warning, "Go Back" | Warning, "Close" | Warning, "Try again" | Warning, "Go Back" |
| | *Advanced* | "Continue" | "Continue" | "Continue" | "Continue" | "Continue" | "Continue" | "Continue" |
| **Major** | *URL* | HTTPS crossed out | HTTPS crossed out | HTTPS | No HTTPS | No HTTPS | No HTTPS | No HTTPS |
| | *Lock-Symbol* | "Not secure" | "Not secure" | "Not secure" | "Not secure" | None | Crossed out | None |
| | *Cert Info* | Warning | Warning | Warning | Warning | - | Warning | - |
| | *Warning* | Warning, "Reload" | Warning, "Reload" | Warning, "Go Back" | Warning, "Reload" | Error | Warning, "Reload" | Error |
| | *Advanced* | Information | Information | Information, "Go Back" | Information | - | Information, "Go Back" | - |
| **HTTP** | *URL* | No HTTPS | No HTTPS | No HTTPS | No HTTPS | No HTTPS | No HTTPS | No HTTPS |
| | *Lock-Symbol* | "Not secure" | "Not secure" | Crossed out | "Not secure" | "Not secure" | Crossed out | "Not secure" |
| | *Cert Info* | Warning | Warning | Warning | Warning | Warning | Warning | - |
| | *Warning* | - | - | - | - | - | - | - |
| | *Advanced* | - | - | - | - | - | - | - |

**Table C.1.:** Overview of browsers and their behavior in five different security scenarios.

# Appendix D.

# Related Ethereum Addresses

In the following, we list all addresses and transaction hashes we found and used throughout this dissertation.

### Main Contracts

- `0x00000000000C2E074eC69A0dFb2997BA6C7d2e1e` **(ENS Registry)**
- `0xaB528d626EC275E3faD363fF1393A41F581c5897` **(ENS Root)**
- `0x58774Bb8acD458A640aF0B88238369A167546ef2` **(DNSRegistrar)**
- `0x21745FF62108968fBf5aB1E07961CC0FCBeB2364` **(DNSSECImpl)**
- `0x765653c78f609826DfD091F9208Aeb610949A28F` **(RSA/SHA1)**
- `0xB83A8AC6900f19d77333e640550Bd8830d5fcb26` **(RSA/SHA256)**
- `0xe571A50F76ff7404F3Ce380D06CBd2c9c6Ca3670` **(ECDSA/SHA256)**

### Transaction Hashes

**limo.eth registration:**
`0xfe0f462e55f552c7ddd63be5b1c9c7e64c4addcc50f26f99a391fb236948b3f2`

**nish.com registration:**
`0x9bc94188116067fb2b8194a30af7ee904e6b3473a831cc02ac76883431adbb84`

**collective.xyz registration:**
`0xcf56c4f4b6c4e5e9cd9b91886062e9e8e7055e9b8735f05f20e258e09f7eb2f4`

**ecollective.xyz registration:**
`0x1663883aceb6f690c99cb6b28d5dc4d9d21141027113cb90465a628eb07b0ca2`

# Appendix E.

# DNS Top-Level-Domains in ENS

In Chapter 5, we develop a dataset for all domains that have been bridged to ENS using DNSSEC. This appendix highlights the distribution of domains over the relevant TLDs.

Figure E.1 gives an overview of all top-level-domains managed within ENS that stem from traditional DNS. Three main TLD categories exist (IANA, 2023b):

- `Generic (gTLD)`: Generic TLDs that do not belong to a country or state.
- `Country-code (ccTLDs)`: TLDs that belong to a country and are managed by respective entities. Noteworthy are TLDs such as `.io` or `.id` that are assigned to British Indian Ocean Territory and Indonesia, respectively. However, communities, such as the computer science community, use these domains for their own purpose as a reference to *Input/Output*.
- `Generic-restricted`: Domains registered in these TLDs must satisfy specific requirements.

In the ENS dataset, three different TLDs are present in the `generic-restricted` category: `.name` (2 records), `.biz` (1 record), and `.pro` (1 record) with the following restrictions:

- `.name` is intended to be used by private persons only[1],
- `.biz` must be used primarliy for commercial purposes[2], and
- `.pro` registrants need to state their profession[3].

[1]See `https://www.verisign.com/en_US/domain-names/name-domains`, accessed 9th March 2023.
[2]See `https://www.icann.org/resources/unthemed-pages/registry-agmt-appl-2001-04-18-en.html`, accessed 9th March 2023.
[3]See `http://www.prweb.com/releases/register_a_pro/domain_name/prweb13076072.htm`, accessed 9th March 2023.

