Stefan More

Trust and Privacy in a Heterogeneous World

Doctoral Thesis

to achieve the university degree of
Doktor der technischen Wissenschaften

submitted to
Graz University of Technology

Advisors: Arne Tauber, Peter Lipp

Assessors: Reinhard Posch
Graz University of Technology

Antonio F. Skarmeta
University of Murcia

Institute of Applied Information Processing and Communications (IAIK)
Graz University of Technology

Graz, October 2023

AFFIDAVIT

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

SIGNATURE INFORMATION	**Signatory**	Stefan Josef More
	Date/Time-UTC	2023-10-31T17:37:49+01:00
	Verification	Information about the verification of the electronic signature can be found at: https://www.signaturpruefung.gv.at
Note	This document is signed with a qualified electronic signature. According to Art. 25 para. 2 of the Regulation (EU) No 910/2014 of 23. July 2014 ("eIDAS-Regulation") it shall have the equivalent legal effect of a handwritten signature.	

Signature

Abstract

In an era where the Internet plays an ever-expanding role, trust is the cornerstone of secure and seamless digital interactions. Trust verification is critical in diverse electronic transactions within our increasingly interconnected digital landscape. This thesis enhances trust verification in diverse electronic transactions within a heterogeneous context.

We address challenges arising from complex electronic transactions covering various trust aspects.

More specifically, we consider transactions consisting of multiple digital credentials issued by different entities under varying trust schemes. To simplify trust establishment across schemes, we propose a trust management infrastructure based on the Domain Name System (DNS). This infrastructure enables verifiers to establish trust in new schemes using human-readable identifiers instead of manually configuring trust anchors and cryptographic material. We also introduce support for global trust scheme recognition and automated trust translations. By doing so, we establish interoperability between trust schemes that use different understandings (or encodings) of trust.

To address the verifier's individual trust perspectives, we introduce an expressive trust policy system, facilitating verifiers to define trust criteria tailored to their use case. Our system's extensibility accommodates future needs and integrates with DNS-based and distributed ledger-based trust management, such as in self-sovereign identity models.

Dealing with diverse credential formats is common in a global context, resulting in interoperability issues. To mitigate those issues, we introduce a framework for trustworthy credential transformations. Our framework allows verifiers to automatically transform data from unknown schemata into a schema they can parse.

This thesis also considers privacy aspects regarding transaction content and user behavior. We extend expressive access control systems with privacy features, enabling seamless integration of privacy-preserving technologies. By introducing a ledger state attestation system, we enhance distributed ledger-based registries to ensure data provenance without compromising user privacy.

Acknowledgements

I would like to thank my thesis' assessors, Prof. Antonio F. Skarmeta for his rapid reviews and helpful feedback, and Prof. Reinhard Posch, also for founding and growing IAIK. Thanks to my advisors, Arne Tauber, for his feedback and the freedom to do my research, and Peter Lipp, for giving me the opportunity to pursue a PhD at IAIK.

I thank the institutions who provided the funding for my research, the European Union's Horizon 2020 research and innovation programme under grant agreements № 700321 (LIGHTest), № 871473 (KRAKEN), № 101020416 (Eratosthenes), and the Secure Information Technology Center Austria (A-SIT).

I am very grateful for the support, collaboration, and encouragement from my "remote mentor", Sebastian Ramacher. A special thanks goes to Andreas Abraham for helping me focus my research, and to Jakob Heher, not only for proofreading this thesis and productive procrastination. I am glad for all curious discussions with my academic (travel) companion Lukas Alber.

Furthermore, I want to thank my "office cohabitants", colleagues, fellow students, and co-authors, including but not limited to, Georg Wagner, Florian Draschbacher, Stephan Keller, Blaž Podgorelec, Christof Rabensteiner, Felix Hörandner, Karl-Christian Posch, Sebastian Mödersheim, Anders Schlichtkrull, Tilen Marc, Bud P. Bruegger, Heiko Roßnagel, Sven Wagner, Martin Hoffmann, Stephan Krenn, Jesús García, Lukas Daniel Klausner, Maria Eichlseder, Thomas Lenz, Bernd Prünster, Hannes Weissteiner, Simon Guggi, Johannes Haring, Benedikt Maderbacher, Nicoo Braud-Santoni, Karl Koch, Lukas Prokop, Daniel Gruss, and Michael Schwarz.

Many thanks for support and continuous discussions to Moritz Lipp, as well as Christian Kollmann, Christopher Kaponig, and Patrick Klampfl. Sincere thanks are given to everyone at IAIK who keeps the institute running. A big thanks also to Michael Rodler for starting our CTF team LosFuzzys, and to all Fuzzys for inspiration and room to grow; happy hacking!

I want to express my gratitude to everyone who supported, cheered, or distracted me throughout my PhD journey. To everyone from realraum, Cryptoparty Graz, STG, org, Elevate, Salon Leobard, and all my other rabbit holes and various homes over the years. To Christian, Bernhard, Mika, and Karl. To my co-conspirators Jakob, Marc, and Peter. To Leo, Julia, Schatz, Klaus, Tatz, Moony, Mona, Katja, ... To Alex, Geri, Dominika, Josef, Michaela, Ramona, Lene, Vicky, Klaus, Hannah, Konstantin, Bine, Daniel, Christina, Marianne, Oz, Lukas, Coco, Isa, Angela, Eva, Gaby, Christina, Priska, Cora, ... and everyone with whom I shared adventure, debate, and glitter.

I am especially grateful to Georg Holzer for creating communities, for fighting corruption on the local level, and for showing me early the wonderful experience of bringing people together. Cheers to you, Georg!

I thank my family for their support and love during the 33-year-long journey to this dissertation. My parents Elisabeth & Peter, for the opportunities they gave me and their trust, my "little" brother Florian, my grandparents Ata & Opipa, Johann & Ilse, and my godfather Hans.

Finally, I am deeply grateful to Anna. Thanks for all your support, your patience, our reflections, the many adventures, and all the moments filled with wonder. Onwards to the next adventures!

Thanks, everyone! Allons-y!

Stefan

Graz, October 2023

Contents

List of Figures

List of Tables

List of Listings

List of Abbreviations

ASiC Associated Signature Container.

ATV Automated Trust Verifier.

CA Certificate Authority.

CAdES CMS Advanced Electronic Signature.

ccTLD Country Code Top-Level Domain.

CMS Cryptographic Message Syntax.

CRL Certificate Revocation List.

CRS Common Reference String.

DER Distinguished Encoding Rules.

DID Decentralized Identifier.

DL Distributed Ledger.

DNS Domain Name System.

DNSSEC DNS Security Extensions.

EBNF Extended Backus–Naur form.

eIDAS electronic Identification, Authentication and Trust Services.

ETSI European Telecommunications Standards Institute.

EU European Union.

EVM Ethereum Virtual Machine.

FE Functional Encryption.

FHE Full homomorphic encryption.

GDPR General Data Protection Regulation.

GTPL Graphical TPL.

HSM Hardware Security Module.

HTTP Hypertext Transfer Protocol.

IDP Identity Provider.

IETF Internet Engineering Task Force.

IPFS InterPlanetary File System.

ISO International Organization for Standardization.

ITU International Telecommunication Union.

JMH Java Microbenchmark Harness.

JSON JavaScript Object Notation.

KSK Key Signing Key.

LoA Level of Assurance.

LOTL List of Trusted Lists.

MPC Multi-Party Computation.

NIZK Non-Interactive Zero-Knowledge Proof.

OCSP Online Certificate Status Protocol.

PAdES PDF Advanced Electronic Signature.

PDF Portable Document Format.

PET Privacy-enhancing Technology.

PII Personally Identifiable Information.

PKI Public Key Infrastructure.

QTSP Qualified Trust Service Provider.

SAML Security Assertion Markup Language.

SNARK Succinct Non-Interactive Argument of Knowledge.

SP Service Provider.

SSI Self-Sovereign Identity.

TDR Transformation Data Registry.

TI Transformation Information.

TLD Top-Level Domain.

TLS Transport Layer Security.

TPAT Trust Policy Authoring Tool.

TSL Trust (Service) Status List.

TSP Trust Service Provider.

TSPA Trust Scheme Publication Authority.

URI Uniform Resource Identifier.

URL Uniform Resource Locator.

VC W3C Verifiable Credential.

VI Verification Information.

W3C World Wide Web Consortium.

WoT Web of Trust.

XAdES XML Advanced Electronic Signature.

XML Extensible Markup Language.

XSD XML Schema Definition.

XSLT Extensible Stylesheet Language Transformations.

ZSK Zone Signing Key.

Problem Statement

1 Introduction

The Internet has become an integral part of our daily lives, providing access to a vast array of information, services, and communication tools. In our connected world, trust and privacy are essential concepts. Trust enables individuals and organizations to rely on each other, facilitating the smooth functioning of our digital world. Privacy technology allows individuals to control their personal information and protect themselves from unwanted surveillance or profiling. As our reliance on the Internet increases, so does the importance of trust and privacy.

Trust is essential for building and maintaining relationships, both online and offline. Without trustworthy information, individuals and organizations may be hesitant or unable to share personal information, conduct transactions, or engage in other forms of online activity. For example, online shoppers may hesitate to provide personal information to an e-commerce site they do not trust to be legitimate. Conversely, a business may be reluctant to accept an order without knowledge about the buyer's identity and financial standing.

If we talk to someone over the Internet, we want to be sure that this someone is really who they claim to be. In the case of online browsing, we rely on our browser to actually communicate with the website we requested, and not with Eve.[1] Trust is also important for other types of transactions: For software downloads or updates over the Internet, users want to trust that the code they download is not replaced with malware by a hosting provider or during transmission. Trust is not always about the identities of the actors: For computations outsourced to the cloud, it is important that the results are trustworthy. In the case of eCommerce, the owner of an online bookstore might not care about the identity of customers as long as she can trust that they pay their books on time and that the money is not counterfeit. Or, when we show our electronic ticket

[1] In the world of Alice and Bob, the Eve character is usually an eavesdropper [BBR85].

to board a train, the automatic gate needs to ensure that our ticket was not manipulated and that it was really issued by the transport agency, but it is irrelevant who we are as long as the ticket is valid. To ensure trust on a technical level, measures like digital signatures and trust schemes are used.

Privacy is also a critical concern on the Internet. Personal information, such as financial data, medical records, and browsing history, could be easily collected, shared, and analyzed against a user's will. This misuse can lead to identity theft, fraud, and other forms of abuse. For example, companies may collect and sell personal information without our knowledge, or governments may use this information for surveillance purposes. This can lead to a loss of trust and control over our personal information, which can affect our security, autonomy, and well-being. Examples are financial loss, reputation damage, psychological distress, or discrimination.

Without proper privacy measures, personal information could be misused or disclosed, leading to serious consequences for the individuals involved. One common example of a privacy measure is using pseudonyms or anonymous identities. Such technology allows individuals to use digital identity systems without revealing their real-world identity, protecting them from potential privacy breaches or discrimination. An example relevant in the context of this thesis is the use of decentralized identity systems, which allow individuals to have more control of their identity data. In such systems, the data is stored decentralized like on the user's phones, providing an additional layer of security and privacy. Further, zero-knowledge proofs allow individuals to prove that they possess certain information without revealing it. These tools can be used in digital identity systems to ensure that private information remains confidential while allowing for secure authentication and authorization.

1.1. Thesis Objectives

Motivated by the importance of both trust and privacy, we present several contributions to the two subjects. On a functional level, this thesis focuses on three aspects of trust and privacy: the encoding and verification of *Credentials*, the trust- and access *Policies* governing this verification, and the *Infrastructure* needed to do so. On the subject of trust, we focus on automated trust management in a global context. To enable this, we develop solutions for establishing trust into electronic transactions in a

heterogeneous environment. On the subject of privacy, we improve the privacy of users interacting with an automated system using access control. In particular, we focus on the process of authenticating of an user with a service by presenting some identity credential. The objective of this thesis address is to consider several points we consider relevant for those goals:

- How can the trustworthiness of complex electronic transactions be verified? We consider transactions that consist of multiple documents, each issued by a different authority, contributing a different perspective to the trust assessment. How can this transaction be verified even if the transaction stems from a different country? While several countries provide the infrastructure to do this on a local level, in Chapter 6, we aim to deliver the interoperability layer to support automated trust management on a global scale. This includes the discovery of trust status data and dealing with the heterogeneous nature of the involved credential systems in Chapter 8.

- How can Service Providers (SPs) customize the conditions that their system uses to determine the trustworthiness of a transaction? Each SP has its own perception of trust, also depending on the concrete use case. It is thus vital to support heterogeneous trust models instead of relying on a pre-defined perception of a single trust scheme. Yet, the customization of source code is expensive, and the formalization of rules is often hard for non-technical domain experts. In Chapter 7, we aim to enable SPs to formulate rules easily, and to configure the system's trust rules to their needs and local regulations.

- How can we ensure the extensibility of trust verification systems to make them future-proof for novel technologies and models? For example, how can users and SPs utilize decentralized technologies to improve their privacy and interact in a trustworthy way? In addition to established trust and identity models, we discuss the integration of novel models like Self-Sovereign Identity (SSI) and Distributed Ledger (DL)-based trust management. By doing so, in Chapter 7 we enable SPs to base their trust decisions on both established as well as novel models in the same transaction. Additionally, in Section 6.6 we enable users to manage their own identity information or publish trust information about each other.

- How can users stay in control of their data while authenticating with a service provider? While a SP might need access to the user's

identity data to make an access decision, users often reveal more data than the SP needs. We discuss how existing access control systems can be extended with privacy features. In Chapter 9, we use the (existing) custom trust rules formulated by SPs, and connect them with Privacy-enhancing Technologys (PETs). This enables users to automatically only share the subset of data that the SP needs to process their access request. Additionally, in Chapter 10, we avoid information about the user's behavior being leaked during an authentication process.

To work towards the stated objectives, rather than providing a single solution, we provide building blocks and insights that can be adapted by existing systems. From these overall objectives, we derive the four research goals of this thesis, which we present in Chapter 5 below.

1.2. Contributions

This thesis groups my contributions into two Research Areas: Research Area 1 (RA1) focuses on the trust management aspects, while Research Area 2 (RA2) focuses on privacy aspects.

RA1 first presents results I developed as part of the LIGHTest project. More specifically, I am one of the designers of, and the lead developer of, the described automated trust verification system, as well as a co-designer of the policy system TPL. In addition, I am one of the lead authors of the three scientific contributions mentioned in this thesis, which originated from this project. The rest of RA1 builds on my work by extending upon the work done in the LIGHTest project by also considering the novel trust management method SSI. RA1 also considers a more open trust management architecture in addition to the hierarchical eIDAS architecture and the SSI model, and discussed credential format interoperability. This research resulted in three additional publications.

In **RA2,** I contribute to questions of how to extend the users' privacy in the (sub-)field of identity management. First, RA2 presents my work in extending existing access control systems with privacy features. Second, RA2 also considers the issue of undetectability during credential showing, specifically for DL-based data. This research resulted in two publications.

1.2.1. Main publications

The research on the topics of this thesis resulted in the publication of eight peer-reviewed papers accepted at international conferences. The author of this thesis is the lead or one of the main authors of all these publications. See Appendix A for more detailed contribution statements. Additionally, some of the results are also implemented and evaluated during three Horizon 2020 research and innovation projects (LIGHTest, KRAKEN, Eratosthenes).

Wagner G, Wagner S, More S & Hoffmann M, **DNS-based trust scheme publication and discovery**. In Open Identity Summit 2019 [Wag+19].

More S, **Trust Scheme Interoperability: Connecting Heterogeneous Trust Schemes**. In ARES SECPID 2023 [Mor23].

Mödersheim S, Schlichtkrull A, Wagner G, More S & Alber L, **TPL: A Trust Policy Language**. In IFIPTM 2019 [Möd+19].

Alber L, More S, Mödersheim S & Schlichtkrull A, **Adapting the TPL Trust Policy Language for a Self-Sovereign Identity World**. In Open Identity Summit 2021 [Alb+21].

More S, Grassberger P, Hörandner F, Abraham A & Klausner LD, **Trust Me If You Can: Trusted Transformation Between (JSON) Schemas to Support Global Authentication of Education Credentials**. In IFIP SEC 2021 [Mor+21].

More S & Alber L, **YOU SHALL NOT COMPUTE on my Data: Access Policies for Privacy-Preserving Data Marketplaces and an Implementation for a Distributed Market using MPC**. In ARES SECPID 2022 [MA22].

More S, Ramacher S, Alber L & Herzl M, **Extending Expressive Access Policies with Privacy Features**. In TrustCom 2022 [Mor+22].

More S, Heher J & Walluschek C, **Offline-verifiable Data from Distributed Ledger-based Registries**. In SECRYPT 2022 [MHW22].

1.2.2. Other publications

In addition to the publications listed above, the author is a contributor to the following works:

- Several project deliverables in the Horizon 2020 projects LIGHTest, KRAKEN, and Eratosthenes.
- Koch K, Krenn S, Marc T, More S, & Ramacher S, A Privacy-Preserving Data Market for Authentic Data. In CoNEXT 2022 [Koc+22].
- Gabrielli S, Rizzi S, Mayora O, More S, Perez Baun JC & Vandevelde W, Multidimensional study on users' evaluation of the KRAKEN personal data sharing platform. In Applied Sciences 2022 [Gab+22].
- Abraham A, Koch K, More S, Ramacher S & Stopar M, Privacy-Preserving eID Derivation to Self-Sovereign Identity Systems with Offline Revocation. In TrustCom 2021 [Abr+21].
- Abraham A, Schinnerl C & More S, SSI Strong Authentication using a Mobile-Phone based Identity Wallet reaching a High Level of Assurance. In SECRYPT 2021 [ASM21].
- Abraham A, More S, Rabensteiner C & Hörandner F, Revocable and Offline-Verifiable Self-Sovereign Identities. In TrustCom 2020 [Abr+20].
- Alber L, More S & Ramacher S, Short-Lived Forward-Secure Delegation for TLS. In CCS CCSW 2020 [AMR20].
- Omolola O, More S, Fasllija E, Wagner G & Alber L, Policy-based access control for the IoT and Smart Cities. In Open Identity Summit 2019 [Omo+19].
- Gruss D, Schwarz M, Wübbeling M, Guggi S, Malderle T, More S & Lipp M, Use-after-FreeMail: Generalizing the use-after-free problem and applying it to email services. In ASIACCS 2018 [Gru+18].
- Wagner G, Omolola O & More S, Harmonizing Delegation Data Formats. In Open Identity Summit 2017 [WOM17].

1.3. Thesis Outline

The two research areas introduced in Section 1.2 guide the structure of this thesis, as visualized in Figure 1.1.

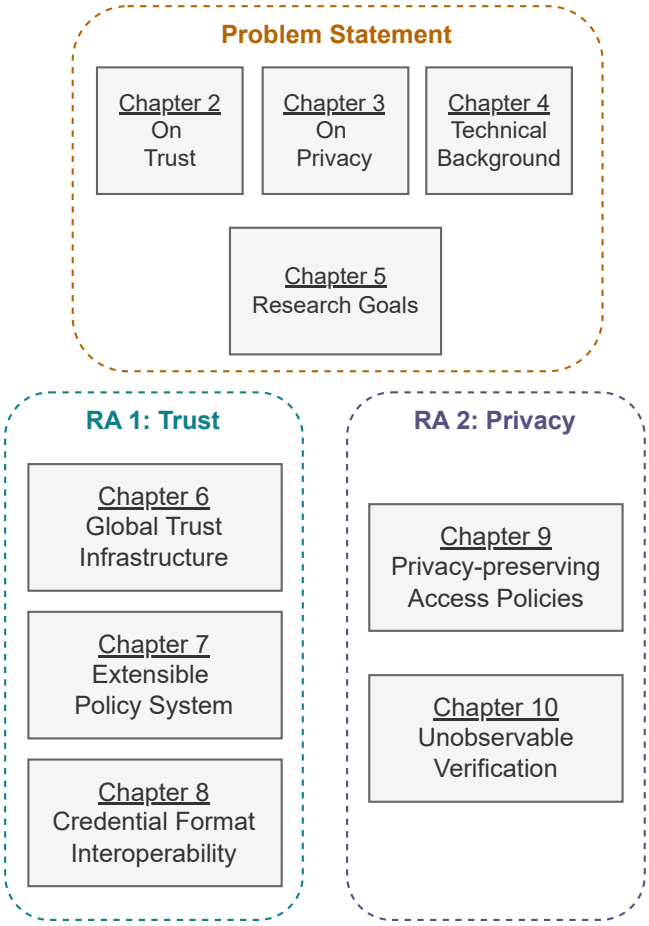

Figure 1.1.: Contribution structure of this thesis.

Part 1: The rest of Part 1 introduces the background of this thesis and states the motivation for our research. In specific, we first discuss the subjects of trust (Chapter 2) and privacy (Chapter 3) in more detail. We then give an overview of the technical building blocks used in this thesis

(Chapter 4). Finally, we introduce the overall research goals of our thesis and define the research gaps we address (Chapter 5). The research goals then further serve as a basis for the evaluation of our research, which we conduct individually in each subsequent chapter.