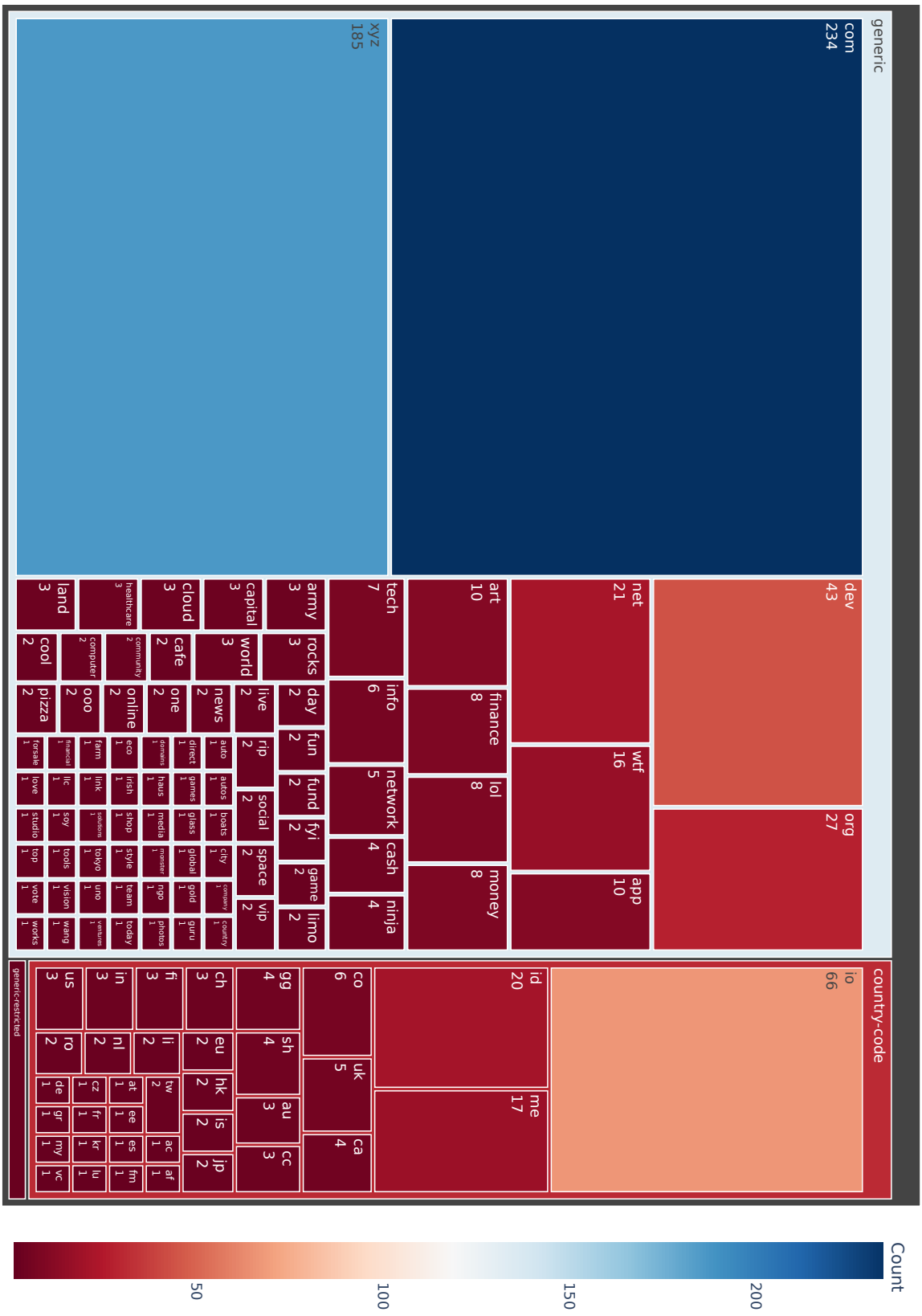



**Figure E.1.:** Treemap of all TLDs bridged via DNSSEC managed within ENS as of 28th February 2023.

# Appendix F.

# DNSSEC Support in Top-Level Domains

In Chapter 6, we investigate the DNSSEC-support of all TLDs and develop a respective dataset including domain count for each TLD. This appendix gives further insights into this dataset.

Figure F.1 gives an overview over all top-level domains categorized by their DNSSEC support. The sizes of the tiles are dependent on the logarithmic count of the domains.

To generate Figure F.1, we

- leveraged the list of TLDs that support DNSSEC by Openprovider (Openprovider, 2023), and
- merged it with a list of domain registrations per TLD by Domain Name Stat (Domain Name Stat, 2023).

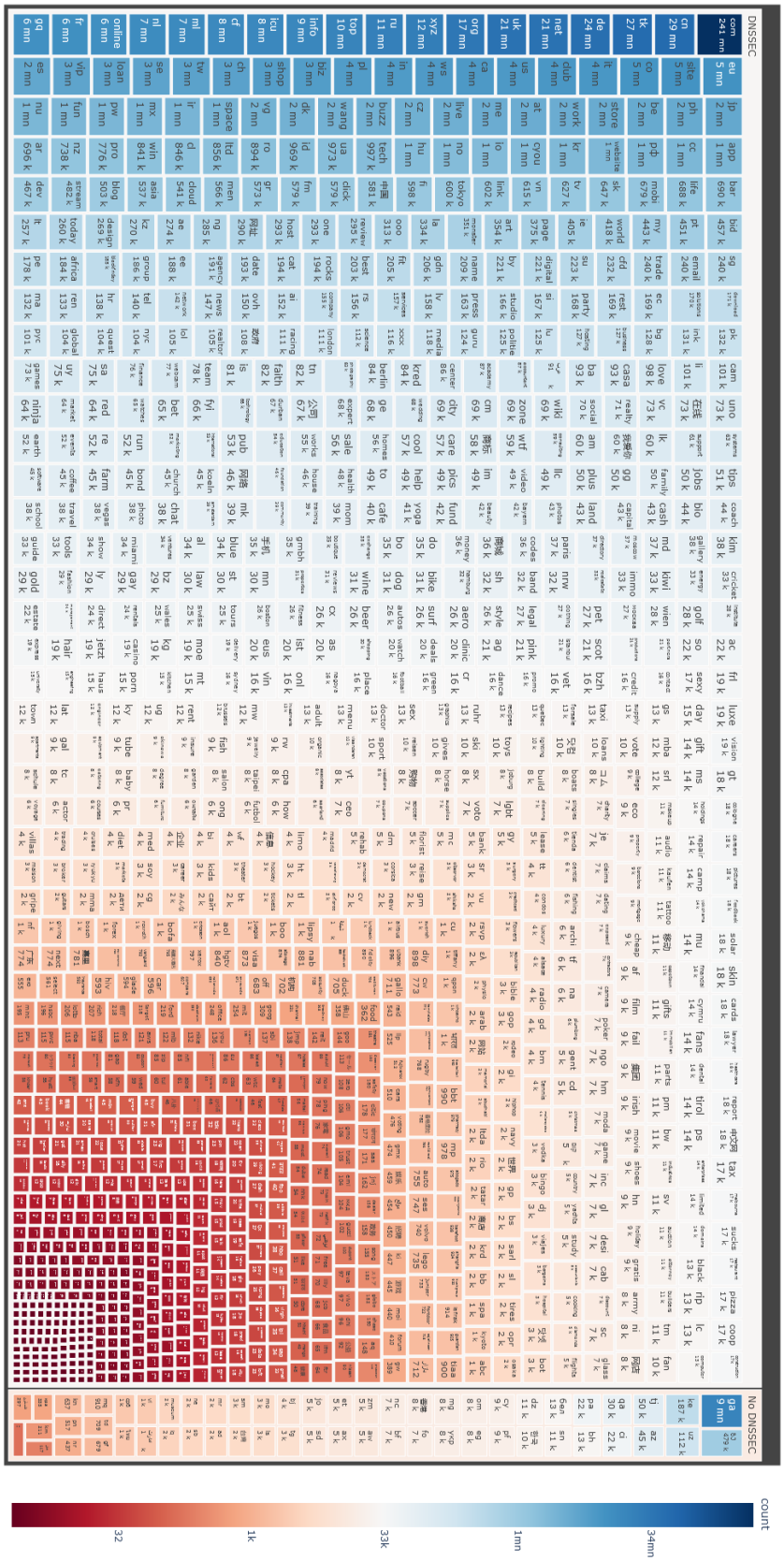



**Figure F.1.:** Treemap of all TLDs categorized by their DNSSEC support as of 28th February 2023.

# Bibliography


APNIC Labs (2023). DNSSEC Validation Rate by country. https://stats.labs.apnic.net/dnssec. Accessed: 2023-02-28.

Allen, Christopher and Tim Dierks (Jan. 1999). The TLS Protocol Version 1.0. RFC 2246. DOI: 10.17487/RFC2246. URL: https://rfc-editor.org/rfc/rfc2246.txt.