Part 2 presents our contributions in the research area of trust. In Chapter 6, we introduce our DNS-based trust infrastructure, which we use to facilitate trust management in a heterogeneous context. In specific, we focus on a global environment, hence we also discuss interoperability between trust schemes. Additionally, we present an alternative approach to centralized trust management architectures, operated on a DL. In Chapter 7, we discuss our access- and trust-policy system "TPL". TPL enables SPs to verify transactions based on their own perception of trust, using existing and novel trust schemes. In this chapter, we also present a case study of the TPL system applied to a distributed online marketplace. Finally, in Chapter 8, we introduce a trustworthy interoperability approach for electronic credentials and their encoding formats.

Part 3 presents and discusses our contributions in the research area of privacy. In Chapter 9, we discuss how to enrich an existing access control policy system with privacy features. Finally, in Chapter 10, we discuss our approach for a verification of undetectable credential verification; we focus on credentials and other data issued from a DL-based registry, hence building on the DL's distributed trust model [ATK18].

2

On Trust

One of the two pillars of this thesis is *Trust*. In this chapter we explore the concept of trust and related concepts, with a specific focus on *trust management*. We begin by defining trust and introduce trust management in the context of this thesis. First, we cover the fundamental architecture of trust management, by introducing the important actors and their relationships. Next, we give an overview of the two common trust management models, and discuss the topics of revocation and trust policies. We conclude the chapter by discussing the use of distributed ledger technology as a way to store trust information in a decentralized manner.

Figure 2.1.: Trust means making yourself vulnerable [Inm16].

Trust is a complex and multi-faceted concept that is central to many aspects of human interaction. Hawley defines trust as "*the reliance on some person or institution to meet their commitments*" [Haw12, pp. 4-6]. It is often a foundation for cooperation and collaboration, as it allows

individuals and organizations to engage in transactions, relationships, and activities with a reduced fear of betrayal or harm. From an organizational research perspective, trust (and trustworthiness) is reducing the cost of transactions between organizations [DC03]. The concept of trust comes in when it is not possible to be absolutely sure about something, so we need to rely on other entities to say the truth. This leaves open the possibility of the trustee not living up to their commitment, for example by unreliability, malice, or incompetence [Blo81]. If something is absolutely clear, it is not a matter of trust.

In a trust relationship, the one putting trust in another entity is the *trustor*, and the entity being trusted is called *trustee* [JKD05]. Thus, a trustor is relying on some trustee to meet their commitments. There is also mechanical trust in things such as that a chair, curtains or car. However, this thesis is focused on the richer interpersonal trust that comes with a moral overtone [Haw12, p. 6].

If there is trust, there can also be distrust [Haw12, p. 8]. We usually place distrust in people with whom we had some form of negative experience, or their reputation tells us so. In contrast, a *lack of trust* is not the same as distrust. A lack of trust is simply the absence of factors that would motivate trust—it is not the presence of factors that motivate distrust.

We often assess the trustworthiness of someone, even if not always doing consciously. Doing so involves our expectations in both their competence as well as their intentions [Haw12, p. 6]. There are various factors we can consider to assess both expectations. For example, what other people say about them (reputation), our previous experience with them (track record), as well as their incentives. Additionally, we might require proof to assess their credibility, such as formal credentials and certifications. Both competence and good intentions are required for someone to be considered trustworthy.

While absolute trust also exists, this is rare situation, and so we usually trust someone only in a specific context or to a different degree [Haw12, p. 6]. For example, we trust our doctor in medical questions, but we are not trusting them to repair our bike. Additionally, we might trust someone up to a certain degree, depending on the consequences in case the trustee is letting us down.

To further differentiate trust, we separate *direct trust* and *indirect trust* [BKW13, p. 39]. Way say trust is direct if we are directly interacting with the entity which we trust. For example, we trust our friend Reinhard to

repair our bike because we know he is skilled in doing so. Some form of direct trust is always the starting point, and it is a requirement for any model of trust.

In contrast to direct trust, we say trust is indirect if we don't directly trust the entity we are interacting with [BKW13, p. 48]. Instead, a (trusted) third party is establishing that trust relationship for us. For example, when we bring our bike to our entrusted bike shop for repair, we indirectly trust their mechanic Mike to be a skilled craftsperson. It is important to note that indirect trust is never absolute, instead it is only valid in some specific context. In our example, we trust the bike shop to assess the skills of their bike mechanics, but that gives us no information about the trustworthiness of Mike in other matters. Additionally, we would also not trust Mike to assess the skills of another mechanic for us. An exception to this is if the shop's manager explicitly told us that we can trust Mike to do so, and we trust the manager's assessment about that fact. This additional level of indirection is common when dealing with trust problems in computer science [Per99].

Identity Since we always trust in something or someone, a concept related to trust is *Identity*. Identity is "*the distinguishing character or personality of an individual*" [Mera], but also of other entities, and even objects. More plainly spoken, a person's identity is "*their name and other facts about who they are*" [Camb]. Identity information is often encoded in the form of a set of identity attributes. This attributes are key-value pairs that describe the characteristics of a subject [Abr22].

> Identity is any subset of attribute values of an individual person which sufficiently identifies this individual person within any set of persons. So usually there is no such thing as "the identity", but several of them [PH10, p. 30].

In contrast, when working with identities in computers science, we use an abstraction of an identity called *Identifier*. An identifier is some information used to identify an entity in a particular context [ITU09]. An identifier is *"a set of numbers, letters, or symbols [...] that is used in a system to represent someone or something* [Cama].

2.1. Trust in Computer Science

In the field of computer science, cryptography is used to enable security and privacy.

To ensure authenticity and integrity of some data, cryptography can be used to create a digital signature.[1] These signatures enable use cases such as private email communication, trustworthy web browsing and secure boot.

A cryptographic signature is a mathematical construct that uses two pieces of information. Those pieces of information are a bunch of numbers, and are commonly called public key and private key. The private key is personal and private to some entity. This private key is used to "sign" some data, which results in a "digital signature", also a set of numbers. This signature can be verified using the corresponding public key. If the assumptions of modern cryptography, that a malicious entity (the adversary) is computationally limited, and that the laws of physics and mathematics (which are commendable[2]), are correct, only the private key can be used to create a signature which successfully validates when using the corresponding public key. Thus, if the verification succeeds, the signature is valid. In that case the verifier can be sure that the signature was created by the entity in possession of the private key.

Cryptographic signatures can only be used in practice if users can trust the authenticity of public keys [BKW13, p. 39]. The remaining question is thus whether the verifier can trust that the public key actually belongs to the entity they assume it does. Ensuring this is one of the main questions of trust management.

Security is "*the quality of being free from danger, threat, or fear*" [Merb]. Security concerns are prominent in many domains, like political, physical, or when working with computers. Examples for political security are public security (e.g., emergency services and police) and international security (i.e., diplomacy and military). In the physical realm, examples are food security, environment security, and airport security. In the context

[1] Other methods to ensure the authenticity/integrity of data involve concepts like Message Authentication Codes (MSCs), Non-Interactive Zero-Knowledge Proofs (NIZKs), and hash functions.

[2] `https://zdnet.com/article/the-laws-of-australia-will-trump-the-laws-of-mathematics-turnbull/`, accessed on 2023-01-19

of computer technology, security deals with the security of computers and communication over networks.

Security in computer science is focused on "information security", which describes "*ways of protecting information, especially electronic data, from being used or seen without permission*" [Oxf]. Concepts important for information security are:

- Confidentiality
- Integrity
- Availability
- Authentication, Access Control
- Accountability and Non-Repudiation

2.2. Trust Management

A common question in computer security is whether a document was really issued by a specific person and not altered during transmission over the Internet. While digital signatures help with the detection of (potentially malicious) changes to the document, it requires more work to authenticate the signer of the document. On a technical level this comes down to ensuring the correct binding of a (cryptographic) key to some identity—often the legal identity of some person.

To explore the technical aspects underlying the automation of trust processes in a more systematic way, in 1996 Blaze et al. introduced *trust management* as a field of research [BFL96]. Trust management deals with "a need for methodologies that enable [...] parties to determine the trustworthiness of remote parties through computer mediated communication and collaboration" [JKD05]. The field is concerned with all problems involved in the formalization and encoding of trust relationships, the formulation of trust policies, the representation of credentials, and answering whether to trust a credential or third party. The field is thereby concerned with protocols to "transfer trust from where it exists to where it is needed" [SM95].

In contrast to credential-based trust management, reputation-based trust management [ZM00; Res+00; JIB07] is concerned with subject like product ratings and "customer reviews" [ST05; Haw12, p. 83]. I.e., in that case,

someone's trustworthiness is assessed on what others say about them, and not from the statement of a qualified authority. While this is an important aspect of the field, it is out of scope of this thesis. It is to be noted though, that the factor of reputation also plays a role in non-reputation based trust management systems. For example, a user might chose a webbrowser based on its reputation. This decision then defines the list of trusted certificate authorities, which is shipped with the webbrowser. This list in turn influences what websites the user can open in a secure manner. In the use case of legal electronic identities, this is not the case, since the list of certificate authorities that are authorized (qualified) to issue such identities is defined by a government (and regulated by a law), which is again a different kind of trust management.

Trust Management Actors In a typical trust management architecture, the involved actors fulfill one or more of these roles [CY10]:

- **Issuer**: The entity that issues some digital information. To do so, an issuer takes some data and uses their private key to sign the data. This is role is often fulfilled by an *Identity Provider (IDP)*. In that case the data that is issued is some form of identity information.
- **Holder** (or User): The entity that the digital information is issued to. The holder receives (signed) data from an issuer and later presents it to some verifier. The issued information is often about the holder, so the holder is also the *Subject* of some credential or certificate.
- **Verifier** (or Service Provider): The entity that receives some signed data and is interested in its trustworthiness. To ensure that the received data is trustworthy, the verifier checks the digital signature of the data. Additionally, the verifier checks if the data was signed by an issuer they trust. This is done using trust management methods.
- **Authority**: In use cases on a larger (e.g., global) scale, direct trust of a verifier in an issuer is often not realistic. To increase the practicability of systems, there is thus a need for a layer between those two parties. A common solution to this is the introduction of one or more authorities that facilitate the trust between verifier and issuer. An example for this is the Public Key Infrastructure (PKI) used to authenticate HTTPS connections on the web. In this scenario, a webserver is presenting some certificate to a webbrowser to prove that it is indeed the correct server for a certain hostname. Thereby, the webbrowser serves in the role of a user, and the webbrowser is

the verifier (of the certificate). While the certificate has been issued by some certificate authority, the webbrowser does not trust that authority directly. Instead the webbrowser trusts a set of (root) authorities that in turn delegate their trust to other authorities, one of them issued the webserver's certificate. This delegation scheme, also possible over multiple levels, is a common construct in trust management.

In general, there are the following relationships between those roles, also shown in Figure 2.2:

- Issuer *issues data to* Holder
- Holder *presents data to* Verifier
- Verifier *trusts* Issuer (or Authority)
- Authority *authorizes* Issuer

Depending on the context, one entity can have multiple roles. For example, a university is a verifier of identity documents and diplomas of incoming students. At the same time, the university is also the issuer of diplomas.

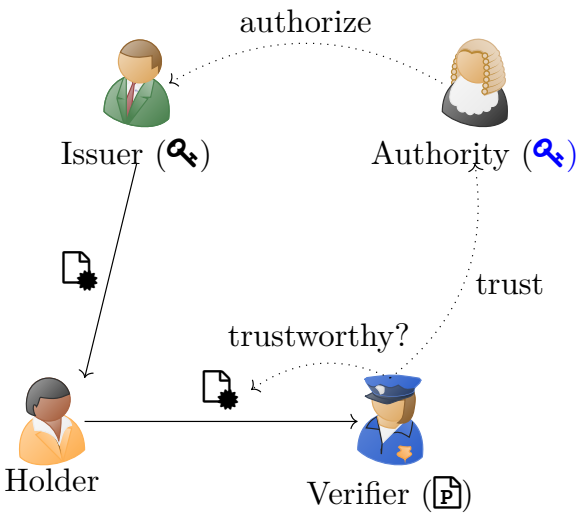

Figure 2.2.: Common roles in trust management and their relationships

Trust Management Models There are various trust management models to ensure the binding between key and identity and to support the discovery of keys [Per99, p. 1].

Buchmann et al. differentiate trust models into *direct* and *indirect* trust models [BKW13]. In the direct trust model, a user establishes trust in some information directly, e.g., by receiving it from the origin using an trustworthy communication channel. In contrast, indirect trust models use one or multiple intermediaries (trusted third parties) to establish trust. Perlman further differentiates indirect trust models into different categories [Per99]. For example:

- **Anarchy** is a trust management models where trust is distributed among multiple parties, rather than being managed by a central authority. Each user starts of with a different set of (directly) trusted entities. For example, the Web of Trust (WoT) model relies on users to vouch for the authenticity of other users' digital certificates in a peer-to-peer way. Users form trust relationships with each other, and these trust relationships are used to determine which certificates should be accepted as valid. A prominent instance of a WoT is PGP [Zim95], which uses a web of trust model to establish trust relationships between users to authenticate email messages and software downloads.
- **Configured CAs with Delegations** is a hierarchical trust management model where a set of central authorities is responsible for managing trust relationships. This set of authorities is pre-configured by the used software, and usually all users of a system agree on the same set of configured authorities. Additionally, this configured (root) authorities can authorize other authorities to act as authorities. In contrast to peer to peer models like WoT, such a model often allows for (legal) liability of authorities [BKW13, p. 48]. A widely used model for managing digital certificates and public keys is PKI. A central authority, called a Certificate Authority (CA), is responsible for issuing and revoking digital certificates. A set of CAs is configured by the user as trustworthy, thus the certificates issued by them can be used to authenticate entities and thus securely encrypt communications. In practice, the trusted CAs don't directly issue certificates. Instead, they authorize other CAs to issue certificates, which in turn can authorize other CAs to do so, forming a chain of delegations [Per99, p. 2]. Only the CAs at the end of this chain sign certificates used to authenticate entities. This form

of PKI increases the practicability of PKI. An example for a PKI of this type is the X.509 certificate system used by the Transport Layer Security (TLS) system for the secure communication between webservers and webbrowsers [Coo+08]. Another example of PKI is the trust infrastructure established by the European Union's (EU) electronic Identification, Authentication and Trust Services (eIDAS) regulation [Eur14]. We discuss the technical aspects of the eIDAS trust service infrastructure in Section 4.1 below.

- **Top-Down** is another hierarchical trust management model. In contrast to the CA model described above, there is usually only a single CA at the root of the hierarchy. Another difference is that there is a hierarchical namespace: a CA is only authorized to certify mappings for names in their subtree of the namespace [Per99, p. 4]. This model is for example used in the DNS Security Extensions (DNSSEC) system to protect the integrity of the DNS system. For example, a CA trusted for domains in the "at." Domain Name System (DNS) namespace (zone) can issue certifications for names like "iaik.at.", but not for, e.g., "acm.org.". Additionally, the CA can delegate a subtree (subzone), e.g., "gv.at.", to another CA, which can then issue certifications for "help.gv.at.", but not for "iaik.at." or "acm.org.".

We note that some models fit into multiple categories, as for example a WoT can be used to build a (hierarchical) PKI.

Direct trust serves as the basis for indirect trust models [Prü16, p. 12]. For example, to use a PKI, the user first needs to retrieve the set of trusted (root) CA certificates. Commonly this is done during the setup of a computer, e.g., as part of the installation of the operating system. Generally, "we say that trust in the authenticity of a public key is direct if the public key is directly obtained from the key owner or its owner directly confirms the authenticity of the key in a way that is convincing for the user" [BKW13, p. 39].

Credentials and Certificates To represent identity information and trust relationships, digital credentials and certificates are used.

A digital credential is a signed data structure which contains a set of attributes. To securely attest the attributes, the credential is signed by some issuer using a cryptographic signature scheme. For the credential to be accepted by a verifier, this issuer needs to be trusted by that verifier.

Examples for credentials are digital versions of their analog counterparts, like university diplomas, or a drivers license. A credential format common in the Self-Sovereign Identity (SSI) world is the Verifiable Credentials Data Model [SLC22].

In contrast, a digital certificate [Coo+08] is used the bind the identity of a subject to a cryptographic (public) key.[3] For example, in the TLS use case, a certificate contains the hostname (the identity) of the server alongside of a set of metadata (like validity and context information). Additionally, the certificate contains the public key of the server. This key is used by the webbrowser to authenticate the connection with the webserver. The issuer that signs the certificate data structure is a certificate authority which is trusted by the user's webbrowser. Another use case of certificates are digital signatures. In this use case, an issuer issues a certificate to a user which binds the user's public key to their identity. The user then uses their corresponding private key to sign a digital document.

Revocation Revocation refers to the process of invalidating some attested information, or a trust relationship that has been established between two entities. This can be done for various reasons, such as when a user's role changes, when a device is lost or stolen, or when a digital certificate is compromised.

The type of revocation depends on the information that is to be revoked. In case of digital certificates and other forms of electronic transactions, a common way to revoke those is by publishing a list of all revoked data; for example in the form of a Certificate Revocation List (CRL). This list is usually published by the issuer of the information, or a entity authorized by that issuer. Another way for a user to prove that their credential is still valid is to directly retrieve a attestation about its validity from the issuer, and prove that (fresh) attestation alongside the data to the verifier; in TLS this is done using the Online Certificate Status Protocol (OCSP).

Another example for revocations is the revoking of a trust relationship that has been published directly in the form of trust statements in some registry. A verifier then checks the presence of this statement before trusting some information. To revoke such a statement, the issuer of that

[3] Since credentials can contain key material to prove the ownership of some credential, and certificates can contain extensive identity information, the two concepts sometimes overlap.

statement simply alters the trust level of the statement, or removes it from the registry entirely.

Trust- and Access-Policies Policies, especially trust policies, enable the automated processing of electronic transactions. They enable users to define their own conditions for trusting a transaction and mechanics for discovering trust information needed to authenticate signers.

Trust policies are sets of rules and procedures that define how trust in an electronic transaction is established. Policies can also be used to specify the level of trust that is required for different types of transactions. For example, a trust policy may require a higher level of trust for a financial transaction than for a simple message exchange. Organizations can use these policies to define if and how they want to rely on existing trust schemes like Europe's eIDAS to add legal value to the decision.

In addition to trust rules, policies can also include rules for authentication, authorization, and access control. For example, a trust policy may specify that a user must provide a valid digital credential containing specific attributes, before being granted access to a resource. Or a policy can also include rules for ensuring the validity of a digital credential and that it is not revoked.

Since trust policies usually encode business rules, they are often authored by domain experts.

Chapter 2 Conclusions

This chapter explored the multifaceted concept of trust and its significance in computer science and beyond, laying the groundwork for our research on automated trust management. We defined trust management in the context of this thesis and gave an overview of the related fields of identity and security. Further, we discussed essential concepts like revocation, trust- and access-policies, and encoding attributes in the form of credentials.

Automating trust management is often complicated, because no direct trust relationship exists between the user and the verifier. To instead facilitate indirect trust relationships, we present the four common roles in trust management (issuer, holder, verifier, and authority) and the relationships between them. Those roles and their relationships are fundamental to our thesis. On the one hand, the entities executing those roles use protocols and tools to automate them. On the other hand, the relationships between the entities form a trust path, the basis that a verifier can establish trust in an electronic transaction they receive. We also discussed trust models used in practice to organize this trust path. These trust models and their technical implementation are the first focus of our thesis.

After discussing the concepts related to trust, we now turn our focus to the second topic of this thesis—the idea of privacy.

3

On Privacy

The second pillar of this thesis is *Privacy*. In this chapter we introduce the concept of privacy and discuss its characteristics. Then, we discuss the relevance of privacy for computer science. We conclude the chapter by defining several terms related to privacy in the context of information technology.

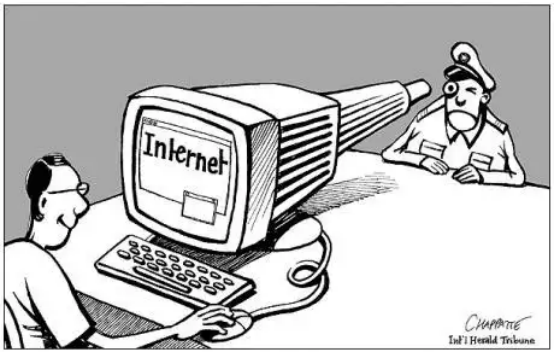

Figure 3.1.: Big brother is watching [Cha05].

Privacy The word *privacy* stems from the Latin word "Privatus", which in Roman law means "what is private", personal and belonging to oneself, and not someone else. In the historical context, this "someone else" was "the government", but our understanding of privacy certainly got more nuanced over time. Today, privacy refers to many aspects of a person's life. When we want to talk to someone in private, we ask for privacy—and we expect that no one is listening when we talk to a group of friends in private. When someone is following us in the streets, or wiretapping to a phone conversation, they are interfering with our privacy. The same is

often the case with pervasive CCTV cameras or tracking cookies on the Internet. The European Convention on Human Rights provides a "*right to respect for his private and family life, his home and his correspondence*".[1] Modern data protection regulations (privacy laws) are concerned with how governments and companies collect personal data, and how they are allowed to use it.

Privacy is "*the right that someone has to keep their personal life or personal information secret or known only to a small group of people*" [Camc]. According to the ubiquitous definition of U.S. legal scholar Westin, (information) privacy is "*the claim of individuals, groups, or institutions to determine for themselves when, how, and to what extent [any] information about them is communicated to others*" [Wes67, p. 7]. Both definitions allow for a broad definition of privacy. This differentiates privacy in a "informational privacy" and a "decisional privacy" [Wac15, p. xiv]. This is also reflected by the German translation of the word "Privacy", which is both "Privatsphäre" and "Privatheit" (privateness). The term privateness as a right means a "right to a private life",[1] and "the right to be let alone" [BW90], free from government interference. Decisional privacy also involves the right to an individuals self-determination, like the right to contraception and abortion.

While this privateness aspects of privacy are also important, in the context of this thesis, we are concerned with the information related aspects of privacy—information privacy. In particular, "*information privacy relates to an individual's right to determine how, when, and to what extent information about the self will be released to another person or to an organization*" [HC09]. Thus, privacy concerns itself with the protection of sensitive personal information [Wac15, p. 2; OBK23].

Characteristics of Privacy Legal scholar Wacks characterizes privacy as concerned with "limiting accessibility" to an individual in three components: limiting information about someone (secrecy), limiting the attention paid to someone (anonymity), and limiting the physical access to someone (solitude) [Wac15, p. 44]. Nevertheless, information privacy, which is the focus of this thesis, is concerned only with the first two components.

What is *not* privacy? A conversation can be private, as much as data or information can be private. But, privacy is only concerned with personal

[1] Convention for the Protection of Human Rights and Fundamental Freedoms, Article 8 – Right to respect for private and family life: https://rm.coe.int/1680a2353d

information, and not other types of sensitive or secret information.[2] For example, from the General Data Protection Regulation's (GDPR) point of view [Eur16, Article 1], secret information belonging to an organization is not considered private.[3] Similar, under the US Privacy Act, corporations do not have privacy rights.[4]

Personal Information (and Personal Identifiable Information) In the digital age, privacy is about questions that "*share a concern to limit the extent to which private facts about the individual are respectively published, intruded upon, or misused*" [Wac15, p. 46]. What are private facts, and what is to be understood by "personal information", depends on culture-specific norms, and conceptions vary over time. Additionally, the extent of what is personal depends on the context. Thus, the understanding of "personal information" refers to both the quality of the information, as well as the desire to control its use (in a specific context) [Wac15, p. 47]. Personal information are all facts relating to a individual "*which it would be reasonable to expect them to regard as intimate or sensitive, and therefore to want to withhold, or at least to restrict their collection, use, or circulation*" [Wac15, p. 48].

Personal data/information also plays an important role in various data protection regulations. For example, Article 4 of the European Union's (EU) GDPR[5] defines "personal data":

> 'Personal data' means any information relating to an identified or identifiable natural person ('data subject'); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person.

[2] If it allows the identification of a natural person, this data is obviously personal data.

[3] https://commission.europa.eu/law/law-topic/data-protection/reform/rules-business-and-organisations/application-regulation/do-data-protection-rules-apply-data-about-company_en, accessed on 2023-03-20

[4] The Privacy Act of 1974 in its definition of "Individual" states "Corporations and organizations also do not have any Privacy Act rights" (https://www.justice.gov/opcl/overview-privacy-act-1974-2020-edition/definitions#individual). But, the Freedom of Information Act, which has an exception for the "invasion of personal privacy", defines "person" in a broader way, cf. https://en.wikipedia.org/wiki/FCC_v._AT&T_Inc., accessed on 2023-03-20

[5] https://gdpr.eu/article-4-definitions, accessed on 2023-01-25

In U.S. law, the term Personally Identifiable Information (PII) is used. PII is defined as all data that can on its own be used to uniquely identify someone. The U.S. Department of Labour defines PII as "*Any representation of information that permits the identity of an individual to whom the information applies to be reasonably inferred by either direct or indirect means*".[6]

It is to be noted that the term personal data covers a wider range of data than personal identifiable information. PII is a subset of personal data, thus, all PII is personal data, but not all personal data is PII. For example, in European law, a device ID or IP address is considered personal data.[7] But, in some court rulings, IP addresses are not considered PII.[8]

Privacy and Security: Ensuring security is both often in conflict with protecting privacy, but also a requirement for privacy. On the one hand, there is a tradeoff between security and privacy. Governments often defend surveillance by arguing that we need to sacrifice some privacy for security [Sol11]. Balancing this tradeoff involves adequate oversight and regulation. On the other hand, the protection of privacy always requires security as a basis [HC09]. A system which is not secure can never ensure the users privacy. This is both true on a societal as on a technical level.

3.1. Privacy in Computer Science

With the rise of computers and threats on privacy, the protection of privacy is of increasing importance and interest. A reason for this is that computers and the Internet are becoming increasingly ubiquitous [RS13, p. 4]. Even before the Internet, computers were used by professionals and everyday citizens for a multitude of tasks, from office work to entertainment. But the increasing interconnection of computers brought this to a new level, and we are now using computers and the Internet throughout the day for all kinds of activities. By doing so, we not only need to entrust various

[6] https://www.dol.gov/general/ppii, accessed on 2023-01-25

[7] cf. Breyer v. Germany; and GDPR Recital 30: https://gdpr.eu/recital-30-online-identifiers-for-profiling-and-identification, accessed on 2023-01-25

[8] cf. 2009 court ruling in Johnson v. Microsoft Corp, the judge found that "for 'personally identifiable information' to be personally identifiable, it must identify a person. But an IP address identifies a computer": https://www.huntonprivacyblog.com/2009/07/10/washington-court-rules-that-ip-addresses-are-not-personally-identifiable-information/, accessed on 2023-01-25

parties with our personal data, but we also make it easier for curious third parties to learn more about us, as behavior in the digital realm is easier to track than in the physical world.

Also, computers themselves are a tool to impede privacy. Advances in information processing and communications make it possible to collect, process and store a unthinkable amount of information from the digital and physical world. Once collected, such information is rarely erased and it is almost impossible to fully control its spread and use. This raises privacy problems [Gal+11]. If financial resources are available, these technologies can be scaled to fit almost any need,[9] which increases the threats on privacy. The digital processing of all kinds of information facilitated by computers enables forms of data analysis and surveillance not possible before. While it was possible to wiretap a telephone conversation or tail a person before, this was a manual effort, and so was analyzing the data gathered by this means. Today, digital communication can be collected by automated systems, stored in large quantities over a long duration, and analyzed with dizzying speed. A term describing this is *big data*, referred to "*things one can do at a large scale that cannot be done at a smaller one, to extract new insights or create new forms of value [...]*" [MC13]. Additionally, digital data processing enables analyzing different types of data from many sources. For example, movement data acquired by tracking a cellphone, and the purchase history retrieved from the bank could be combined to derive deeper insights into a person's behavior.

The protection of privacy requires more than the encryption of some data. For example, for an observer it might not be necessary to see the content of a user's communication to learn something about the user. The mere fact that a communication between two users happened may represent sensitive information. Thus, privacy is also concerned with the protection of metadata.

[9] e.g., the U.S. intelligence community operates a datacenter with a storage capacity estimated to be around 3-12 exabytes as of 2013: `https://en.wikipedia.org/wiki/Utah_Data_Center`, accessed on 2023-01-25

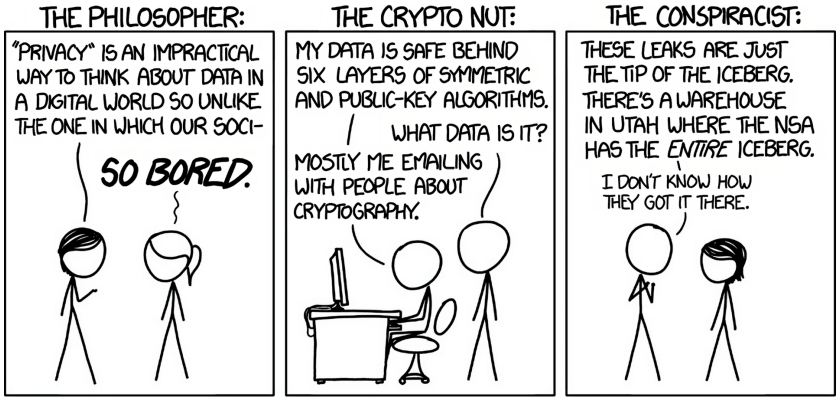

Figure 3.2.: Opinions on Internet Privacy [Mun13].

3.2. Privacy Concepts

In computer science, important characteristics of privacy are:

Anonymity means that a user can access a resource while remaining "*not identifiable within a set of subjects, the anonymity set*" [PH10, p. 9]. The anonymity set is the set of all users who this access to the resource could be plausibly attributed to. Thus, the owner of that resource does not learn which of the possible users just accessed the resource.

Anonymity is crucial to protect users from being tracked or profiled, and to maintain their privacy.

Pseudonymity: Sometimes full anonymity is not productive, i.e., when a service provider wants to link multiple requests by the same user, but is not interested in learning their identity. In that case a pseudonym is used to identify the user. "*A pseudonym is an identifier of a subject other than one of the subject's real [legal] names.* [...] *A subject is pseudonymous if a pseudonym is used as identifier instead of one of its real [legal] names*" [PH10, p. 21].

Unlinkability of two or more users means that "*an attacker cannot sufficiently distinguish whether these users are related or not*" [PH10, p. 12]. For example, if two messages were sent from users within the same

anonymity set, an observer cannot tell if the messages came from the same user or from different users. In a system where the user is unlinkable even from the perspective of the owner of the resource, the owner does not learn if two requests to a resource came from the same user.

Unlinkability ensures that different data sets, transactions or activities cannot be linked together to infer information about an individual, or to identify them. For example, if someone visits a bar two times and the barkeeper remembers them, their visits are linkable and thus more information can be derived than from a single visit. Unlinkability is essential to protect user's privacy, particularly in scenarios where personal information is collected and used.

Undetectability of an action from an observer's perspective means that the observer cannot sufficiently distinguish whether the action happened or not [PH10, p. 16]. For example, if a students shows their ID card to a bouncer, the government is not aware of that bar visit, meaning the visit is undetectable to them. But, when the student shows a digital credential, and the bouncer queries for the credential's validity, the government learns about this query. Since such a query often comes with contextual information like IP address or even location, the government becomes aware of the student's bar visit. Thus, the bar visit is not undetectable (from the perspective of the government). An example for undetectability is that a communication is undetectable if an observer watching some communication channel cannot distinguish the communication from random noise [PH10, p. 16].

Unobservability: In contrast to undetectability, *unobservability* means that while a party involved in a communication can obviously detect it, they do not know with whom they are communicating. For example, a user is unobservable, if they can retrieve some information from a government API without the government learning who just queried the API, and without any observer learning that someone executed a query. In other words, unobservability implies undetectability (to an observer) and anonymity (to the involved communication party) [PH10, p. 19].

Context privacy means that an observer should not be able to learn contextual information like location of a user unless intentionally disclosed [LR09]. It concerns the protection of sensible information that requires

to be guaranteed even if data content is fully secured utilizing traditional security mechanisms [Gal+11]. Contextual information can also be called metadata, thus context privacy is concerned with the protection of metadata in contrast to the message content.

Confidentiality is the characteristic of keeping information or data private and protected from unauthorized access or disclosure. It ensures that only authorized parties have access to private information. Confidentiality is important to protect individuals from having their personal information accessed, used or shared without their consent. Confidentiality is often achieved through the use of encryption, access control, and other security measures.

Chapter 3 Conclusions

This chapter discussed the concept of privacy and its paramount importance for computer science. Privacy—an individual's right to control information about the self—is a fundamental human right and is also enshrined by law in various legislation, like the EU's GDPR. These rights are constantly threatened by new achievements in computational power and digital surveillance capabilities. However, the digitization of everyday life also enables new forms to protect one's privacy—by using cryptography, or by cleverly designing systems.

As a foundation for our research, we provided an overview of crucial privacy concepts. These concepts form the basis of our research goals concerning privacy preservation in trust management systems. Specifically, when a user presents some credential to a verifier, this credential often contains more personal information than the user needs to reveal—violating the principle of confidentiality and data minimization. Further, when a verifier contacts an authority to verify a credential's validity, this reveals the user's behavior to the authority—violating the principle of undetectability. Mitigating these issues is the second focus of our thesis.

With this foundation laid, we now focus on the relevant technical concepts for our research.

4

Technical Background

In the previous chapters, we discussed trust, privacy, and related concepts. In this chapter, we introduce technical concepts used in the rest of this thesis.

4.1. Trust Schemes and Trust Status Lists

The value and meaningfulness of both a certificate and a credential depend on its issuer. A fundamental requirement for any trust in a credential or certificate is that the relevant verifier trusts the issuer of that credential. In use cases requiring information with legal value, the only relevant issuer is the government, or an entity authorized by the government. Such an issuer is thus *qualified* to issue certain information.

For example, if some qualified issuer issues a certificate to a user, and that user uses that qualified certificate to sign a PDF file, then this signature is called a *qualified signature*. A common approach to assess the legal value of an electronic transaction is thus to check if a qualified certificate was used to sign it.

"Trust schemes" are used to assess the legal value of a transaction. Trust schemes are the organizational, regulatory/legal, and technical measures to assert trust-relevant attributes about enrolled entities in a given trust domain [Wag+19]. We discuss trust schemes in more detail in Section 6.1 below.

For example, the European Union's (EU) eIDAS regulation introduces such a trust scheme for the EU member states. In eIDAS, a requirement for the status of qualified signature is that the certificate of the signer has been issued by a qualified Trust Service Provider (TSP). To ensure that a certain issuer is a qualified TSP, the verifier uses a Trusted List, or Trust (Service) Status List (TSL). The verifier retrieves this list, and checks if

the list contains the issuer as currently qualified TSP. Trusted lists contain all currently and previously qualified trust service providers. They are specified by the eIDAS regulation as public lists in Extensible Markup Language (XML) format following an European Telecommunications Standards Institute (ETSI) standard [ETS16].

In eIDAS, every European member state (MS) signs and publishes such a trusted list,[1] containing TSPs from their country. To discover and authenticate a member state Trust (Service) Status List (TSL), a verifier first needs to load the List of eIDAS Trusted Lists (LOTL), signed and published by the European Commission at a known location.[2] The LOTL contains pointers to the MS lists and the certificates used to sign them.

If an issuer is listed on such an MS list as active and Qualified Trust Service Provider (QTSP), it can provide qualified certificates, which can then be used to create qualified signatures. Qualified signatures have the same legal value as handwritten signatures in the EU.

4.2. Identity Management

A common application for trust management is the field of identity management. Identity management systems are responsible for issuing and maintaining digital identities, which can include verifying the identity of users through authentication methods such as digital credentials, and managing access to resources based on those identities. To verify whether a digital credential is trustworthy, methods from the field of trust management are used.

Fundamental concepts in identity management are identity and identifier, credentials and certificates (cf. Chapter 2).

X.509 and PKIX: The X.509 standard is a International Telecommunication Union (ITU) and International Organization for Standardization (ISO)/IEC standard for defining public key certificates [ITU88; ISO20]. X.509 also specifies certificate revocation lists (Certificate Revocation Lists (CRLs)), which are used to distribute information about certificates that have been deemed invalid. The X.509 standard is widely used in various

[1] The trusted list of Austria is published at `https://www.signatur.rtr.at/currenttl.xml`

[2] `https://ec.europa.eu/information_society/policy/esignature/trusted-list/tl-mp.xml`

applications, such as web security (Transport Layer Security (TLS)) and email encryption (S/MIME).

The Internet Engineering Task Force's (IETF) "Public-Key Infrastructure (X.509)" working group (PKIX) adapted the standard to the internet's needs.[3] Consequently, the term X.509 certificate often denotes the IETF's PKIX certificate and CRL profile [Coo+08].

Self-Sovereign Identity: Centralized trust infrastructures represent an attractive target for criminals and cyberwar [Fri20]. Additionally, centralizing the storage of identity data increases the likelihood of data breaches [Ber20; Ber17; Tho+17], resulting in a growing demand for more secure systems [ZS18]. In the last decade, the concept of Self-Sovereign Identity (SSI) gained popularity. Together with distributed ledgers, they aim to decentralize several aspects of trust and identity management.

SSI [Müh+18] is an identity model, which puts identity data back in the hands of users instead of storing them in data silos. The SSI community introduced several interesting concepts to represent (verifiable) credentials [SLC22] and (decentralized) identifiers [Ree+21; SSP23] in an interoperable, decentralized, and privacy-enabling way, supported by Distributed Ledgers (DLs).

In the SSI model, a user creates a **Decentralized Identifier (DID)** for their identity [Ree+21; SSP23] and publishes it by defining a DID document containing the DID, a corresponding public key, and other application-dependent information. That is often done using a DL.

The vision of SSI is that any party can certify attributes of any other party, e.g., a higher education institution can accredit graduation to a student, or an interior ministry can attest someone's date of birth. For such certifications, it is common to use the **W3C Verifiable Credentials (VCs)** data model [SLC22]. This specification defines a generic way of packaging claims and the corresponding issuing authority in a signed JavaScript Object Notation (JSON) document.

But, while those concepts enable or improve new use cases, they frequently build on under-specified or insecure standards and file formats [Hal20]. Additionally, they often lack scientific analysis. This is reinforced by a lack of liability in the governance models of used distributed ledgers.

[3] `https://datatracker.ietf.org/wg/pkix/about/`, accessed on 2023-03-14

Verifiable Credentials (VCs) are used as a typical data format in SSI systems and are specified by the World Wide Web Consortium (W3C) [SLC22]. The specification defines a generic envelope to represent a set of claims (*credentialSubject*) signed by its issuer. It defines a JSON structure for the credential envelope itself while allowing credential issuers to choose their own schema for the credential subject.

While the schema of the envelope is defined by the W3C, the claims inside a VC are encoded according to a JSON schema [WAH20] so that credentials are sufficiently flexible for various use cases. This flexibility of the content's schema also represents a challenge for verifiers when facing content in an unknown schema.

4.3. Distributed Ledgers

A DL is a distributed (and often decentralized) data storage model. The data is stored redundantly at several distributed nodes maintained by different entities, improving resilience. Each node preserves independent control and agrees on a common state by running a consensus protocol [Xia+20].

DLs (often deployed as blockchains) enable storage of data not only in a distributed way but also with decentralized governance. Access-wise, DLs support a large spectrum of models, mostly grouped and described by the terms public, private, permissionless, and permissioned [Zhe+18]. Proposed DL Public Key Infrastructure (PKI) solutions [Ale+17; Koa+] try to mitigate the weaknesses of conventional PKIs, focusing on issues like a single point of failure, vulnerability to split-world attacks, and the lack of identity retention [Koa+].

Distributed vs. Decentralized In contrast to systems running on a single computer, a system can also be distributed onto a collection of autonomous computers [VT17]. These computers are often distributed geographically in different data centers. But, whether a system is distributed says nothing about who controls the individual computers. While the distribution of a system enables the decentralization of its control, many systems are distributed but depend on central authorities. For example, (not only) large organizations like Google distribute their data centers all around

the world,[4] but nevertheless control them centrally. Conversely, Bitcoin's blockchain technology was introduced as a system without depending on central authorities [Nak08]. But, in general, whether a *distributed* ledger is also *decentralized* depends on its governance model.

Smart Contracts: Many ledgers support the storing of code on the DL, which is then deterministically executed by the nodes performing the consensus protocol.

Examples of this are the Ethereum ledger[5] and various ledgers from the Hyperledger project.[6] One particular example is Hyperledger Besu,[7] an Ethereum client specifically designed for use in permissioned consortium ledgers. Such code is called a "smart contract" (SC) [But+14] in the Ethereum world, while the Hyperledger project also calls it "chaincode" [Cac+16]. Variables in the code are stored on the DL as well and can be read and modified using functions supplied by the contract. SC code is written in a high-level language and then compiled to ledger-specific bytecode. Only this bytecode is then written to the DL. When a user sends a function call to a contract, nodes execute this bytecode, for example, using the Ethereum Virtual Machine (EVM).[8] The resulting state is only written to the ledger if all nodes agree on the result of the computation.

SCs can be used to provide simplified views on complex data stored on the DL, forming a generic stored-procedures-like query and filtering system akin to stored procedures in a traditional database.

Ethereum RPC API: To allow other entities to access the state of the DL and call SC functions, Ethereum nodes provide an HTTP API.[9] It offers a JSON-RPC interface, which can be used by entities who do not wish to operate a full node, or cannot participate in the ledger.