Amann, Johanna, Oliver Gasser, Quirin Scheitle, Lexi Brent, Georg Carle, and Ralph Holz (2017). Mission accomplished? HTTPS security after DigiNotar. *Proceedings of the 2017 Internet Measurement Conference*, pp. 325–340.

Arends, Roy, Geoffrey Sisson, David Blacka, and Ben Laurie (Mar. 2008). DNS Security (DNSSEC) Hashed Authenticated Denial of Existence. RFC 5155. DOI: 10.17487/RFC5155. URL: https://www.rfc-editor.org/info/rfc5155.

Barnes, Richard, Jacob Hoffman-Andrews, Daniel McCarney, and James Kasten (Mar. 2019). Automatic Certificate Management Environment (ACME). RFC 8555. DOI: 10.17487/RFC8555. URL: https://www.rfc-editor.org/info/rfc8555.

Benet, Juan (2014). Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*.

Berners-Lee, Tim and Daniel W. Connolly (Nov. 1995). Hypertext Markup Language - 2.0. RFC 1866. DOI: 10.17487/RFC1866. URL: https://www.rfc-editor.org/info/rfc1866.

Blocher, Walter et al. (2019). *Rechtshandbuch Smart Contracts*. CH Beck.

Blockchair (2023). Index of /ethereum/blocks/. https://gz.blockchair.com/ethereum/blocks/. Accessed: 2023-02-27.

Boeyen, Sharon, Stefan Santesson, Tim Polk, Russ Housley, Stephen Farrell, and Dave Cooper (May 2008). Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280. DOI: 10.17487/RFC5280. URL: https://rfc-editor.org/rfc/rfc5280.txt.

Brunner, Clemens, Fabian Knirsch, Andreas Unterweger, and Dominik Engel (2020). A Comparison of Blockchain-based PKI Implementations. *Proceedings of the 6th International Conference on Information Systems Security and Privacy - Volume 1:*

*ICISSP,* INSTICC. SciTePress, pp. 333–340. ISBN: 978-989-758-399-5. DOI: 10.5220/0008914503330340.

BuiltWith (2022). Extended Validation Usage Distribution in the Top 1 Million Sites. https://trends.builtwith.com/ssl/extended-validation. Accessed: 2022-09-04.

CNET (2015). This guy bought Google.com for $12. https://www.cnet.com/tech/services-and-software/this-guy-bought-the-most-valuable-website-in-the-world-for-12/. Accessed: 2023-03-04.

Chartrand, James, Stuart Freeman, Ulrich Gallersdörfer, Matt Lisle, Alexander Mühle, and Sélinde van Engelenburg (2020). Building the digital credential infrastructure for the future. *The Digital Credentials Consortium.* URL: https://digitalcredentials.mit.edu/docs/white-paper-building-digital-credential-infrastructure-future.pdf.

ConsenSys (2023). Reentrancy - Ethereum Smart Contract Best Practices. https://consensys.github.io/smart-contract-best-practices/attacks/reentrancy/. Accessed: 2023-03-04.

Crocker, Dave (Mar. 2019). Scoped Interpretation of DNS Resource Records through "Underscored" Naming of Attribute Leaves. RFC 8553. DOI: 10.17487/RFC8553. URL: https://www.rfc-editor.org/info/rfc8553.

Dierks, Tim and Eric Rescorla (Apr. 2006). The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346. DOI: 10.17487/RFC4346. URL: https://rfc-editor.org/rfc/rfc4346.txt.

Dnspython (2023). dnspython. https://www.dnspython.org/. Accessed: 2023-02-27.

Domain Name Stat (2023). Domain name registrations in All TLDs. https://domainnamestat.com/statistics/tldtype/all. Accessed: 2023-02-28.

Dune Analytics AS (2023). Dune: Crypto data by and for the community. https://dune.com. Accessed: 2023-02-27.

Duong, Tuyet Anh (2022). Quick Analysis of the Wormhole attack. https://research.kudelskisecurity.com/2022/02/03/quick-analysis-of-the-wormhole-attack/. Accessed: 2023-03-08.

EFF (2021). HTTPS Is Actually Everywhere. https://www.eff.org/deeplinks/2021/09/https-actually-everywhere. Accessed: 2023-02-28.

Eastlake 3rd, Donald E. (Mar. 1999). Domain Name System Security Extensions. RFC 2535. DOI: 10.17487/RFC2535. URL: https://www.rfc-editor.org/info/rfc2535.

— (Jan. 2011). Transport Layer Security (TLS) Extensions: Extension Definitions. RFC 6066. DOI: 10.17487/RFC6066. URL: https://www.rfc-editor.org/info/rfc6066.