On the client-side, the web3.js[10] library is commonly used to interact with nodes' JSON-RPC API. It provides users a high-level interface to interact

[4] `https://www.google.com/about/datacenters/locations/`, accessed on 2023-03-14
[5] `https://docs.soliditylang.org`
[6] `https://hyperledger-fabric.readthedocs.io/en/release-2.2/chaincode4ade.html`
[7] `https://besu.hyperledger.org`
[8] `https://ethereum.org/en/developers/docs/evm`
[9] `https://eth.wiki/json-rpc/API`
[10] `https://github.com/ethereum/web3.js`

with SC functions and translates these function calls to a representation that the EVM understands.

InterPlanetary File System (IPFS) is a distributed file system based on an open peer-to-peer network [Pro21]. Similar to a public DL, there is no central party controlling the system, and everyone can join the network of nodes and retrieve data. As files on IPFS are content-addressed (i.e., a file is addressed based on a hash of its content), the files' integrity is ensured by design, which reduces the trust requirements towards the nodes. Hence, IPFS is an ideal data storage system for decentralized apps hosted on a DL.

4.4. DNS and DNSSEC

The Domain Name System (DNS) is a hierarchical distributed naming system that is commonly used for translating domain names to IP addresses [Moc87a; Moc87b]. DNS enables users to access websites and other internet resources using human-readable domain names instead of numeric IP addresses. The governance of the DNS is managed by the Internet Corporation for Assigned Names and Numbers (ICANN), which oversees the allocation of domain names and IP addresses to organizations and individuals around the world. ICANN works in collaboration with regional domain name registries, delegating the management of a Top-Level Domain (TLD). For example, EURid is responsible for operating the *.eu* TLD.

In addition to resolving IP addresses, the DNS can also be used to translate domain names into other identifiers. To do so, the DNS uses resource records to store information about domain names and their associated values [Moc87b]. Resource records are stored in DNS (name) servers, which are distributed around the world and are responsible for providing resource records to users.

The DNS Security Extensions (DNSSEC) is a set of specifications that are designed to enhance the security of the DNS system [Are+05a]. DNSSEC works by adding digital signatures to DNS resource records, which are used to ensure the integrity and authenticity of the information stored in the records. DNSSEC uses a key management system that is based on public-key cryptography. The public keys are distributed through the DNS system using resource records, which allows clients to verify the digital signatures on resource records and ensure their authenticity.

DNS supports several resource record types that are used to store information. For example, the URI resource record is used to store Uniform Resource Identifiers (URIs), e.g., an HTTP Uniform Resource Locator (URL). The PTR resource record maps IP addresses to domain names and is commonly used in reverse DNS lookups. But, PTR records can also be used to point from one domain name to another. The TSLA record is used to store a X.509 certificate in DNS,[11] enabling the DNS to serve as a root of trust for, e.g., a TLS connection [HS12]. Additionally, there are resource record types related to DNSSEC. On the one hand, a DNSKEY resource record is used to store the public key for a DNS zone, which is used to verify the digital signatures on the zone's resource records. On the other hand, the RRSIG resource record stores the digital signature for a resource record, which can be used to verify the authenticity of the information stored in the record. Further, the NSEC and NSEC3 resource records link to the next DNS name in a zone and are used to verify the nonexistence of a DNS name [Lau+08].

For operational reasons, each zone has two key pairs: The Zone Signing Key (ZSK) is used to sign all the records in the zone. This ZSK is then added to a DNSKEY in the zone. In turn, this DNSKEY record is then signed by the Key Signing Key (KSK), resulting in a RRSIG record. To establish trust into this KSK, it is signed by the zone one level higher in the DNS hierarchy, resulting in a RRSIG in that zone. For example, the root zone uses its ZSK to sign the KSK's of all top-level domains (e.g., *.at*), and stores the resulting signatures. The root zone's KSK is publicly known and directly trusted by all participants of the DNSSEC system. This structure forms a trust hierarchy from the root zone's KSK down to the individual record (e.g., *iaik.tugraz.at*). Each zone operator can only sign keys for zones in the subtree of the DNS namespace of which it is the root (e.g., the *.at* operator cannot sign the key for *europa.eu*). This trust hierarchy thus represents a top-down trust model using name subordination, as discussed by Perlman, p. 4.

4.5. Prolog and logic programming

Prolog is a logic programming language [ISO95]. In Prolog, programs are composed of a set of rules and facts (predicates) that define relationships between objects and concepts. The language is based on a formal system

[11] when used in combination with DNSSEC, this is called DNS-based Authentication of Named Entities (DANE)

of logic called definite Horn clauses, which allow for the representation of complex relationships and the derivation of new facts from existing ones through logical inference.

A **Horn clause** is a logical formula with at most one positive literal, also called the head, and any number of literals, which are also called the body. The head and body are separated by the reversed implication symbol (←) or the symbol `:-`.

In Prolog, a relationship between terms is specified as a rule, given as a *definite* clause. A **definite Horn clause** is a special type of Horn clause that has exactly one positive literal in the head and any number of literals in the body. In other words, a Definite Horn clause is a Horn clause with a single positive literal in the head. Definite Horn clauses are used in Prolog because they are well-suited for reasoning and inference in Prolog programs.

We discuss the syntax and semantics of Prolog in more detail in Section 7.3.

4.6. Cryptography and Privacy-enhancing technologies

Signatures: Digital signature schemes are a fundamental cryptographic primitive [Lys02]. In a signature scheme based on public-key cryptography, each user can generate a key pair consisting of a private key and a public key. The user can then sign a message using the private key, resulting in a digital signature. Another user can then verify this message using the signer's public key since it is public. If the signature is valid, the verifier can be sure of the message's authenticity, i.e., that the message has not been altered since it was signed. Since the signing key is private, only the signer can generate a valid signature; hence, the verifier can be sure that that user signed the message.[12]

We recall the standard definition of a signature scheme [GB08, Section 10.1; AMR20]:

A signature scheme SIG is a triple $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ of PPT algorithms,[13] which are defined as follows:

[12] Verifying that the public key belongs to the correct user is a separate task, as discussed in Chapter 2.

[13] probabilistic polynomial time algorithms [KL14, Section 3.1.2]

$\mathsf{KeyGen}(1^\kappa)$: On input security parameter κ outputs a (private) signing key sk and a (public) verification key pk with associated message space $\mathcal{M}$.

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

$\mathsf{Verify}(m, \sigma, \mathsf{pk})$: On input a message $m \in \mathcal{M}$, a signature σ, and a public key pk, outputs a bit $b \in \{0, 1\}$ indicating the validity of the signature.

Multi-Signatures: In extending signature schemes to multi-signature schemes, signatures on the same message w.r.t. some public keys can be aggregated into one compact signature. This aggregated signature is valid w.r.t. an aggregated public key. We define such signatures following the definition of Drijvers et al. [Dri+19, Section 5.1; Abr+20]:

A multi-signature scheme MSIG is a signature scheme SIG with an additional triple $(\mathsf{APKs}, \mathsf{ASigs}, \mathsf{AVerify})$ of PPT algorithms, which are defined as follows:

$\mathsf{APKs}(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$: This algorithm takes n public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ as input and outputs an aggregated public key pk_M.

$\mathsf{ASigs}((\sigma_1, \mathsf{pk}_1), \ldots, (\sigma_n, \mathsf{pk}_n), m)$: This algorithm takes n signatures $\sigma_1, \ldots, \sigma_n$ on the same message m and the corresponding public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$, and outputs an aggregated signature σ_M on the message m or $\perp$ on error.

$\mathsf{AVerify}(m, \sigma_M, \mathsf{pk}_M)$: This algorithm takes a message $m \in \mathcal{M}$, an aggregated public key pk_M, and an aggregated signature σ_M as input and outputs a bit $b \in \{0, 1\}$ indicating the validity of the signature.

Boneh–Lynn–Shacham (BLS) is a provable secure pairing-based signature scheme for producing short signatures [BLS04]. One property of BLS is that it can be used as a multi-signature scheme; signatures by multiple private keys can be combined into a single constant-size aggregate signature. This saves space and verification time [Bol03; BDN18].

Non-interactive Zero-knowledge (NIZK) proof systems represent powerful tools that enable a prover to convince a verifier of the validity of a

statement without revealing any other information [GMR85; BFM88]. For an NP-language $L \subset X$ and a statement $x \in X$, a prover can present a proof to the verifier that $x \in L$, e.g., there exists a witness w such that $x \in L$. No other information about the witness w is leaked to the verifier.

Formally, let R be the associated witness relation such that $L = \{x \in X \mid \exists w\colon R(x, w) = 1\}$. A non-interactive proof system Π consists of algorithms $\mathsf{Setup}(1^\kappa)$ producing a common reference string crs, $\mathsf{Proof}(\mathsf{crs}, x, w)$ taking a statement $x \in X$ and a witness w, and outputting a proof π, and $\mathsf{Verify}(\mathsf{crs}, x, \pi)$ taking a statement x and a proof π, and outputting the verification status.

We require such a proof system to be *complete* (i.e., all proofs for statements in the language verify), *sound* (i.e., a proof for a statement outside the language verifies only with negligible probability), and *zero-knowledge* (i.e., a proof reveals no information on the witness).

Succinct Non-Interactive Arguments of Knowledge (SNARKs) are one of such systems and of particular interest to us, as they come with small proofs that are independent of the witness size and allow for fast verification [Bit+12; Abd+23]. With the work of Groth [Gro16], SNARKs have been improved in various directions. To reduce the trust assumptions necessary for the generation of the common reference string crs, subversion-resistant and updatable versions have been investigated [Gro+18; ARS20].

Toolsets[14] including ZoKrates [ET18], arkworks[15] or xJsnark [KPS18] provide compilers to turn arbitrary programs into circuits suitable for SNARKs or implement building blocks to help with the design of suitable circuits.

ZoKrates offers a high-level language syntax akin to Python with static typing. It allows to implement functions and programs that represent statements to be proven with a Non-Interactive Zero-Knowledge Proofs (NIZKs). Internally, the program will be represented as rank-1 constraint systems to be consumed by bellman[16] which implements Groth's SNARKs [Gro16].

[14] `https://github.com/ventali/awesome-zk`, accessed on 2023-09-14
[15] `https://arkworks.rs`, accessed on 2022-07-07
[16] `https://github.com/zkcrypto/bellman`, accessed on 2022-07-01

5

Research Goals

In this thesis, we improve the state of the art of both trust management and privacy in a global context. To do so, we focus on different steps of the identity and trust verification process.

The overall goal is a authentication and authorization system that

- uses qualified information to assess the trustworthiness of a transaction and different authorities for different aspects of trust
- can be used globally in a heterogeneous world
- supports both existing and novel identity management models, and is extensible to be adapted to new technologies and models
- provides privacy to the user

In the remainder of this chapter, we describe the goals we will aim to satisfy in the following chapters. To better illustrate the scientific contribution, we also discuss the challenges involved in working on those goals. Our contribution to each challenge is described in detail in the later chapters of this thesis.

5.1. Goal 1: Support Different Qualities of Trust

To check the trustworthiness of some digital information, a Service Provider (SP) needs to verify if a issuer indeed issued the respective information. Further, the SP needs to check if the issuer is qualified to issue information of that type. To do so, the SP needs to discover information about the issuer (qualification status, cryptographic material, metadata) and authenticate it. Real-world electronic transactions are complex, involving many different qualifies of trust. Hence, in this thesis, we focus on

transactions that consist of multiple pieces of information, issued by different issuers, trusted in different trust schemes.

5.1.1. Complex Transactions

When evaluating the trustworthiness of an electronic transaction, a SP wants to get a complete picture about the transaction. Since different trust schemes might be relevant for different aspects of trust, a SP might want to consider trust information from different schemes instead of just a single authority. For example, while identity-centric schemes provide information about (legal) identities, reputation-centric schemes could provide information about customer satisfaction ratings, and business-centric schemes are responsible for credit ratings [BL16].

We address these challenges in this thesis by using electronic transactions that consist of multiple attestations, for example various credentials and certificates. Each of those attestations contributes a specific piece of trust information. Combining them during verification then enables a holistic trust assessment. While information about those trust aspects is coming from inside the same trust scheme (e.g., country's eID scheme), it might also be issued by different schemes.

5.1.2. Setup of Trust Anchor

To automate trust verification, it is necessary to automatically discover the required trust information. In traditional trust management systems, the SP either already has this information (in a trust store) or knows where to retrieve it (e.g., from a trust status list). In addition to discovering the source of trust information itself, the SP also needs to retrieve up-to-date cryptographic material to authenticate the trust information.

For example, in the eIDAS trust scheme, a SP needs to be aware of the latest URL of the European List of Trusted Lists (LOTL) published by the European Commission (EC) [Eur14]. This list is needed to assess whether an issuer is qualified. Also, to authenticate the list, the SP needs to retrieve (the fingerprints of) the certificates that the EC authorized to sign its LOTL. Both information is published in non-machine readable form in the Official Journal of the European Union.[1] This information forms the root of trust and is thus required for each scheme which is

[1] https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:52019XC0816(01)&from=EN, accessed on 2023-02-03

relevant for a SP. Thus, so far, manual effort is needed to set-up and maintain trust information retrieval for each scheme.

In Chapter 6, we address this challenge by introducing a trust management infrastructure based on the Domain Name System (DNS). Using this infrastructure, a SP establishes trust in a previously unknown trust scheme by simply configuring a single human-readable identifier. Our system then resolves this identifier and automatically retrieves the trust anchor of the scheme. We use the DNS Security Extensions (DNSSEC) to authenticate the cryptographic material. Afterwards, the SP can use our system to directly query the trust scheme, e.g., to verify whether a issues is qualified in that trust scheme. Doing so simplifies the setup process which is otherwise tedious to perform. This is especially the case if multiple trust schemes are involved in the assessment of an electronic transaction, since a separate trust anchor is needed for each scheme.

5.1.3. Different Types of Trust Schemes

When certifying the trustworthiness of a certain entity or information, different trust schemes use different means to communicate this quality of trust [BL16]. For example, some schemes only certify whether they consider a certain entity trustworthy or not (boolean). Others use a ordinal level to represent their degree of confidence, e.g., in the form of a level of assurance they have into this fact [WKR19]. We discuss the different types of trust schemes in more detail in Section 6.1.

A system that supports not a single but many trust schemes hence also needs to support this heterogeneous nature of representing trust status information. This is both the case for the trustworthiness of information in a credential as well as trust information about an issuer/authority. Additionally, when a trust verification process works with information from multiple trust schemes, there is the need to also "translate" this trust information between those schemes [Roß17].

We address this challenge within this thesis. In particular, in Chapter 7, we enable SPs to define their own rules on the required quality of trust in an expressive way. Further, in Section 6.3, we discuss the automated translation of trust data between schemes of different type.

5.1.4. Local Trust View

When a SP relies on a trust scheme to assess and certify the trustworthiness of some information, it depends on that scheme's understanding of trust. Similar, as a common alternative some SPs send the attestation to a Validation Authority (VA) for trust verification.[2] In that case, the SP depends on the VA's perception of trust, i.e., on the trust scheme(s) that the VA trusts.

This is a limitation of current approaches, since the rules about the trustworthiness of data on hand are specific to the concrete SP or a business use case. Those rules are usually not known to a scheme operator or VA as they represent a very specific perception of trust.

To mitigate this, the SP needs to formulate their own perception of trust for the verification of a transaction. Hence, the SP must be enabled to define their own trust rules in an expressive way. This is not possible if the SP has to rely on a single trust scheme, or if the software used by the SP codifies these trust rules in its source code.

In addition to specifying rules about whether to trust an entity, the SP also needs to specify additional rules about which transactions they accept under what condition. For example, a merchant might require a higher level of trust for orders of more expensive goods. Those rules are usually even more specific to a certain business case, thus decoupling them from the implementation is a requirement as well.

We address this challenge in Chapter 7 by introducing a expressive policy system. Our system separates the trust and access control logic and encoded rules from the business logic/software. The result is a configuration file that encodes the specific rules, a so-called "policy".

5.2. Goal 2: Global Interoperability

A modern trust management system has to deal not only with electronic transactions issued in the local trust scheme, but also from other trust schemes. In our first goal, the SP trusts the scheme directly. Further, it is likely that all system participants can agree on a format to encode data. However, both assumptions are less likely in the global setting. Thus,

[2] e.g., signaturpruefung.gv.at, or *trusted validators* in the EU's proposed eHealth Network [GL23]

as we focus on trust management in this heterogeneous environment, additional challenges arise.

5.2.1. Global Trust Scheme Interoperability

When operating within a single trust scheme (like a country, or the EU), there is often a single root of trust (like the government, or the EC) that is qualified to certify information or authorize issuers. Even if there are multiple issuers for various trust contexts (like stated above), when it comes to legal binding of information, there is only one qualified root per trust scheme. In contrast, in a heterogeneous setting there is no such thing as a single root of trust. An example for this is an European customer interacting with a Chinese online shop.

We address these challenges by introducing a trust scheme recognition and translation system (see Section 6.3). Following this approach, a trust scheme authority that recognizes another trust scheme can publish this recognition using our DNS-based infrastructure. A SP can then securely retrieve this recognition, and use it to establish trust in attestations issued in that foreign scheme. If the two trust schemes use the same understanding of trust, our approach is conceptually similar to eIDAS's Article 14 about international aspects [Eur14, Article 14]. However, we also cover the case of trust schemes with different understandings of trust, by introducing an automated trust translation system.

Trust Policy Aspect of Global Trust Management: An architecture that enables the discovery and authentication of trust information from multiple schemes is the basis for global automated trust management. To also bring the benefits of custom trust perceptions discussed above to the global setting, the involved policy system needs to support this scenario as well. Additionally, some business processes might require the definition of additional restrictions on trustworthy trust schemes. On the other side, it is also possible that some business cases or domains allow the translation of certain trust attributes that are not globally valid. Thus, our automation system also needs to support the codification of rules concerning global trust management. In Section 7.4.3, we tackle this challenge by extending our policy system with support for trust scheme recognitions and translations.

5.2.2. Attestation Format Interoperability

Attestations like credentials and certificates are encoded using various formats and schemata. A challenge of global trust management is the large variety of formats and schemata in a heterogeneous world. A credential issued in some trust scheme might be encoded using a different encoding schema or even file format than a credential issued in a different scheme. The SP needs to not only ensure the trustworthiness of some credential, but also understand the semantics of the data encoded in the credential. But a SP can only understand a limited set of credential schemata. To mitigate this, the FutureID project introduces the concept of simple credential transformers (SCTs) that can handle different types of credentials [Fut14]. But, both the SCTs as well as the policies they enforce are hardcoded in the application, and the project does not discuss the secure and trustworthy retrieval of additional SCT.

We tackle this challenge in Chapter 8 by introducing a generic framework for trustworthy credential transformations. Using this framework, a SP automatically retrieves the data needed to transform a credential into a format and schema it can parse. Further, our framework optionally also allows to discover and retrieve the required custom signature verification code. Doing so, we enable SPs to keep using the access policies formulated for some local credential schema while handling a larger set of credential representations.

5.3. Goal 3: Extensibility

When building a framework tailored for a global context, it is impossible to anticipate all technologies and models that are relevant today and tomorrow. Hence, it is important to build extensibility into the system already on the conceptual level. By introducing an open framework based on the DNS and utilizing trust translation, we enable the extensibility for additional trust schemes. Additionally, our policy system is extensible to support new features, trust data sources, and transaction formats. Further, our system is also open for additional trust models, to make it future-proof for novel technologies and needs.

5.3.1. Novel Models

Authorization systems rely on different trust management models. For example, many established identity management models use centralized (and often hierarchical) trust management models. Additionally, user data is often stored on servers, or directly in the registry of some authority. In contrast, novel identity management models like Self-Sovereign Identity (SSI) aim to store the identity data directly on the device of the user [Abr17]. Additionally, these novel models use data structures like Distributed Ledgers (DLs) for trust management. It is a challenge for automated trust verification systems to support both centralized and distributed models, and to keep up with new developments.

In our thesis, we enable SPs to formulate trust policies for centralized (see Section 7.4.2) and decentralized (see Section 7.4.3) trust schemes. To be future-proof, we introduce concepts that enable the easy extensibility of our policy system without requiring modifications to the policy language itself. We demonstrate this approach by extending our policy system with support for the SSI model (see Section 7.4.4).

5.3.2. Trust among Peers

Since many trust structures in the real world follow a hierarchical approach, a system using a hierarchical architecture (like in the Public Key Infrastructure (PKI) model) is able to represent those processes in a machine-processable way. However, some use cases use trust information not in a hierarchical structure, but provide trust information among peers (Web of Trust (WoT)). For example, when verifying the authenticity of a diploma issued by a foreign institution, a university might ask another university for their previous knowledge about this institution. This peer-to-peer trust information can then be used by a SP to authenticate incoming information, or to enrich existing information (from hierarchical systems) and aiding automated decision support.

To enable trustworthy peer-to-peer information sharing in a generic way, in Section 6.6 we discuss a system where any entity can encode and publish (trust) information about any other entity. For this we publish trust information on a DL. By using the WoT model, we ensure that a SP uses only trusted information. Additionally, we connect this system with our credential transformation system (as introduced in Chapter 8) by publishing transformation information on InterPlanetary File System

(IPFS) and authenticating it using the WoT on the DL.

5.4. Goal 4: Privacy

While privacy is important (and a human right), ensuring privacy in complex systems presents a considerable challenge. Thus, privacy is a "horizontal" goal of this thesis, concerning several aspects. It considers the content of an electronic transaction as well as the user's behavior. See also Section 3.2 for a discussion of related privacy concepts.

5.4.1. Confidentiality

The retrieval and verification of trust information from trust status lists is a tedious process [SLL13] (see also Section 5.1.2). Therefore many SPs turn to VAs for transaction verification. This has the advantage that they only need to send the transaction to the VA, and receive a verification status as result. However, in sending the full transaction in plaintext to the VA, they might reveal sensitive data to the VA [BL16], which violates the requirement of confidentiality (cf. Section 3.2). Further, since the transaction contains the certificates, the VA also learns about the identity (and thus their behavior) of the transaction's signer.

We address this challenge in Chapter 6 by removing the need to send the transaction to a third party. Instead, we enable the SP to easily establish trust in the transaction locally. Using our DNS-based approach, the SP directly queries the system for the issuer's status, and only the fingerprint of the issuer's certificate is send to the trust management infrastructure.

5.4.2. Integration of privacy technology into access control systems

Authentication, authorization, and trust verification are central parts of an access control system. The conditions for granting access in such a system are collected in access policies. Since access conditions are often complex, policy languages for defining policies are in use.

However, current implementations suffer from privacy issues. Users are often in possession of credentials that certify numerous attributes. When showing a credential to an SP, users reveal all attributes to the SP, which is often neither desirable nor necessary to fulfill an authentication request.

By integrating privacy-preserving technologies into the access control process, users are enabled to only reveal a subset of the attributes, or even prove a statement without revealing any attribute at all.

Match expressive Access Request with Access Policies: Various privacy-enhancing/-preserving technologies exist to protect the user's privacy, ensure confidentiality, and sometimes even enable anonymity or pseudonymity. But integrating these technologies into existing access control systems is not straightforward.

Several points need to be addressed: How should sensitive attributes be marked hidden in a policy? Which statements on the hidden attributes need to be revealed? How is the user informed on the statements they need to prove? Which privacy-preserving technologies can help to overcome these challenges while being flexible enough to preserve the expressiveness of the policy languages?

We tackle this challenge in Chapter 9 by extending an expressive access control system with privacy features. In our approach, existing access control policies are re-used, requiring only minor modifications to define which attributes need to be revealed. Our system then automatically derives an access control flow from the policy. This effectively turns an existing access control system into a privacy-preserving access control system.

5.4.3. Undetectability

Trust management systems often use registries to dynamically obtain additional data needed to authenticate attestations or form trust decisions. Examples are revocation registries and trust status lists.

Online Registries: During the verification of an attestation the SP queries the registry, e.g., to retrieve trust status information of the credential. While this ensures trustworthy information, the connection from the SP to the registry poses a privacy issue, as it leaks information about the user's behavior (see also Section 3.2). Additionally, the process requires the SP to be online and the authority available, which poses a challenge for the availability of the system. While concepts like PKI enable the offline verification of the trustworthiness of a credential, this does not help with revocation checks.

Decentralized Registries: By introducing concepts like DLs, it is possible to create decentralized registries. Instead of contacting a registry, a SP then needs to query a node of the respective ledger, e.g., to retrieve trust status information during the verification of a credential. This leaks information about the user's behavior to the DL node (or an observer). Additionally, such a check does not work if the user is offline. Abraham et al. discuss the offline verification of DL-based revocation information.[Abr+20].[3] However, their system is limited to only the revocation use case and a specific revocation scheme. Modifying this system for other use cases is possible, but each additional use case requires adaptions on all of the DL's nodes.

We resolve these issues in Chapter 10 by extending existing ledger APIs to support generic results that are trustworthy without directly communicating with the DL node during the interaction with a SP. By introducing a simple query system, we enable the attestation of any data stored on the ledger, without the need to modify the system for each use case. Further, the results are trustworthy even in an offline setting. Our approach is similar to Online Certificate Status Protocol (OCSP) but adapted to a distributed architecture: We introduce attestations of the ledger's state, issued by ledger nodes, aggregatable into a collective attestation by all nodes. This attestation enables a user to prove the provenance of any DL-based data to an offline SP.

5.5. Summary

In the rest of this thesis, we address the stated research goals. Table 5.1 illustrates the chapters of our thesis and their relation to the research goals.

[3] Note: The author of this thesis co-authored the publication.

Table 5.1.: Overview of our research goals and where in our thesis we adress them.

Goal	Sub-goal	Contribution
1) Different Qualities of Trust	Complex Transactions	Chapter 6, Chapter 7
	Setup of Trust Anchor	Chapter 6
	Different Types of Trust Schemes	Chapter 6, Chapter 7
	Local Trust View	Chapter 7
2) Global Interoperability	Global Trust Scheme Interoperability	Chapter 6 (Section 6.3), Chapter 7
	Attestation Format Interoperability	Chapter 8
3) Extensibility	Novel Models	Chapter 7
	Trust among Peers	Chapter 6 (Section 6.6)
4) Privacy	Confidentiality	Chapter 6, Chapter 9
	Integration of privacy technology	Chapter 9
	Undetectability	Chapter 10

Automated Trust Management in Heterogeneous Environments

Research Area 1 Introduction

Outlook

In this part of the thesis, we present our research in the area of **Trust**. Our work in this area resulted in six scientific publications (see Section 1.2.1). The presentation of this research area is structured into the following three chapters:

In Chapter 6, we present our **Global Trust Infrastructure**. Using this infrastructure, Service Providers (SPs) can verify the trustworthiness of a complex electronic transaction, even if the transaction stems from a different country (or trust scheme).

First, we introduce our publication method for trust schemes based on the Domain Name System (DNS). In this method, trust scheme operators use the DNS to publish information about the qualified issuers of their trust scheme. The result of this publication is a simple and human-readable identifier for the trust scheme. The verifier of an attestation can directly resolve this identifier to assess the trustworthiness of an issuer. To authenticate the trust information, we use the DNSSEC root as a global root of trust. We then extend this trust scheme publication system beyond a single trust scheme. In our extended method, trust scheme operators that recognize another trust scheme publish this recognition in a machine-readable form. For trust schemes that don't agree on an understanding of trust, we further allow the publication of trust translations. Using these translations, a verifier can translate the trust data of an incoming attestation to assess whether it considers it trustworthy. We also provide a reference implementation and discuss our approach.

Besides our DNS-based trust publication approach, we present an alternative publication method that uses a distributed ledger. We enable all system participants to publish trust statements about each other, which forms a web of trust. The publication of these trust statements is done in a smart contract. This results in an open trust system, enabling use cases that follow a peer-to-peer sharing of trust information.

In Chapter 7, we present our **Trust- and Access-Policy** system *TPL*. SPs use this system to specify rules about their own perception of trust, i.e., which schemes they consider trustworthy.

The TPL system enables SPs to customize their system's conditions to determine an incoming transaction's trustworthiness automatically. The TPL system supports our novel DNS-based trust infrastructure, as introduced in Chapter 6. As opposed to the state of the art, TPL can be extended to work with novel trust models and trust data sources. To demonstrate this, we extend TPL with concepts used in the Self-Sovereign Identity (SSI) model. Doing so enables SPs to base their trust decisions on established as well as novel models in the same transaction. Besides the specification of such a trust policy, TPL also provides means to specify an access policy, encoding further business logic-related rules. The TPL system also serves as a basis for graphical policy-authoring tools. Such tools allow non-technical domain experts to create and modify policies to configure the system's trust rules to their needs and local regulations.

In Chapter 8, we present our work on **Credential Format Transformations**. To enable automated trust management on a global scale, SPs need to be able to trust incoming transactions (as described in Chapter 6) and understand the contents of those transactions.

In our transformation approach, trusted entities publish transformation information describing how to transform credentials between formats (i.e., encoding schemas). Depending on the type of the trust scheme, those trusted entities are either authorities qualified in the scheme or entities trusted on a peer-to-peer basis. This transformation information is then automatically retrieved and authenticated by SPs. Afterwards, SPs can use the information to transform incoming credentials from a foreign scheme's credential format into a format they understand.

6

Towards a Global Trust Infrastructure

This chapter is based on the papers *DNS-based Trust Scheme Publication and Discovery* by G. Wagner, S. Wagner, More et al. [Wag+19] and *Trust Scheme Interoperability: Connecting Heterogeneous Trust Schemes* by More [Mor23]. Parts of the latter paper have been copied verbatim. Additionally, parts of this chapter were conceptualized and implemented as part of the Horizon 2020 project LIGHTest.

Assessing the truth of received information is a crucial component of electronic transactions. For example, to establish liability in a business transaction, a Service Provider (SP) needs to determine the identity of their business partner. In that regard, trust schemes are used to support SPs in assessing the trustworthiness of the identifier of some entity. Further, trust schemes are also used to establish trust in other aspects of electronic transactions. To provide the SP with all required information to assess an electronic transaction, the transaction can consist of multiple attestations issued by different issuers. For example, a seller of goods might be interested in the identity of prospective buyers as well as their financial standing.

One critical step during the verification of a signed electronic transaction is trustworthy identification, hence to establish trust in the transaction's signer. This is commonly done by verifying the signer's attestation (e.g., certificate) and assessing the trust in the issuer who issued the attestation. The last step is done by determining if the issuer is a member of a trusted trust scheme. In various trust schemes such as the European Union's (EU) eIDAS, the trusted issuers[1] are published in list form—a so-called *trusted list* (see also Chapter 4).

[1] in eIDAS, such issuers are called (Qualified) Trust Services, operated by Qualified Trust Service Providers (QTSPs)

While in the EU all eIDAS-compliant verifiers of electronic transactions are aware of the location of this European trusted list, this is not the case for trusted lists of other trust schemes, such as those of non-member state countries or domain-specific schemes. The same is true for the cryptographic key material needed to authenticate the trusted list: in eIDAS this has been elegantely solved by publishing the EU's trust list signing certificates in the Official Journal of the European Union.[2] But, the secure retrieval and update of those certificates remains a manual process. Further, as the EU Journal acts as a *legal* root of trust, the verifier needs to authenticate it beforehand,[3] which is not possible for foreign trust schemes without prior trust relationship.

In addition to discovering and authenticating foreign trust schemes, it is necessary to ensure the other trust schemes can actually be trusted. This is especially a challenge if the other trust scheme uses a different understanding of trust.

In this chapter, we introduce a trust management architecture which enables a verifier to discover and retrieve required trust information automatically from various sources by using the Domain Name System (DNS). Doing so, users need to only configure the DNS identifier of the schemes they directly trust, freeing the users from manually configuring trust information and cryptographic material. This is especially crucial if a user in a single transaction relies on multiple trust schemes for different qualities of trust, which would otherwise multiply the setup costs. Our approach further allows the verification of attestations issued in foreign trust schemes. We use the DNSSEC's global root of trust to automatically authenticate the retrieved trust information without requiring tedious management of cryptographic keys.

Chapter 6 Goals:

We discuss the overall goals of this thesis in Chapter 5. In this chapter, we focus on Goals 1, 2, and 3. In specific, we tackle the following aspects:

- from Goal 1: Complex Transactions, Setup of Trust Anchor, Different Types of Trust Schemes
- from Goal 2: Global Trust Scheme Interoperability
- from Goal 3: Trust among Peers

[2] https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv:OJ.C_.2019.276.01.0001.01.ENG

[3] https://eur-lex.europa.eu/content/help/oj/authenticity-eOJ.html

Chapter 6 Outline:

We start the chapter with a conceptual introduction to trust schemes (Section 6.1). In this section, we also introduce the relevant actors and describe the three types of trust schemes.

In Section 6.2, we present our first contribution: a DNS-based architecture to enable the publication and automated discovery of trust scheme information. We extend this architecture in Section 6.3 with a method to recognize foreign trust schemes; furthermore, we present a system to *translate* trust data between the scheme-specific structure and semantics of different trust schemes. We provide an implementation in Section 6.4. In Section 6.5, we evaluate and discuss our approach and state ideas for future work.

As an alternative to the DNS-based trust scheme publication, in Section 6.6 we discuss the publication of trust scheme information using a distributed ledger.

6.1. On Trust Schemes

Trust schemes help entities to automatically make trust decisions about electronic transactions they receive. To do so, trust schemes consist of legal regulations, technical standards, organizations, and infrastructure.

6.1.1. Actors

On a technical level, trust schemes are used to authenticate **attestations**. An attestation is a signed document containing some claims about the holder. For example, the document is called a certificate if it attests the binding between the user's name and their public key (e.g., X.509 certificate). Or, it is called assertion or credential if it contains identity attributes about the holder directly used to authenticate at a service (e.g., Security Assertion Markup Language (SAML) assertions, W3C Verifiable Credentials (VCs)).

A simplified trust scheme consists of the following actors, also shown in Figure 6.1:

- **Holder:** the user in possession of some **attestation**, which they present to the verifier or use to sign some data.

- **Verifier**: a component that is typically operated by each SP. It assesses the authenticity of incoming attestation by checking if the attestation was issued by a trustworthy issuer. The verifier relies on the trustworthiness of the data in the attestation and is thus also called "relying party".
- **Issuer:** the entity that attests the holder's attributes after checking them. To do so, the issuer signs the holder's claims. This results in an attestation, which the issuer sends to the holder. For example, in the eIDAS qualified signature use case, an issuer is called "trust service".
- **Authority:** the entity that authorizes some issuer to issue attestations, commonly the operator of a trust scheme. Depending on the trust model, authorities are either directly authorized, or they receive their authorization from some other authority. The authority on top of this authorization chain is called *root of trust* and must be trusted by all entities in a trust scheme.

An issuer is considered trustworthy by the verifier if it is qualified in a trust scheme that the verifier trusts. I.e., if it is authorized by an authority of that scheme. The trusted scheme and other trust requirements are configured in the verifier's **trust policy**. Additionally, the verifier checks if the attestation values match other business logic-specific rules. To do so, it evaluates the values of the attestation against the verifier's **access policy**.

6.1.2. Types of trust schemes

A trust scheme is, among other things, characterized by the requirements on how to acquire a qualified attestation. For example, a trust scheme might require that holders need to prove their identity in person, and use secure hardware to store their credentials. Another scheme could support remote identity verification and require no specific storage hardware. Trust schemes can thus be categorized by those requirements and how the fulfillment of these requirements is encoded. Following this, Wagner et al. describe three types of trust schemes [WKR19]:

Boolean trust schemes only allow attestations that fulfill all requirements. In that case, the mere existence of a (qualified) signature on the attestation is enough to certify that the attestation fulfills *all* requirements of the trust scheme.

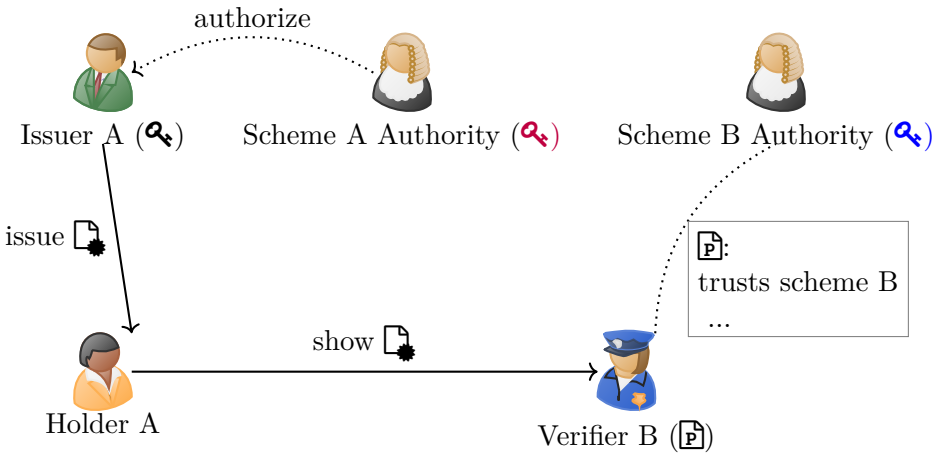

Figure 6.1.: Global trust management actors and their relationship. Verifier B does not trust Issuer A authorized in the untrusted trust scheme A.

Ordinal trust schemes differentiate between different types of attestations depending on the degree of trust they can achieve. This differentiation is represented in the form of a numeric level, such as the Level of Assurance (LoA) a verifier can have about the received information. By stating that an attestation has a certain level, an issuer certifies that the specific requirements for that level are fulfilled. Verifiers can then specify the required level in their policy.

Tuple-based trust schemes are an even more flexible approach to differentiate the level of trust. This flexibility is achieved by directly publishing the complete list of requirements of the trust scheme, and specifying which are fulfilled by a certain attestation in the form of key-value pairs. In the simplest form, each value is a boolean but can also be a ordinal level or a string.

For ordinal and tuple-based schemes, the level or tuples can be part of the attestation document, e.g., the certificate or assertion. In some trust schemes, the information is instead part of the issuer's authorization. For example, in a scheme using trusted lists, it can be part of the issuer's trust list entry.

6.1.3. Example: eIDAS Trust Scheme Verification

Inside the EU, to verify if an incoming attestation (e.g., certificate used to sign a PDF file) has been issued by a qualified issuer, a verifier uses the trust scheme specified by the eIDAS regulation (see Section 4.1). The verifier first loads and verifies the List of eIDAS Trusted Lists (LOTL) provided by the European Commission. It then uses one of the signing certificates it previously retrieved from the EU's Journal to authenticate the list. Afterwards, the verifier uses the LOTL to discover, load, and verify the Trusted List of the respective member state of the issuer. Using this country list, the verifier ensures that it lists the issuer of the attestation as a qualified trust service. This process determines whether the attestation is qualified, trustworthy, and thereby legally binding.[4] Since all trusted lists are published publicly on the web in Extensible Markup Language (XML) form, and XML Advanced Electronic Signature (XAdES) are used to authenticate them, this process can be automated. But, this is only possible if the location of the LOTL (the trust root) and the certificate used to sign it is known to the verifier.

6.2. DNS-based Trust Scheme Publication and Discovery

To check the trustworthiness of an incoming attestation, a SP uses an automated trust verifier. To know if the attestation has any legal value, the verifier must check if it was issued by a qualified issuer. An issuer is qualified with respect to a certain trust scheme if it is a member of that trust scheme, i.e., if the issuer is authorized by that trust scheme's authority.

Additionally, the verifier needs to assess the confidence it can have in the information provided in the attestation. For a certificate that binds a cryptographic key to a (legal) identity, this means how sure the verifier can be about the identity of the holder that signed the incoming document. The possible degree of confidence depends on the type of the trust scheme. For example, the trustworthiness of the identity is determined by factors such as how the holder needed to prove their identity (at enrollment), e.g.,

[4] The requirements for qualified electronic signatures involve further points, e.g., that the signature must be created by a qualified signature creation device (QSCD) [Eur14, Article 32 and Annex I].

remote or in-person. Other factors are the type of information known about the holder and how the credential is stored and managed.

In this part of the thesis, we enable trust scheme operators to publish trust scheme information. Further, we enable verifiers to retrieve this information securely and to use it to assess the trustworthiness of incoming attestations automatically. By doing so, we demonstrate how the DNS can be used to publish trust scheme information, either replacing or enriching the established trusted list approach. Further, we show how the global agreement on a single root of trust in DNSSEC could be used to facilitate a global trust management system.

Phase 1 **Trust Scheme Publication:**

We enable trust scheme operators to publish trust scheme information using the DNS. By doing so, they provide information about the issuers that are a member of their scheme. Depending on its type, a trust scheme can also characterize the trustworthiness of attestations issued by its members. Our system ensures the authenticity of the published information by signing it using the DNS Security Extensions (DNSSEC).

Using a DNS domain name for publication results in a simple and human-readable identifier for each trust scheme and issuer. Since a trust scheme identifier is "resolvable" (using DNS), it can directly be used by all participants and other trust schemes to discover trust information relevant to this scheme. Besides pointing to trust information, the DNS records also contain digital certificates which are used to authenticate the trust information.

Phase 2 **Trust Scheme Discovery:**

During the verification of an attestation, the published trust scheme information can then be discovered, retrieved, and authenticated. This enables a verifier to automatically check the trust scheme membership of the attestation's issuer. Further, the verifier can use the information about the quality of the attestation and compare it with the SP's trust policy.

While this system can be used to authenticate attestations issued by the scheme the verifier directly trusts; it also forms the basis for the verification of attestations issued in a *foreign trust scheme*, introduced in Section 6.3 below.