Ebel, Jonas (2021). Augmenting the MetaMask-Wallet with Domain Name based Authentication of Ethereum Accounts. *Master's Thesis*. Technical University of Munich.

Ethereum Name Service (2023a). ENS Contract. https://github.com/ensdomains/ens-contracts. Accessed: 2023-03-04.

— (2023b). ENS Documentation. https://docs.ens.domains. Accessed: 2023-03-01.

— (2023c). Ethereum Name Service. https://ens.domains/. Accessed: 2023-03-15.

— (2023d). NameWrapper docs. https://github.com/ensdomains/ens-contracts/tree/master/contracts/wrapper. Accessed: 2023-03-11.

Etherscan (2023a). Contract Source Code Verified. https://etherscan.io/address/0x58774bb8acd458a640af0b88238369a167546ef2. Accessed: 2023-03-04.

— (2023b). Contract Source Code Verified. https://etherscan.io/address/0x21745ff62108968fbf5ab1e07961cc0fcbeb2364. Accessed: 2023-03-04.

— (2023c). Ethereum Full Node Sync (Archive) Chart — Etherscan. https://etherscan.io/chartsync/chainarchive. Accessed: 2023-02-16.

— (2023d). Ethereum Gas Tracker. https://etherscan.io/gastracker. Accessed: 2023-03-02.

Felt, Adrienne Porter et al. (2015). Improving SSL warnings: Comprehension and adherence. *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, pp. 2893–2902.

Felt, Adrienne Porter et al. (2016). Rethinking connection security indicators. *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*. USENIX Association, pp. 1–14.

Freier, Alan O., Philip Karlton, and Paul C. Kocher (Aug. 2011). The Secure Sockets Layer (SSL) Protocol Version 3.0. RFC 6101. DOI: 10.17487/RFC6101. URL: https://www.rfc-editor.org/info/rfc6101.

Fröwis, Michael, Andreas Fuchs, and Rainer Böhme (2019). Detecting Token Systems on Ethereum. *Financial Cryptography and Data Security*. Ed. by Ian Goldberg and Tyler Moore. Cham: Springer International Publishing, pp. 93–112. ISBN: 978-3-030-32101-7.

Gallersdörfer, Ulrich, Jonas Ebel, and Florian Matthes (2021b). Augmenting MetaMask to support TLS-endorsed Smart Contracts – Supplementary Material. https://github.com/UliGall/paper-tesc-metamask.

Gallersdörfer, Ulrich, Jonas Ebel, and Florian Matthes (2021a). Augmenting MetaMask to support TLS-endorsed Smart Contracts. *5th International Workshop on Cryptocurrencies and Blockchain Technology*.

Gallersdörfer, Ulrich, Friederike Groschupp, and Florian Matthes (2021d). Mirroring Public Key Infrastructures to Blockchains for On-chain Authentication – Supplementary Material.

— (2021c). Mirroring Public Key Infrastructures to Blockchains for On-chain Authentication. *5th Workshop on Trusted Smart Contracts In Association with Financial Cryptography 2021*.

Gallersdörfer, Ulrich, Patrick Holl, and Florian Matthes (2020). Blockchain-based Systems Engineering – Lecture Slides. URL: `https://github.com/sebischair/bbse`.

Gallersdörfer, Ulrich and Florian Matthes (2020). AuthSC: mind the gap between web and smart contracts. *arXiv preprint arXiv:2004.14033*. URL: `http://arxiv.org/abs/2004.14033`.

— (2021b). TeSC: TLS/SSL-Certificate Endorsed Smart Contracts – Supplementary Material. `https://github.com/UliGall/paper-TeSC`. Accessed: 2022-10-31.

— (2021a). TeSC: TLS/SSL-Certificate Endorsed Smart Contracts. *The 3rd IEEE International Conference on Decentralized Applications and Infrastructures*.

Glomann, Leonhard, Maximilian Schmid, and Nika Kitajewa (2020). Improving the blockchain user experience-an approach to address blockchain mass adoption issues from a human-centred perspective. *Advances in Artificial Intelligence, Software and Systems Engineering: Proceedings of the AHFE 2019 International Conference on Human Factors in Artificial Intelligence and Social Computing, the AHFE International Conference on Human Factors, Software, Service and Systems Engineering, and the AHFE International Conference of Human Factors in Energy, July 24-28, 2019, Washington DC, USA 10*. Springer, pp. 608–616.

Google (2023). HTTPS encryption on the web. `https://transparencyreport.google.com/https/overview`. Accessed: 2023-02-28.

Grajek, Garret, Stephen Moore, and Mark Lambiase (2010). Method and system for generating digital certificates and certificate signing requests. US Patent App. 12/326,002.

Groschupp, Friederike (2020). Exploring the Use of SSL/TLS Certificates for Identity Assertion and Verification in Ethereum. *Master's Thesis*. Technical University of Munich.

Herrmann, Pascal, Manov Metodi Ma Tuan Anh, and David Stübing (2021). TLS-based Authentication Management on Blockchain. *Project Work in Software-Engineering for Business Applications - Lab (IN2106, 2129, 4077)*. Technical University of Munich.

Hickson, Ian and David Hyatt (2011). Html5. *W3C Working Draft WD-html5-20110525*, p. 53.

Hoops, Jan Felix (2021). Threat Analysis, Evaluation, and Mitigation for Smart Contracts Endorsed by TLS/SSL Certificates. *Master's Thesis*. Technical University of Munich.

Huston, Geoff (2023). To DNSSEC or Not? `https://labs.apnic.net/index.php/2023/02/21/to-dnssec-or-not/`. Accessed: 2023-02-28.

IANA (2022). About us. `https://www.iana.org/about`. Accessed: 2022-06-20.

— (2023a). Domain Name System (DNS) Parameters – Underscored and Globally Scoped DNS Node Names. `https://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml#underscored-globally-scoped-dns-node-names`. Accessed: 2023-03-04.

— (2023b). Root Zone Database. `https://www.iana.org/domains/root/db`. Accessed: 2023-02-27.

ICANN (2023). Domain Name System Security (DNSSEC) Algorithm Numbers. `https://www.iana.org/assignments/dns-sec-alg-numbers/dns-sec-alg-numbers.xhtml`. Accessed: 2023-03-02.

ISO (2022). ISO/IEC 9594-8:2020. `https://www.iso.org/standard/80325.html`. Accessed: 2022-09-04.

ISRG (2023). Let's build a better Internet – 2022 Annual Report. `https://www.abetterinternet.org/documents/2022-ISRG-Annual-Report.pdf`. Accessed: 2023-02-28.

Internet Society (2022a). DNSSEC Statistics. `https://www.internetsociety.org/deploy360/dnssec/statistics/`. Accessed: 2022-09-04.

— (2022b). Our Vision: The Internet Is for Everyone. `https://www.internetsociety.org/mission/`. Accessed: 2022-05-19.

Johnson, Nick (2020). ERC-3668: CCIP Read: Secure offchain data retrieval. `https://eips.ethereum.org/EIPS/eip-3668`. Ethereum Improvement Proposal.

King, April, Lucas Garron, and Chris Thompson (2022). https://badssl.com. `https://badssl.com`. Accessed: 2023-02-22.

Kraus, Lydia, Martin Ukrop, Vashek Matyas, and Tobias Fiebig (2020). Evolution of SSL/TLS Indicators and Warnings in Web Browsers. *Security Protocols XXVII: 27th International Workshop, Cambridge, UK, April 10–12, 2019, Revised Selected Papers 27*. Springer, pp. 267–280.

Kushwaha, Satpal Singh, Sandeep Joshi, Dilbag Singh, Manjit Kaur, and Heung-No Lee (2022). Systematic review of security vulnerabilities in ethereum blockchain smart contract. *IEEE Access* 10, pp. 6605–6621.

Käslin, Kilian (2020). Establishment of a Minimum Viable Self-Sovereign Identity Network. *Master's Thesis*. Technical University of Munich.

Laurie, Ben, Adam Langley, and Emilia Kasper (June 2013). Certificate Transparency. RFC 6962. DOI: 10.17487/RFC6962. URL: https://www.rfc-editor.org/info/rfc6962.

Laurie, Ben, Adam Langley, Emilia Kasper, Eran Messeri, and Rob Stradling (Dec. 2021). Certificate Transparency Version 2.0. RFC 9162. DOI: 10.17487/RFC9162. URL: https://www.rfc-editor.org/info/rfc9162.

Lazarenko, Aleksandr and Sergey Avdoshin (2018). Financial risks of the blockchain industry: A survey of cyberattacks. *Proceedings of the Future Technologies Conference*. Springer, pp. 368–384.

Let's Encrypt (2022). ACME Client Implementations. https://letsencrypt.org/docs/client-options/. Accessed: 2022-09-04.

— (2023). A nonprofit Certificate Authority providing TLS certificates to 300 million websites. https://letsencrypt.org/. Accessed: 2023-02-28.