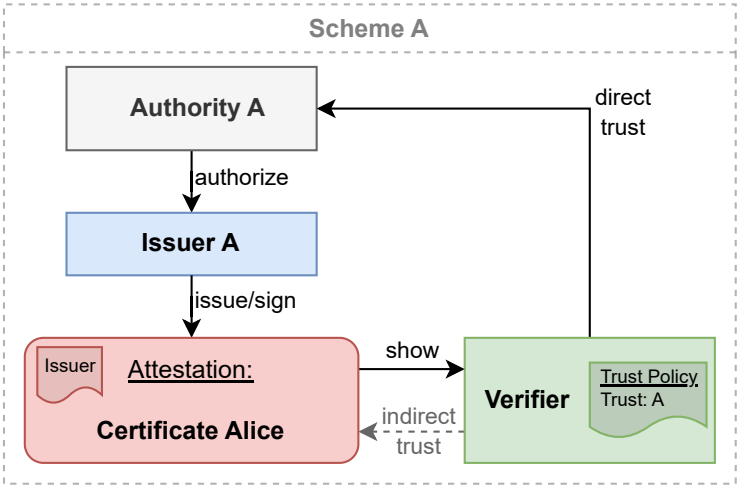

Figure 6.2.: Example trust path for a published trust scheme.

6.2.1. Trust Scheme Information

Trust scheme information is the set of issuers that are a member of the trust scheme, thus qualified to issue qualified information in that scheme. This member set can be published in the form of a list.[5] This list further contains additional information about the individual scheme members. The additional information depends on the type of the trust scheme. For example, for ordinal schemes, each list entry may contain a specific (ordinal) level (if this level is not stored directly in the issuer's certificate). Correspondingly, for tuple-based schemes, the scheme membership is characterized by the values of a set of tuples. The set of tuples is defined by the trust scheme, and for each member, the scheme operator assigns values to all of those tuples, depending on which requirements the individual issuer does (not) fulfill.

6.2.2. Publication Process

Encoding of scheme information: A trust scheme operator uses its governance framework to determine the list of issuers that are members of its trust scheme. The operator then encodes this list in machine-readable form and signs it using its certificate.

[5]This list is commonly called Trusted List (eIDAS), Trust Status List (Kantara Initiative), or Trust Service Status List (FutureTrust) project.

Additionally, the list also contains some data about the list itself. Most importantly, it specifies the time period in which it is valid.

Publication of scheme information: The trust scheme operator then publishes the list file on a public server and makes it available using a network protocol understood by all parties (i.e., Hypertext Transfer Protocol (HTTP)). It then performs the publication by publishing in the DNS a pointer to the list. This is done by operating a DNSSEC-enabled DNS nameserver for the DNS zone representing the scheme's identifier. The publication is done in the form of a Uniform Resource Identifier (URI) record at the DNS name of the scheme. For example, the operator of the "Example.com" scheme operates a DNS nameserver at `example.com`. The operator is also in possession of the DNSSEC private key of this DNS zone. A URI record at `_scheme._trust.example.com` then points to the (HTTP) location of the published list. Additionally, the operator publishes its certificate in a TSLA record at the same DNS identifier. URI and TLSA records are signed using the DNSSEC key of the zone. This DNSSEC signature delegates the trust from DNS to the signed list.

As an additional shortcut, ordinal trust schemes can optionally publish a pointer to a (signed) sub-list containing all entries of a certain level. For example, a pointer to the list with all entries of level 3 could be published at `_level3._scheme._trust.example.com`. After doing so, the trust scheme information can be discovered and authenticated by any verifier using solely the (DNS) scheme identifier. The result of a trust scheme publication is a DNS zone, as shown in Listing 6.1. The chain from the verifier to the attestation forms a trust path, as shown in Figure 6.3.

Adding and revoking issuers: If the trust information of a scheme changes, the scheme operator re-issues a new list and publishes it at the same (HTTP) location or a different one requiring an update of the corresponding DNS records as well. This update is needed if the set of qualified issuers changes or if the level or tuples of one of the issuers changes. In the same way, a trust scheme operator can *revoke* a trust scheme membership by removing the entry of the corresponding issuer from the trust scheme information and re-publishing the list. Since any previously published list remains valid, the effectivity of the revocation depends on how often verifiers update their cached lists. An adversary capable of injecting an old list into a verifier could hide the revocation of an issuer until the list's expiry date. The same is true for DNS records[6]

[6] e.g., using DNS hijacking or DNS (cache) poisoning

```
_scheme._trust.example.com. IN URI  1 1 https://tspa.example.com/trust-list.xml
_scheme._trust.example.com. IN TLSA 3 1 2 <cert...>

_scheme._trust.example.com. IN RRSIG URI  13 4 10 20240501000000 20180501000000
                                    ↪  22222 example.com. <signature...>
_scheme._trust.example.com. IN RRSIG TLSA 13 4 11 20240501000000 20180501000000
                                    ↪  22222 example.com. <signature...>


example.com. IN DNSKEY 256 3 13 <key_22222 data>
example.com. IN RRSIG DNSKEY 13 2 8640 20240501000000 20111001000000
                                    ↪  11111 example.com. <signature...>

example.com. IN DNSKEY 257 3 13 <key_11111 data>
```

Listing 6.1.: Example DNS records of a scheme publication of the *example.com* scheme. TTL fields omitted for clarity. DNSSEC setup: Key 22222 (ZSK) signs the zone (RRSIG records for the URI and TSLA records), while key 1111 (KSK) signs the ZSK's record. Key 1111 is, in turn, signed by the .com zone (in its zonefile).

with a not-yet-expired signature (in the RRSIG record).[7] Depending on the scheme's structure and governance model, the revocation speed could be improved by requiring issuers to revoke their certificates, using standard X.509 revocation mechanisms.

Directly publishing trust scheme information in DNS: An alternative to publishing a pointer to the trust scheme information in the DNS is to directly publish the set of trust scheme members in the DNS. The DNS names for the URI records of individual issuers are formed by prepending the identifier of the issuer certificate (i.e., fingerprint) to the identifier of the trust scheme. Additionally, a TSLA record with the issuer's certificate is published alongside the URI record at the same location. For example, the URI record at `<fingerprint>._level3._scheme._trust.example.com` resolves directly to the trust information of the corresponding issuer certificate. Additionally, the existence of a (valid) TSLA record at that DNS name is enough to prove the issuer's trust scheme membership. Since the TLSA record contains the issuer's certificate, it also provides all information needed to verify that membership cryptographically. This approach makes most sense for boolean and ordinal trust schemes. Since the length

[7] RRSIG records have an expiry date independent of its TTL value, see [Are+05b, section 3.1.5] and [Are+05a, section 8.1].

of DNS names is limited,[8] the encoding of tuples into DNS identifiers is not possible. Further, this approach complicates caching of trust information by verifiers. Thus, it makes only sense for trust schemes where the set of qualified issuers changes frequently.

6.2.3. Trust Scheme Membership Claim

An issuer is a member of one or more trust schemes. To inform a verifier about this membership is the purpose of a trust scheme membership claim. This claim is represented by the (DNS) identifier of the trust scheme.

There are two methods to provide this claim to the verifier: Using a **direct trust scheme membership claim**, the issuer adds the scheme's identifier directly into its certificate. Alternatively, the issuer can also use an **indirect trust scheme membership claim**. In that case, the issuer certificate contains the (DNS) identifier of the issuer and not of the scheme. This claim is added to the issuer's certificate as X.509 extension *Subject Alternative Name*, or directly in the holder's attestation as *Issuer Alternative Name* [Coo+08, Sections 4.2.1.6 and 4.2.1.7]. At the stated DNS identifier, the issuer publishes the scheme membership claim(s) in the DNS. This publication is done by operating a DNSSEC-enabled DNS nameserver in the same way as described above. The (indirect) claim publication happens in the form of a PTR record, pointing at the scheme identifier of the trust scheme. For example, Listing 6.2 shows the DNS zone of Exampleissuer claiming membership in the example.com scheme.

```
_scheme._trust.exampleissuer.com. IN PTR  example.com.
_scheme._trust.exampleissuer.com. IN RRSIG PTR 13 4 10 20260501000000
                                  ↪  20200501000000 33333
                                  ↪  exampleissuer.com. <signature...>
```

Listing 6.2.: Example DNS records of an indirect trust scheme membership claim. The records are signed using exampleissuer.com's zone signing key (ZSK). TTL fields and DNSSEC records omitted for clarity.

An advantage of the indirect membership claim is that it is more flexible, e.g., when the scheme memberships of an issuer change after its certificate

[8] limit for each label: 63 bytes, for the full DNS name: 255 bytes [Moc87b].

was issued. An additional advantage of the indirect approach is that it only uses existing X.509 extensions. As this approach is more generic, the rest of this chapter uses indirect membership claims.

6.2.4. Discovery and Verification Process

When a verifier receives an attestation, it uses the published data to verify it. To assess whether the issuer of the attestation is qualified, it first extracts the issuer (DNS) identifier from the attestation (e.g., from the issuer certificate's Subject Alternative Name). It then resolves the identifier using DNS, retrieving PTR records with one or more trust scheme membership claims. As part of this DNS query, the verifier uses its configured DNSSEC root signing key and the DNSSEC signature chain to verify the signatures on the DNS records and checks if they are still valid. Next, it loads its trust policy to retrieve the list of identifiers of schemes it trusts. Using this list, the verifier checks if the issuer's membership claim is for a trusted scheme. If the scheme is trusted, the verifier directly proceeds. Otherwise, it first tries to establish trust in that scheme by using a trust recognition (see Section 6.3).

As the membership claim alone is not proof of scheme membership, the verifier proceeds with verifying the claim. To do so, it first "discovers" the trust scheme information by querying the DNS for the scheme identifier, asking for the URI record and corresponding TLSA record. Again, it checks the DNSSEC signatures on these records. Using the URI record, the verifier retrieves the trust information of the scheme, commonly using an HTTP request to the referenced webserver. Since this list is signed, the verifier then uses the certificate from the TLSA record to authenticate the list. Doing so, it verifies that this list was indeed issued by the trusted trust scheme. Further, it checks if the expiry date of the list is in the future. If this is the case, it proceeds by iterating on the list to extract the entry corresponding to the attestation issuer. If an entry for the issuer in question exists, the verifier considers the issuer qualified. It then concludes the process by using the issuer certificate extracted from the list to verify the signature on the attestation.

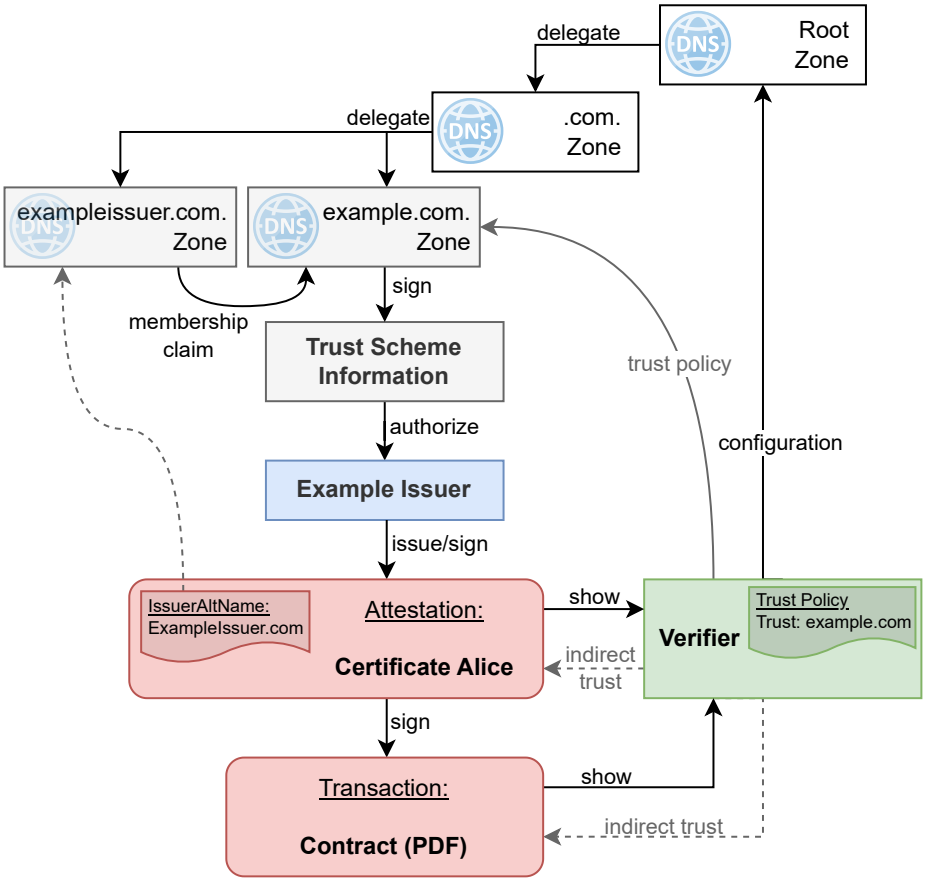

Figure 6.3.: Example trust scheme publication and trust scheme membership claim published in DNS. (1) The verifier extracts the claim from the attestation and uses it to discover the scheme information. (2) It then authenticates the scheme information using DNSSEC, and compares the scheme's identifier to the one configured in the trust policy. (3) The now-authenticated scheme information is then used to check if the issuer is qualified. (4) The certificate of the now-qualified issuer is then used to verify the attestation.

6.3. Connecting Heterogeneous Trust Schemes

Verifying a trust scheme membership is a challenge if the issuer is qualified in a different trust scheme than the one the verifier trusts. Since there is no common root of trust between the schemes, the trust scheme authorizing the issuer is not known to the verifier. Thus, the verifier has no information about the trustworthiness of that issuer. Consequentially, the attestation has no value for the verifier.

In some cases, there is a legal relationship between two trust schemes, e.g., a treaty between two countries, or some other form of recognition. In that case, the verifier needs to learn about this relationship. This allows the verifier to assess the trustworthiness of the foreign issuer, but it is currently a manual process that involves non-machine-readable information.

Trust Recognition: We enable trust scheme operators who recognize other schemes as equivalent to publish this recognition in the form of a *trust recognition*. The trust recognition can later be retrieved by a verifier. Using this recognition, the verifier can automatically authenticate the issuer of information, even if this information was issued in a different trust scheme.

Different understanding of trust: Since different legislations and use cases require a different confidence in a holder's identity, the verifier needs to ensure that the provided attestation meets their requirements. To do so, the verifier acquires the level of trust of the attestation and compares it with their local trust policy. The structure and semantics of this level of trust depend on the individual trust scheme. For their local trust scheme, either the verifier is aware of the trust levels of attestations issued by qualified issuers. Or, there is no need to know since the trust scheme already ensures compliance with the local laws which the verifier needs to follow. But, this is not the case if the attestation was issued in a different trust scheme.

Trust Translation: For situations where trust schemes are not equivalent, we provide a system to translate information about the quality of identities between schemes. We do so by attaching *trust translation data* to the published recognition. This translation data can be used by a verifier to automatically map trust data into a trust scheme type and semantics it understands. This enables a verifier to seamlessly work with trust information from trust schemes with a different understanding of trustworthiness.

6.3.1. Concept

When a verifier receives an attestation as part of some electronic transaction, it first authenticates the issuer of this attestation. To do so, it checks if this issuer was authorized by an authority of the local trust scheme (see Section 6.2). If this check succeeds, the verifier can directly check whether the attestation satisfies the service's trust policy. But: this authentication check fails if the attestation was signed by an issuer not authorized by one of the authorities in this trust scheme.

In that case, the verifier checks if there is a **trust recognition** from the issuer's trust scheme to the verifier's trust scheme. If this recognition exists, the verifier retrieves it and checks if the issuer is qualified in the other trust scheme. Further, if the schemes are not equivalent, the verifier retrieves the **trust translation** and translates the trust data into its own scheme.

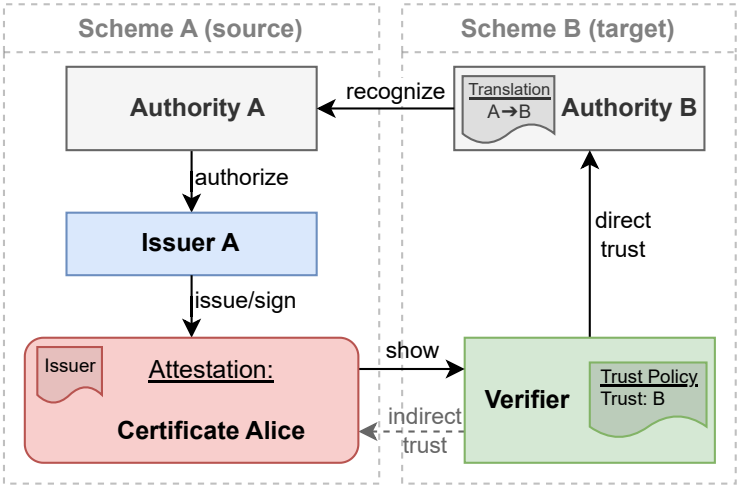

Figure 6.4.: Example trust path between two trust schemes. Trust scheme B recognizes trust scheme A and also publishes a translation from A to B. The verifier in scheme B can use this trust path to authenticate a attestation issued in scheme A, and execute the translation to understand trust data from scheme A.

6.3.2. Roles of trust schemes in a translation

We name the roles of the involved trust schemes from the perspective of the translation process.

- **Source scheme** is the trust scheme of the holder and their issuer. It is responsible for defining the authorities that, in turn, authorize issuers qualified in that scheme. One of these issuers issued the attestation we are going to verify. The verifier does not trust the source scheme directly.
- **Target scheme** is the trust scheme of the verifier. The verifier trusts the target scheme, and uses its infrastructure to retrieve the list of qualified issuers. Additionally, the target scheme is responsible for publishing the trust recognition and translation data.

For example, if country B—operating the target scheme—recognizes the trust scheme of some other country A, it publishes a trust translation *from* that trust scheme A to its own trust scheme B. From the verifier's perspective, the "target scheme" is the home scheme, and the "source scheme" is the foreign scheme.

6.3.3. Trust Recognition

If a trust scheme operator recognizes some other trust scheme, it publishes this statement in form of a trust recognition. A recognition is published by *the target* of the recognition since it represents the authority (the root of trust) for its own scheme. It is thus already trusted by all verifiers operating in its scheme. In contrast, the authority of the source scheme is not trusted by a verifier; thus the verifier does not trust a recognition published by that other scheme.

For example, if scheme B recognizes scheme A, this recognition requires a translation from scheme A to scheme B ($A \rightarrow B$). This recognition is hence published by scheme B, as illustrated in Figure 6.4.

The recognition of a trust scheme by another trust scheme leads to a trust model with cross certifications [Gas+89; Per99], as both scheme authorities are the respective root of trust in their scheme. This removes the need for a root of trust shared by all trust schemes.

If the two schemes are equivalent, the sole existence of the recognition is already enough for a verifier in the target scheme to work with an

attestation issued in the source scheme. It can use the now-trusted authorities of the other scheme to authenticate the attestation's issuer. If the schemes are not equivalent, the verifier first needs to translate the trust data of the attestation and its issuer.

6.3.4. Trust Data

The trust data of an attestation specifies the requirements for trust scheme entry. Those are the requirements that the holder needs to fulfill to receive the attestation. Trust data is either directly attached to that attestation or can be retrieved by the verifier from the trust scheme. The structure of the trust data depends on the type of the trust scheme (see Section 6.1). Trust data is encoded in different ways depending on the type of scheme and the relevant trust scheme requirements. Also, the semantics of trust data depends on the scheme. But, a verifier needs the trust data in an encoding, and with semantics it understands. Therefore trust data is the subject of the trust translation process.

6.3.5. Translation Data

The trust translation is a function to translate trust data between two schemes. The goal is to translate trust data of the source scheme's type and semantics to the target scheme's type and semantics.

The operator of the target scheme is trusted by all verifiers inside that scheme. Since it manages legal recognition between countries, it also defines this trust data mapping and encodes it in the form of *trust translation data*. The trust scheme then publishes this translation data alongside the corresponding recognition. As also shown in Figure 6.5, the input to the `translate` function are trust data (issued by the source scheme) and the corresponding translation data (published by the target scheme). The output of the function is trust data (understood by entities in the target scheme).

For publication, the translation function is formulated as a table. Table 6.1 illustrates a trust translation data table for a simple translation. The first column lists all possible trust scheme requirement combinations as trust

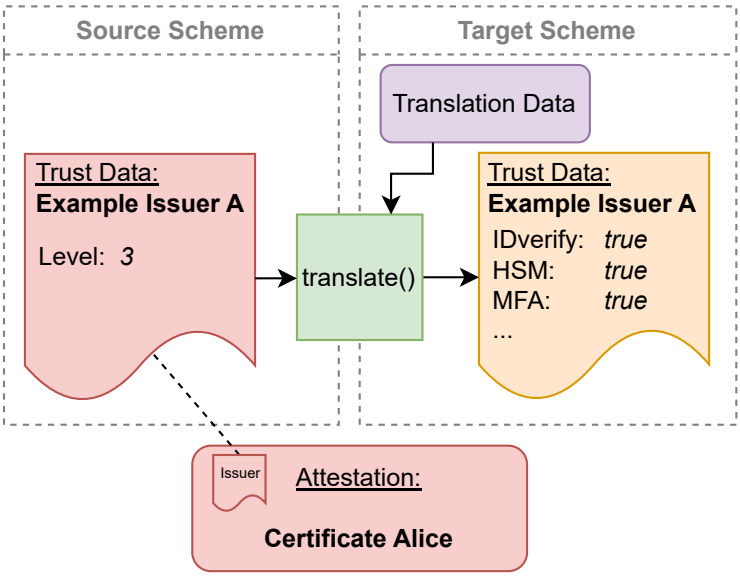

Figure 6.5.: Example trust data translation process. The trust data describing Alice's attestation is translated from a ordinal scheme into a tuple based scheme.

data for the incoming attestation.[9] In the case of the example, the source scheme is an ordinal scheme, so this list consists of all possible levels. The second column maps this levels to a set of tuples of the target scheme.

In the Table 6.1 example, the highest level, 3, requires that an issuer verifies the holder's legal identity (*IDverify*), and that the holder uses an Hardware Security Module (HSM) to store the attestation key material (*HSM*), and has performed multi-factor authentication (*MFA*). An example translation executing this translation data is shown in Figure 6.5. This table is then encoded in machine-readable form (e.g., as XML or JavaScript Object Notation (JSON)) and published by the trust scheme. Since trust translation data published by a trust scheme authority can be used by all entities operating in that trust scheme, the translation data needs to be encoded in a data format understood by all entities in the scheme.

There is only a mapping to some combinations of tuples since other

[9] If the source scheme is a tuple based scheme, in this naive approach the table can become quite large. In that case it makes sense to only specify the tuples and their values required to reach a certain level, and leave all other tuples open. This is also the case for mapping a tuple based scheme into another tuple based scheme.

Table 6.1.: Example trust translation data for a translation from an ordinal scheme into a simple tuple-based scheme.

Source Scheme	Target Scheme
Level: 1	IDverify: true
	HSM: false
	MFA: false
Level: 2	IDverify: true
	HSM: false
	MFA: true
Level: 3	IDverify: true
	HSM: true
	MFA: true
...	...

combinations of requirements are not possible in the source scheme. For example, in the source scheme, there are no attestations issued without identity verification. The table for the other direction of the translation (formulated and published by source scheme) might look different. And, multiple combinations of tuples could be mapped to the same level.

Special case: equivalent schemes If the semantic meaning of all levels/tuples is the same in both schemes, the schemes can be considered *equivalent*. In that case, the translation function is the identity map, and thus the trust recognition can be published without trust translation data.[10] But: the translation of trust data between two schemes of the same type is not automatically an identity map, as different levels could have different semantics in each scheme.

6.3.6. Recognition and Translation Process

Our approach can be split into three phases:

1. Initially, two trust scheme operators negotiate a legal agreement and compare their trust schemes. They also agree on a translation between their schemes and formulate this translation of trust data.

[10] However, to prevent an adversary from hiding a translation, it is important to also publish a statement about the non-existence of a translation. A verifier can then consider two schemes equivalent only if this information exists.

2. The trust scheme operators encode the trust recognition and translation in machine-readable form and publish it.

3. Later, a verifier receives an electronic transaction.

To be able to verify it, the verifier retrieves the trust recognition and translation data, and executes the translation.

Phase 1 **Legal Agreement:**

The preparation of a trust recognition starts with a negotiation phase between the two trust scheme operators (e.g., two governments). The details of this phase depend on legal circumstances but might include a feasibility study of the planned recognition. Further, a self-assessment of the individual trust schemes could be conducted. Next, the scheme operators need to ensure technical compatibility between their schemes. While agreeing on a technical standard to encode attestations is preferable, we discuss alternatives to this limitation in Section 6.5.4.

Mapping Trust Scheme Data: To formulate a trust translation, the scheme operators then compare the characteristics of their schemes. For example, they both map the characteristics of their schemes to a unified data model. A possible data model for mapping trust schemes was established by Wagner et al. [WKR19]. Or, they use a process like the eIDAS Article 14 Mutual Recognition check-list, which maps the 3rd party requirements to the eIDAS requirements [CEF21]. The result of this process is a mapping between the two schemes. If there is a direct mapping between all tuples of the two schemes (i.e., the mapping is the identity map), they can be considered equivalent. In that case, only the recognition itself—and no translation data—is needed. If the schemes are not equivalent, a mapping between different trust data is created. The scheme operators then individually formulate their mapping as trust translation data.

Mutual or Unilateral Recognition: While these steps describe a mutual recognition of two schemes, it is also possible that only one scheme recognizes the other. Similarly, it is possible that the schemes mutually recognize each other, but use different translation rules. Thus, the outcome of the first phase might differ between the two schemes, but the following phases are performed in the same way.

Phase 2 **Publication:**

After formulating their trust recognition, the trust scheme operators

publish it (using the DNS-based infrastructure introduced in Section 6.2). They also encode and publish the trust translation data. Each trust scheme individually performs this phase. The encoded translation data is a machine-readable document in a format that all involved trust schemes understand.

In our implementation, we serialize this table as a JSON file. This file contains a list of `trust-recognition` JSON objects, where each line in the table is represented by one object in the list.[11]

The operator then signs this JSON document to ensure its integrity and authenticity. It uses the same key it uses for signing the trust scheme's list of authorities.

Next, the trust scheme operator publishes the recognition and translation data at a DNS name known by all entities in the trust scheme. The domain name is constructed by combining the DNS identifiers of both schemes (i.e., *<source-scheme>._translation.<target-scheme>*). At this location, the (target) scheme publishes a PTR record pointing to the DNS identifier of the source scheme. Additionally, it publishes a URI record with a HTTP link to the trust translation data document.[12] Both the URI and the PTR records are signed using the DNSSEC key of the scheme's zone. The signing key used by the (target) scheme to sign the translation data is already published in a TLSA record at `_scheme._trust.example.com` (see Section 6.2.2). The result of a translation publication process is a DNS zone, as shown in Listing 6.3. Together with the publications from the previous section, this results in a trust path like shown in Figure 6.6.

Phase 3 **Verification & Trust Translation:**

The verification process starts when a service provider receives an attestation as part of an electronic transaction.

Local Trust Scheme: To ensure the authenticity of the attestation, the verifier first extracts the information about its issuer. It uses the issuers' trust scheme membership claim to assess if the issuer is indeed a member of its local scheme. If the verifier can verify the membership claim, it proceeds with verifying the signatures of the trust chain and executes

[11] An example trust recognition encoded as JSON is shown in Listing 6.4 below.

[12] If the schemes are equivalent, no URI record is needed. In that case, NSEC3 records are used to prove the nonexistence of the URI record (see Section 4.4). This prevents an adversary from hiding a translation, which would "promote" a source scheme to be equivalent even if it is not.

```
pof.org._translation._trust.example.com. IN PTR pof.org.
pof.org._translation._trust.example.com. IN URI 10
                                      ↪  https://tta.example.com/pof.org.json

pof.org._translation._trust.example.com. IN RRSIG PTR 13 6 11 20240501000000
                                      ↪  20180501000000 22222 example.com.
                                      ↪  <signature...>
pof.org._translation._trust.example.com. IN RRSIG URI 13 6 10 20240501000000
                                      ↪  20180501000000 22222 example.com.
                                      ↪  <signature...>
```

Listing 6.3.: Example DNS records of a trust recognition and translation published by the *example.com* scheme (recognizing the *pof.org* scheme). The records are signed using example.com's zone signing key (ZSK). TTL fields and DNSSEC records omitted for clarity.

the service provider's trust policy. Otherwise, the verifier proceeds to the trust recognition process.

Discover Trust Recognition: The verifier uses the issuer's trust scheme membership claim to query its own trust scheme authority for a trust recognition to the specified scheme. In that case, the verifier queries for a PTR record at the recognition domain name constructed from the DNS identifiers of its trusted scheme (target scheme) and the schemes from the issuer's claim (source scheme), i.e., *source-scheme._translation.target-scheme*. If such recognition exists, the verifier attempts to also retrieve the corresponding trust translation data from the scheme. This is done by querying the URI record at the same domain name and verifying the DNSSEC record. If such a URI record does not exist,[13] there is no translation mapping between the schemes, and hence the schemes are equivalent. If a URI record exists, the verifier uses its value to retrieve the translation data (the signed JSON published in phase 2). It then authenticates this data using the public key of the target scheme (retrieved from the scheme's TLSA record). If the signature verification succeeds, the foreign scheme is trusted. Using this recognition, the verifier established trust in the foreign scheme's authority.

Verify (Foreign) Trust Scheme Membership: The verifier then queries the DNS of the now trusted authority of the foreign scheme for the issuer's certificate. This is done in the same way as with a local issuer (see

[13] the nonexistence of a DNS record is verified using the NSEC3 record of the zone [Lau+08], see Section 4.4.

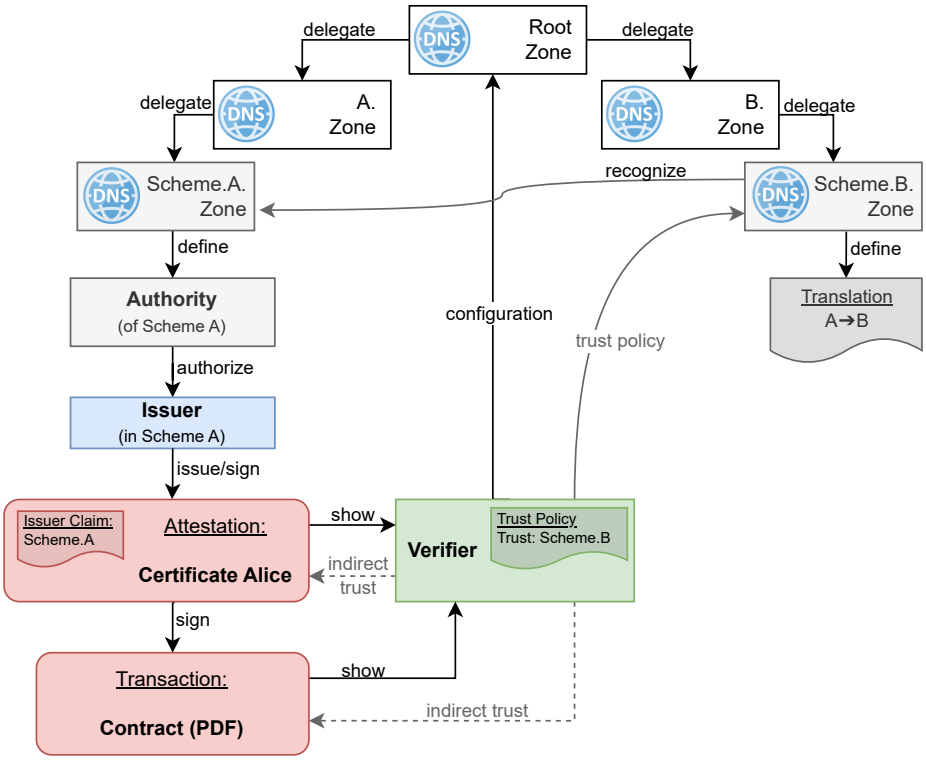

Figure 6.6.: Example trust recognition and translation published in DNS. The verifier trusts Scheme B (on the right), but the transaction it received was issued in Scheme A (on the left). Since Scheme B recognizes Scheme A, it publishes this recognition and the corresponding translation data.

Section 6.2.2). If the authority returns the correct certificate, the issuer's membership claim is confirmed. Next, the verifier uses the retrieved issuer certificate to verify the signature on the attestation. If the schemes are not equivalent, the verifier executes the retrieved trust translation.

Execute Trust Translation: The verifier retrieves the trust data of the attestation, either directly from the attestation (see Section 6.3.4), or using the (now trusted) source scheme. This trust data uses the source scheme's format and also follows its semantics. Thus, the verifier cannot directly use it in its trust policy and needs to translate it. The translation is performed by applying the translate function on the trust data with the previously retrieved trust translation data. The translation process is a

simple table lookup: the verifier iterates over the list of trust recognitions and compares each source scheme specification (i.e., the left side of the translation data table) with the trust data at hand. If it finds a match, it returns the corresponding target scheme specification (i.e., the right side of the table).

The result of this translation process is trust data in the format of the target scheme. Since the verifier now understands the data, it can load its trust policy and executes it on the trust data, thereby assess the trustworthiness of the attestation.

6.4. Reference Implementation

> This section is based on the author's work in the Horizon 2020 project LIGHTest. The described software was developed in collaboration with other members of the LIGHTest consortium.

We implement our trust publication and recognition approach to study it further. This implementation also demonstrates the feasibility of our approach. Building on the abovementioned concepts, we develop several software components to assist trust scheme operators, issuers, and verifiers.[14]

Trust Scheme Publication Authority (TSPA)

We provide a tool to support trust scheme operators with the publication and maintenance of their trust scheme information. This tool can also be used by issuers to publish their trust scheme membership claim. The TSPA provides a high-level API offering the (re-)publishing of trust schemes. It takes the desired trust scheme identifier and a trust scheme information file as input and publishes it in the DNS while abstracting away the technical concepts of the publication. To do so, the TSPA connects to both a webserver and a DNS API. We also provide this DNS API to manage trust information published in signed DNS zones. Once configured, the DNS API takes trust scheme publications or membership claims as input and creates signed zonefiles for the NSD nameserver.[15]

[14] The source code of all software components is available on GitHub: `https://github.com/H2020LIGHTest`

[15] `https://www.nlnetlabs.nl/projects/nsd/about/`, accessed on 2023-03-13

Automated Trust Verifier (ATV)

We implement the Automated Trust Verifier (ATV) tool to automatically discover, retrieve, and authenticate published trust scheme information. Further, we extend the ATV to retrieve trust recognitions and execute trust translations. To enable verifiers to define which trust scheme they trust, we develop and integrate the flexible and extensible trust policy system TPL, which we introduce in Chapter 7 below.

In addition to a REST API and a Java library, the ATV also provides a GUI. The ATV GUI takes as input an electronic transaction and a TPL trust policy, as shown in the screenshot in Figure 6.7. An electronic transaction is a file with an XML, Cryptographic Message Syntax (CMS), or Portable Document Format (PDF) advanced electronic signature.[16] The ATV automatically parses the transaction, retrieves all required trust information, and executes the trust policy. Additionally, the ATV verifies all DNSSEC from the root zone down to the TSPA zone. By doing so, it automatically establishes a trust path from the verifier to the transaction (see also example scheme publications in Figure 6.3 and Figure 6.6).

Trust Translations

A trust scheme operator publishes recognitions of other schemes in the same way they also publish the scheme information itself. If the schemes are not equivalent, they also publish translation data. Both recognition and translation are published using the DNS tool mentioned above. In our implementation, we serialize the trust translation data table as a JSON file. An example of trust translation data encoded as JSON is shown in Listing 6.4.

In our implementation, the service provider uses the verification tool ATV to authenticate attestations. Additionally, the SP uses the trust policy system TPL to codify its rules about trust conditions, such as which trust scheme it considers trustworthy (see Chapter 7). The trusted scheme is specified by its DNS identifier. In the trust policy, the SP also activates trust translations. An example TPL trust policy with enabled trust translation is shown in Listing 6.5.

[16] XML as XAdES, PDF as PDF Advanced Electronic Signature (PAdES), or a ZIP container in Associated Signature Container (ASiC) format.

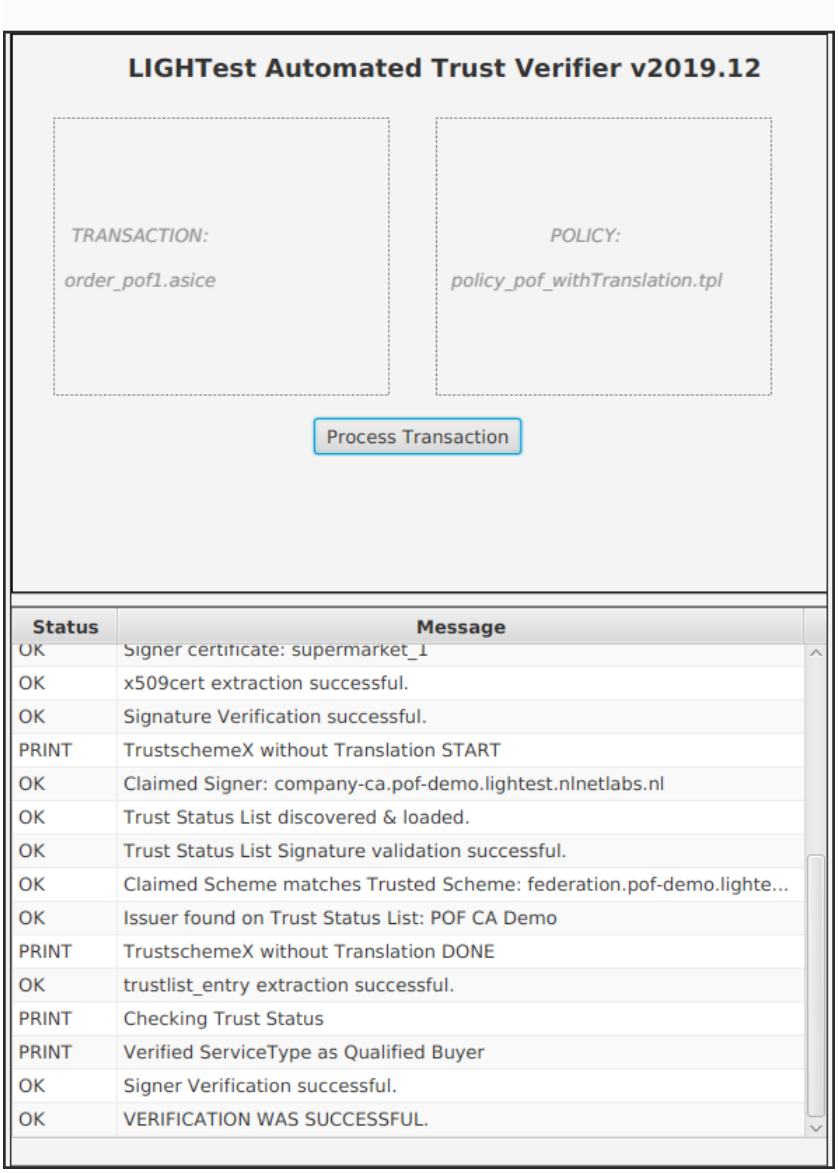

Figure 6.7.: Screenshot of the Automated Trust Verifier (ATV) tool. The ATV executes the SP's trust policy and uses the DNS-based trust scheme publication system to authenticate an order. The order is stored in an ASiC container signed by a supermarket.

```
{
 "trust-recognitions": [
  {
      "source": {
       "name": "pof.org",
       "provider": "Pumpkin Seed Oil Federation",
       "level": "3"
      },
      "target": {
       "name": "example.com",
       "provider": "exampleCom Scheme",
       "params": [
        {
         "value": "ServiceTypeIdentifier",
         "name": "ETSI/CA/QC"
        }
       ]
      },
      "name": "pof-to-example-esig",
      "creation-date": "2023-02-13",
      "activation-date": "2023-02-23",
      "status": "active",
      "expiry-date": "2023-12-24"
  },
  // further trust-recognition objects
 ]
}
```

Listing 6.4.: Example (unsigned) trust translation data published by the operator of the exampleCom trustscheme.

```
accept(Transaction) :-
  extract(Transaction, signer_cert, Certificate),
  check(Certificate, format, x509cert),
  extract(Certificate, pubKey, PK),
  verify_signature(Transaction, PK),
  check_qualified(Certificate).

check_qualified(Certificate) :-
  extract(Certificate, issuer, IssuerCert),
  % look for path from trusted exampleCom scheme
  % (logical OR: only one `trust` predicate needs to succeed)
  trust(IssuerCert, exampleCom, TrustData),
  % apply policy to (now-local) trust data
  verify_service_type(TrustData),

  extract(TrustData, pubKey, PkIss),
  verify_signature(Certificate, PkIss).

trust(IssuerCert, TrustedScheme, TrustData) :-
  % → Attempt to establish trust with local trust scheme
  % extract trust scheme membership claim
  extract(IssuerCert, trustScheme, Claim),
  % retrieve issuer's trust data from local scheme
  trustlist(Claim, IssuerCert, TrustData),
  % check if scheme of issuer is trusted
  trustscheme(Claim, TrustedScheme).

trust(IssuerCert, TrustedScheme, TrustedData) :-
  % → Attempt to establish trust using recognition/translation

  % extract trust scheme membership claim
  extract(IssuerCert, trustScheme, Claim),

  % encode DNS identifier for translation
  encode(Claim, TrustedScheme, TTAdomain),

  % query translation DNS and retrieve translation data
  lookup(TTAdomain, TranslationData),

  % retrieve issuer's trust data from foreign scheme
  trustlist(Claim, IssuerCert, TrustData),

  % execute translation on issuer's trust data
  translate(TranslationData, TrustData, TrustedData).
```

Listing 6.5.: Example TPL trust policy including an explicit trust translation to target scheme *exampleCom*.