Lovelock, John-David, Martin Reynolds, B Granetto, and Rajesh Kandaswamy (2017). Forecast: Blockchain business value, worldwide, 2017-2030. *Gartner, Stamford, USA*.

Mazorra, Bruno, Victor Adan, and Vanesa Daza (2022). Do not rug on me: Zero-dimensional Scam Detection. *arXiv preprint arXiv:2201.07220*.

McFadden, Mark (June 2019). *Methodology for Researching Security Considerations Sections*. Internet-Draft draft-mcfadden-smart-rfc3552-research-methodology-00. Work in Progress. Internet Engineering Task Force. 10 pp. URL: https://datatracker.ietf.org/doc/draft-mcfadden-smart-rfc3552-research-methodology/00/.

MetaMask (2023). A crypto wallet & gateway to blockchain apps. https://metamask.io/. Accessed: 2023-03-09.

Mockapetris, Paul (Nov. 1983a). Domain names: Concepts and facilities. RFC 882. DOI: 10.17487/RFC0882. URL: https://www.rfc-editor.org/info/rfc882.

— (Nov. 1983b). Domain names: Implementation specification. RFC 883. DOI: 10.17487/RFC0883. URL: https://www.rfc-editor.org/info/rfc883.

— (Nov. 1987a). Domain names - concepts and facilities. RFC 1034. DOI: 10.17487/RFC1034. URL: https://rfc-editor.org/rfc/rfc1034.txt.

— (Nov. 1987b). Domain names - implementation and specification. RFC 1035. DOI: 10.17487/RFC1035. URL: https://rfc-editor.org/rfc/rfc1035.txt.

Mozilla Developer Network (2022). webRequest.getSecurityInfo() – Mozilla — MDN. URL: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/webRequest/getSecurityInfo.

Nakamoto, Satoshi (2008). Bitcoin: A peer-to-peer electronic cash system.

Narayanan, Arvind, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder (2016). *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton University Press.

Nilsson, Alexander, Pegah Nikbakht Bideh, and Joakim Brorsson (2020). A survey of published attacks on Intel SGX. *arXiv preprint arXiv:2006.13598*.

Nottingham, Mark (May 2019). Well-Known Uniform Resource Identifiers (URIs). RFC 8615. DOI: 10.17487/RFC8615. URL: https://www.rfc-editor.org/info/rfc8615.

Openprovider (2023). List of TLDs that support DNSSEC. https://support.openprovider.eu/hc/en-us/articles/216648838-List-of-TLDs-that-support-DNSSEC. Accessed: 2023-02-28.

Parity Technologies (2017). A Postmortem on the Parity Multi-Sig Library Self-Destruct. https://www.parity.io/blog/a-postmortem-on-the-parity-multi-sig-library-self-destruct/. Accessed: 2023-02-15.

Provable (2022). Provable Documentation. http://docs.provable.xyz/#security-deepdive-authenticity-proofs-types. Accessed: 2022-09-11.

Rescorla, Eric (Aug. 2018). The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. DOI: 10.17487/RFC8446. URL: https://rfc-editor.org/rfc/rfc8446.txt.

Rescorla, Eric and Tim Dierks (Aug. 2008). The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246. DOI: 10.17487/RFC5246. URL: https://rfc-editor.org/rfc/rfc5246.txt.

Rey, Richard et al. (2023). CheckMyHTTPS – Check your secured WEB flows. https://checkmyhttps.net/. Accessed: 2023-02-13.

Ritchie, Hannah, Edouard Mathieu, Max Roser, and Esteban Ortiz-Ospina (2023). Internet. *Our World in Data*. https://ourworldindata.org/internet.

Rose, Scott, Matt Larson, Dan Massey, Rob Austein, and Roy Arends (Mar. 2005a). DNS Security Introduction and Requirements. RFC 4033. DOI: 10.17487/RFC4033. URL: https://www.rfc-editor.org/info/rfc4033.

Rose, Scott, Matt Larson, Dan Massey, Rob Austein, and Roy Arends (Mar. 2005b). Protocol Modifications for the DNS Security Extensions. RFC 4035. DOI: 10.17487/RFC4035. URL: https://www.rfc-editor.org/info/rfc4035.

— (Mar. 2005c). Resource Records for the DNS Security Extensions. RFC 4034. DOI: 10.17487/RFC4034. URL: https://www.rfc-editor.org/info/rfc4034.

SSLMate (2023). Cert Spotter - Stats. https://sslmate.com/resources/certspotter_stats. Accessed: 2023-02-13.

Sleevi, Ryan (2016). New CT Policy for Chrome Published. https://archive.cabforum.org/pipermail/public/2016-May/007573.html. Accessed: 2022-09-04.

Solidity Team (2023). Solidity Releases. https://blog.soliditylang.org/category/releases/. Accessed: 2023-03-04.

Sporny, Manu, Dave Longley, Markus Sabadello, Drummond Reed, Orie Steele, and Christopher Allen (2022). Decentralized Identifiers (DIDs) v1.0. https://www.w3.org/TR/did-core/. W3C Recommendation.

StJohns, Michael (Sept. 2007). Automated Updates of DNS Security (DNSSEC) Trust Anchors. RFC 5011. DOI: 10.17487/RFC5011. URL: https://www.rfc-editor.org/info/rfc5011.

Statista (2022). Number of worldwide internet hosts in the domain name system (DNS) from 1993 to 2019. https://www.statista.com/statistics/264473/number-of-internet-hosts-in-the-domain-name-system/. Accessed: 2022-09-04.

Stevens, Marc, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov (2017). The first collision for full SHA-1. *Advances in Cryptology–CRYPTO 2017: 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2017, Proceedings, Part I 37*. Springer, pp. 570–596.

Strugala, Jan-Niklas (2020). Leveraging TLS/SSL-based Identity Assertion and Verification Systems for On-chain Authentication and Authorization of Real-world Entities. *Master's Thesis*. Technical University of Munich.

Styler, Joe (2022). The top 25 most expensive domain names. https://www.godaddy.com/garage/the-top-20-most-expensive-domain-names/. Accessed: 2022-09-04.

Sun, Zhiyuan (2022). Anonymous user sends ETH from Tornado Cash to prominent figures following sanctions. https://cointelegraph.com/news/anonymous-user-sends-eth-from-tornado-cash-to-prominent-figures-following-sanctions. Accessed: 2022-10-09.

Szabo, Nick (1997). Formalizing and securing relationships on public networks. *First monday*.

Szurdi, Janos, Balazs Kocso, Gabor Cseh, Jonathan Spring, Mark Felegyhazi, and Chris Kanich (2014). The long "taile" of typosquatting domain names. *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pp. 191–206.

TBS Internet (2021). SHA1 Root Certificates - the case of servers returning the root certificate. `https://www.tbs-certificates.co.uk/FAQ/en/serveur-avec-racine.html`. Accessed: 2023-03-03.

Thompson, Christopher, Martin Shelton, Emily Stark, Maximilian Walker, Emily Schechter, and Adrienne Porter Felt (2019). The web's identity crisis: understanding the effectiveness of website identity indicators. *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pp. 1715–1732.

Ukrop, Martin, Lydia Kraus, Vashek Matyas, and Heider Ahmad Mutleq Wahsheh (2019). Will you trust this tls certificate? perceptions of people working in it. *Proceedings of the 35th annual computer security applications conference*, pp. 718–731.

Waked, Louis, Mohammad Mannan, and Amr Youssef (2020). The sorry state of TLS security in enterprise interception appliances. *Digital Threats: Research and Practice* 1 (2), pp. 1–26.

Wieczner, Jen (2017). Ethereum: CoinDash ICO Hacked, $7 Million in Ether Stolen — Fortune. URL: `https://fortune.com/2017/07/18/ethereum-coindash-ico-hack/`.

Willmore, Joel (2022). The Seal of Approval: Know What You're Consenting To With Permissions and Approvals in MetaMask. `https://consensys.net/blog/metamask/the-seal-of-approval-know-what-youre-consenting-to-with-permissions-and-approvals-in-metamask/`. Accessed: 2022-10-30.

Wood, Gavin et al. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* 151 (2014), pp. 1–32.

Xia, Pengcheng, Haoyu Wang, Zhou Yu, Xinyu Liu, Xiapu Luo, Guoai Xu, and Gareth Tyson (2022). Challenges in decentralized name management: the case of ENS, pp. 65–82.

Zhang, Fan, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi (2016). Town Crier: An Authenticated Data Feed for Smart Contracts. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS '16. Vienna, Austria: ACM, pp. 270–282. ISBN: 978-1-4503-4139-4. DOI: `10.1145/2976749.2978326`. URL: `http://doi.acm.org/10.1145/2976749.2978326`.

Zimmermann, Hubert (1980). OSI reference model-the ISO model of architecture for open systems interconnection. *IEEE Transactions on communications* 28 (4), pp. 425–432.

http.dev (2023). 999 Request Denied. `https://http.dev/999`. Accessed: 2023-02-28.