6.5. Evaluation & Discussion

In this section, we first present a functional and a qualitative evaluation of our approach. We then briefly discuss legal aspects. We further discuss several additional aspects of our approach, namely the questions of translating attestation formats and whether transitive trust recognitions are a good idea. We further suggest the use of NAPTR records as an alternative to URI records.

6.5.1. Proof of Concept Demonstrator

Our implementation shows the technical feasibility of the DNS-based trust scheme publication approach.

Pumpkin Seed Oil Federation: In order to test our implementation, together with LIGHTest project partners, we built a testbed using the components described in Section 6.4 above.[17] To simulate a trust scheme, we configured a TSPA on a cloud server alongside a DNSSEC-enabled DNS nameserver and used it to publish several qualified issuers. Further, we used this method to simulate two additional trust schemes of different type. In specific, we simulated a "Pumpkin Oil Federation" (POF) trust scheme and a Turkish trust scheme. To evaluate our trust recognition approach, we published several trust translations. Additionally, we took the existing trusted list of eIDAS and imported it into our system as additional (simulated) trust scheme we publish. This allowed us to use real trust services (i.e., the Austrian mobile phone signature) to generate test attestations.

We then set up a SP using an ATV instance on a laptop, which we configured to trust the POF scheme as well as schemes recognized by it. Using this ATV, we successfully verified several test attestations, which we created using issuers qualified in all three schemes.

Further, the approach was evaluated in the context of the PEPPOL eProcurement large scale pilot [Dou+19]. Additionally, it was discussed in the context of IOT and smart cities [Omo+19].

[17] https://github.com/H2020LIGHTest/PumpkinSeedOilDemo

6.5.2. Evaluation

In this chapter, we focused on Goals 1, 2, and 3 of our overall goals (see Chapter 5). In specific, we consider the following aspects:

- Aspects of Goal 1: Complex Transactions, Setup of Trust Anchor, Different Types of Trust Schemes
- Aspects of Goal 2: Global Trust Scheme Interoperability
- Aspects of Goal 3: Trust among Peers

We now discuss the evaluation of our approach against the tackled aspects:

Complex Transactions: The first goal was to facilitate different perceptions of trust for different SPs and use cases. For this it is necessary that individual trust schemes are not aggregated by a bigger scheme or validation authority, but instead published directly and in a queryable way. This has been achieved by enabling different trust schemes of different type to be published in the DNS. Those trust schemes contribute different perspectives to the trust assessment of a electronic transaction consisting of multiple attestations. It is optionally possible to also implement the subsidiarity principle, delegating the publication of a part of the trust scheme to another entity. To customize the trust requirements to their individual understanding of trust, SPs formulate trust policies that define a set of trusted schemes for specific transactions (see Chapter 7).

Setup of Trust Anchor: To authenticate electronic transactions, a SP always needs information about qualified issuers and a way to establish trust in this information (trust anchor). This is a tedious, manual process, which needs to be performed for each trust scheme that the SP trusts directly. Hence, the goal was to simplify the process and replace it with a single cryptographic trust anchor for all trust schemes. This has been achieved by using the (established) root of trust of DNSSEC as global trust anchor.

To set up a verifier that uses our DNS-based trust discovery approach, a SP needs to configure the human-readable DNS-identifier of the scheme it trusts. This is the only step that is needed for each trust scheme that

the SP directly trusts.[18] Additionally, as a pre-requisite, the SP needs to provide a DNS and DNSSEC setup on its machine. This involves the network address of at least one DNS resolver, as well as the DNS' root key signing key (KSK).[19] The retrieval of the root KSK and the identifiers of the trusted schemes requires a secure channel, although the KSK is often shipped with the operating system[20] and the scheme identifier is human-readable and can thus be more easily acquired. As an alternative, the SP can send the transactions and its policy to an ATV operated by an entity the SP trusts (this is similar to an eIDAS validation authority).

Different Types of Trust Schemes: In addition to supporting the use of multiple trust schemes in a single transaction, the goal was to support trust schemes of different types. After analyzing the three types of trust schemes (see Section 6.1), we design our trust scheme publication approach accordingly. We support the publication of boolean and ordinal trust schemes directly in the DNS, while we point to trust list entries for tuple based schemes. Using this information, SPs can use the trust characteristics of the individual schemes to define their policies. Further, we extended our approach by trust translations. This enables that SPs define their trust policy for a local scheme type, and then translate transactions from different schemes using our system (see Section 7.4.3).

Global Trust Scheme Interoperability: As we focus on trust management in a heterogeneous environment, our next goal was to take indirect trust in a scheme into account. For example, when a SP directly trusts scheme A, and scheme A recognizes scheme B, the SP can establish trust in transactions issued in scheme B.

In our approach, trust schemes that recognize other trust schemes publish this information in the DNS (see Section 6.3). By means of trust translations, it is further possible that trust schemes with different understandings of trust recognize each other. This is done by publishing a mapping between the scheme's trust data alongside the recognition. By

[18] We note that this trust-anchor approch could be adapted to the TSL world by publishing a global LOTL at a well-known location and using the established DNSSEC root of trust to sign this global TSL. The challenges on the root governance remain (see below).

[19] `https://www.iana.org/dnssec/files`, e.g., using `https://github.com/iana-org/get-trust-anchor`

[20] See also the discussion on direct vs. indirect trust in Chapter 2.

doing so, it is also possible to map between trust schemes of different type, e.g., from a boolean scheme into an ordinal scheme (see Section 6.1). While our approach uses the DNS for the publication and discovery, the underlying trust translation concept is technology-neutral. As such, it can also be used with other trust scheme models, e.g., with eIDAS, by publishing translation data in the TSL.

Related Work: The eIDAS regulation's Article 14 establishes a legal framework for the recognition of foreign trust schemes [Eur14, Article 14]. It establishes that if there is a mutual recognition agreement (MRA) between the EU and the third country, any trust service recognized in the third country shall be recognized as legally equivalent to qualified trust services provided in the EU. Article 14 is concerned with the recognition of trust services and does not consider, e.g., electronic identity schemes. On a technical level, this has been piloted by the Pilot for the International Compatibility of Trust Services.[21] The pilot imposes the requirement that third countries publish a trust list conforming to ETSI TS 119 612 . To simulate the recognition, the pilot publishes a simulated EU trust list[22] that contains a pointer to the trust list of the (simulated) third country. In turn, the third country adds a pointer to the eIDAS LOTL in its list.[23] Both pointers are published alongside a set of *MRA Information*[24] to describe the recognition. This information can for example be used to restrict the recognition to a specific type of trust service (e.g., qualified certificates for electronic seals). The MRA information can also map identifiers used in the trust list, but it is limited to technical identifiers and direct mappings (cf. Section 6.3.5). To facilitate the technical validation of Article 14, the EU also publishes a (fictional) "Third Countries List of Trusted Lists (LOTL)" recognizing the (real) trust list of Ukraine.[25]

Trust among Peers: While our trust publication approach discussed so far follows a hierarchical model (with side delegations [Per99]), our last goal was to also consider a peer-to-peer trust approach. We discuss our Web of Trust (WoT)-based trust publication system in Section 6.6 below. This system is not intended to provide qualified information, but instead facilitate the easy and trustworthy exchange of trust knowledge

[21] `https://eidas.ec.europa.eu/efda/intl-pilot/#/screen/home/demo`, accessed on 2023-06-21
[22] `https://eidas.ec.europa.eu/intl-comp/tl/lotl.xml`, accessed on 2023-06-21
[23] `https://eidas.ec.europa.eu/intl-comp/tl/tc-tl.xml`, accessed on 2023-06-21
[24] `http://ec.europa.eu/tools/lotl/mra/schema/v2`, accessed on 2023-06-22
[25] `https://eidas.ec.europa.eu/efda/tl-browser/#/screen/tc-tl`, accessed on 2023-06-21

between peers. Hence, it can be used to enrich the qualified information from our DNS-based system, or used separately. As this system allows the trustworthy publication of any information (not just trust information), it also serves as a potential basis for the trusted credential transformation framework introduced in Chapter 8.

6.5.3. Need for a Legal Framework

Any country that already operates its own trust scheme and is interested in implementing a trust recognition approach needs to establish a legal framework for trust scheme recognition. In addition to the mapping of trust data between schemes, rules for the operation of trust infrastructure are needed. These rules also need to define the liability of involved parties for damage caused intentionally or negligently. In the context of eIDAS, the regulation defines a liability model in Articles 11 (for electronic identities) and Article 13 (for trust services, such as electronic signatures) [Eur14].

Legal Aspects of our Approach: In contrast to eIDAS, there is no existing legal model for our approach, and establishing such a model for a global system remains a challenge.

Our reference implementation builds on the LIGHTest system, which uses the hierarchical DNS for trust data publication. While the DNS has a governance structure, it only defines rules and processes about the management and publication of standard DNS records [ITU05; IANb]. Using the DNS as a trust scheme root of trust has different legal implications. It thus requires that the government of the trust scheme establishes a legal framework that defines the liability of the country registry to establish trust in the Country Code Top-Level Domain (ccTLD). Additionally, an adaption to the governance structure of the DNS root zone establishing a similar liability model is needed. It is unclear who could create the legal basis for such a governance model. We note though that the most important requirement at the root level is that each ccTLD is delegated to an institution authorized by the respective country (operating the trust scheme). This requirement is not different than in the existing DNS governance model overseen by IANA [IANa], but has different implications and thus might require a different liability model.

Alternative Legal Frameworks: As an alternative, or for the time until interested governments establish such a legal framework, the LIGHTest project proposes an alternative framework directly between the involved parties [GJ19a; GJ19b].

We note that it is not always necessary that a government legally implements our approach for trust translations to provide value. For example, even if the government of Fantasyland is currently not interested in recognizing the eIDAS trust schemes, the business trade union of Fantasyland could nevertheless use its own trust scheme to publish a trust recognition to eIDAS. Since this recognition is not published by the government, it does not have legal value. However, it still reduces the transaction overhead between the two schemes by supplying local business with trustworthy information.

6.5.4. Attestation Format

In its basic form, our trust translation approach assumes that both trust schemes use the same standard for encoding attestations. This is required because the verifier needs to be able to parse and understand the attestation and the certificate of the signer. Since X.509 is an established standard that is commonly used by trust schemes, this is a realistic assumption [Hou+99; WKR19]. If another standard is used, but both schemes are using the same format (e.g., VCs), our approach can be applied as well.

If one scheme uses a different attestation format than the other, there is a lack of understanding of the attestation, which is an issue for trust verification. In that case, the verifier needs to translate the attestation as well. To do so requires translation information about how to transform the attestation from one format into the other. We introduce an approach to translate attestations between different file formats and schemas in Chapter 8. Additionally, the verifier requires information about how to verify the signature on the attestation in its original format since transforming it breaks the signature. This attestation translation information could be provided by the same party that publishes the trust translation information.

6.5.5. Transitive Trust Recognitions?

Trust transitivity is a property of trust where an entity accepts the trust assumptions of another entity about a third entity (if A trusts B and B

trusts C, then A trusts C). In the case of trust recognitions, this means that a verifier would not only trust a recognized scheme, but also all other schemes recognized by that scheme. While transitive recognitions—and even translations—are technically possible with our approach, we note that trust recognition transitivity is more complicated regarding recognition agreements. Just following along a transitive recognition can lead to unintentional trust in an issuer, i.e., trust that is not covered by the recognition agreement [CH96]. Since our approach explicitly resolves each trust recognition, is does not transitively resolve the recognitions of a recognized scheme.

An alternative to trust recognition transitivity is to negotiate a agreement with the third scheme and add a explicit trust recognition to that scheme. In the example above, scheme A could negotiate an agreement with scheme C directly and publish that recognition. A middle course is to allow trust recognition transitivity up to a certain depth of transitivity, i.e., limit the length of the path in the trust graph. This enables that A trusts all schemes recognized by scheme B, such as scheme C, but does thereby not trust schemes recognized by scheme C. Doing so requires adding a shared understanding of trust recognitions to the trust agreement.

6.5.6. NAPTR Records

In our architecture, we use URI records for the publication of trust scheme information in the DNS. These URI records map a DNS identifier to an (HTTPS) URI. An alternative to URI records for doing so are NAPTR resource records [MD00]. NAPTR records are commonly used in internet telephony to map telephone numbers to Session Initiation Protocol (SIP) addresses.[26] In contrast to a URI record, the response to a NAPTR query does not contain the final URI directly. Instead, the nameserver returns a regular expression. The verifier then uses this regular expression to construct the desired URI. Thus, NAPTR records could be used for the publication shortcut discussed in Section 6.2.2 without the need to explicitly publish separate records for each scheme level or issuer. While NAPTR records provide more flexibility, they also introduce complexity and were hence not used in our implementation.

An example NAPTR record is shown in Listing 6.6. The regular expression in this record informs the verifier about how to transfer a scheme identifier

[26] RFC 3401 generalizes NAPTR records further into the Dynamic Delegation Discovery System framework [Mea02].

into the URI of the corresponding trust scheme information. The `U` flag tells the verifier that the output of the regular expression field is a URI that the verifier can query directly.

```
_scheme._trust.exampl.com. IN NAPTR 100 10 "U" "trustscheme"
                          ↪  "!^.*$!https://tspa.example.com/scheme/\1.xml!" .
```

Listing 6.6.: Example alternative trust scheme publication using a NAPTR record. TTL field and DNSSEC records omitted for clarity.

6.5.7. Related Work

Automated Trust Verification: The Public Key Infrastructure (PKI) based on X.509 certificates (PKIX)—called *web PKI*—is the predominant authentication model on the internet [Hou+99; Dur+13]. In this hierarchical model, the list of Certificate Authorities (CAs)—the trust store—forms the root for trust chains binding an entity's identifier (e.g., domain name) to its cryptographic keys. Root CAs are only considered trustworthy if a client trusts them directly, which is usually achieved by bundling a set of root certificates with the operating system or web browser. While the current web PKI serves its purpose, it is limited to establishing the binding between a domain name and a cryptographic key.

The EU's eIDAS regulation and technical specification [Eur14] establishes a trust infrastructure between the EU's member states along with legal liabilities but is currently limited to Europe. eIDAS uses its own PKI, where each EU member state defines a Trust (Service) Status List (TSL), i.e., a list of all trusted certificate authorities (trust services) [SLL13]. The locations of those lists are in turn published by the European commission in the form of a LOTL, forming eIDAS' root of trust.[27]

Galal and Lehmann discuss the EU's eHealth Network specification (for the verification of COVID certificates). The EU specification proposes to outsource the complicated validation of certificates to trusted validation services, i.e., a trusted third party (TTP) [GL23]. Galal et al. further propose a privacy-preserving alternative to the EU's system (*PP-COV*)

[27] https://ec.europa.eu/tools/lotl/eu-lotl.xml

using non-interactive zero-knowledge proofs [GL23, Section 4]. The goal of these projects is to hide the user's attributes from the verifier (but reveal them to the validation authority). In contrast, we focus on revealing the (required) attributes only to the verifier without the need for a TTP.

The Trust over IP project combines DIDs, VCs and related technology and governance models into a stack compatible with our approach [Dav+19].

Trust Scheme Interoperability: Article 14 of the eIDAS regulation provides a framework for the recognition of trust services operated by trust service providers from a country outside of the EU (the "3rd country") [Eur14, Article 14]. This is relevant if the EU and the 3rd country have reached a legal agreement about that recognition. Article 14 covers only trust services (e.g., signatures and seals), but not electronic identities. A prerequisite for mutual recognition under Article 14 is that the 3rd country publishes a trusted list.[28] If recognized, a pointer to the 3rd country list would be included in the EU's LOTL, next to pointers to EU member states' lists. ETSI TR 103 684 is a technical report that studies existing trust schemes around the world and their possible mutual recognition in the EU [ETS20].

Wagner et al. propose a unified data model by consolidating the data models of nine existing trust schemes [WKR19]. It can be used to describe trust schemes in a unified way, and to compare different schemes to simplify mutual recognition.

The Futuretrust project [Hüh+16] proposed the concept of a global Trust Status List (gTSL) [Fut19a] to extend the EU's TSLs to institutions from outside the EU. This gTSL is hosted on a Distributed Ledger (DL); access is provided using a smart contract and write access is limited to governmental authorities. The stated goal is "[...] to manage and provide information related to qualified trust service providers across European Member States and beyond, extending the current TSL model [...]" [Fut17]. Additionally, it "[...] aims at decentralizing the current distribution scheme in order to improve its resilience aspects as well as manageability". Sellung et al. discuss further academic and industry perspectives of trust scheme interoperability [Sel+19, Section 8].

[28] The "CEF Pilot for the International Compatibility of Trust Services" recommends [CEF21] that this trusted list follows the same standard as the EU's lists (ETSI TS 119 612) [ETS16]. See also `https://eidas.ec.europa.eu/efda/intl-pilot`

FutureID and LIGHTest: In the FutureID project, Bruegger et al. consider a TSL-based infrastructure not scalable and thus unsuited for the implementation of a larger (e.g., global) trust infrastructure. They also state that trust lists and local trust stores "put a significant burden on relying parties who need to securely provision trust data (e.g., certificates or trust lists), keep them up to date, and query them for individual trust decisions" [BÖ14]. To mitigate this limitations, they propose to instead use the DNS with security extension (DNSSEC) as the base for a trust infrastructure.

The LIGHTest project[29] built on this ideas to develop a lightweight, global trust infrastructure, which enables automatic validation of trust based on the policy of a verifier [BL16; Roß17; Wag+19]. As such, LIGHTest utilizes the DNS with its existing global infrastructure, organization, governance and security standards. LIGHTest uses DNS to publish information protected by DNSSEC. It enables trust scheme operators to publish information (trust data) about their trust schemes. Further, LIGHTest provides an ATV which is capable of automated discovery and retrieval of the published trust information [Wag+19]. To enable verifiers to define which scheme they trust, LIGHTest uses the trust policy system TPL [Möd+19]. The author of this thesis participated in the LIGHTest project; Chapter 6 and Chapter 7 of this thesis build on ideas of the project.

Digital Object Architecture and X.1255: An alternative to DNS as identifier resolution system is the Digital Object Architecture (DOA) and its Handle system [KC88; KW06]. DOA consists of a set of components and protocols to facilitate the resolution of identifiers (handles) to the location of digital objects. Governance of DOA identifier registration, policy development, and operation is managed by the Digital Object Architecture (DONA) Foundation [Ala19, Section 9.3]. An example system based on DOA is the Digital Object Identifier (DOI) system well-known in, e.g., academic publishing. Similar to DNS, a DOA handle is structured in a hierarchical way,[30] allowing for delegation in the resolution process. Kim provides a more detailed comparison of DNS and DOA [Kim19]. DOA provides origin authenticity of the resolved objects by means of a built-in PKI, where the root key is stored in the root handle (`0.0/0.0`)

[29] `https://www.lightest.eu`, accessed on 2023-02-13

[30] A DOA handle has the form `prefix/suffix` (e.g., `10.3030/700321`), forming a left-to-right hierarchy where the prefix is globally unique. A DNS identifier forms a right-to-left hierarchy (e.g., iaik.tugraz.at).

and manged by the DONA Foundation. This root key is used to sign the handle records of keys lower in the hierarchy, similar to DNSSEC [Ala19, Section 8.6]. However, there is no documented mechanism on the management of the root key, nor how it could be rolled out in the future [Ala19, Section 8.6]. Based on DOA, the International Telecommunication Union (ITU) published recommendation X.1255 "to provide an open architecture framework in which identity management information can be discovered" [ITU13]. X.1255 is a framework that enables the discovery of identity information and its provenance [ITU13, Section 1]; as such, it can act as an alternative publishing system for our approach (cf. Section 6.2.4 and Section 6.3.6).

6.6. Alternative Approach using a Distributed Ledger

> This section is based on the paper *Trust Me If You Can: Trusted Transformation Between (JSON) Schemas to Support Global Authentication of Education Credentials* by More, Grassberger et al. [Mor+21]. Parts of this paper has been copied verbatim. The prototype was implemented by our student Peter Grassberger.

In this section, we discuss an alternative approach to mitigate the problem stated above by introducing a decentralized and open system to automatically verify the legitimacy of issuers. This infrastructure can also serve as a basis for the credential transformation system introduced later in Chapter 8.

We use a DL to enable all involved entities to publish trust information about credential issuers. This effectively forms a WoT, a directed graph of certifications published by institutions based on previous manual evaluations. This web of trust further enables verifiers to define their own policies on how to automatically evaluate a (previously unknown) entity's legitimacy.

In contrast to the architecture discussed in the previous sections, in this approach, trust in a credential issuer is not established by an authority. Instead, entities participating in the system issue statements about their trustworthiness about each other. Thus, the role of an authority does not exist here. Instead, entities that are issuing trust statements about other entities are acting in the role of a so-called "trust certifier". Since

the system works on a peer-to-peer basis, an entity can act in several roles. Correspondingly, entities that act as trust certifiers can also act as credential issuers or verifiers.

A trust certifier is usually attesting the trustworthiness of a credential issuer. The meaning of trustworthiness depends on the context of the system. For example, in the education diploma use case, universities issue digital diplomas in the form of credentials to students (holders). During a student mobility process, those credentials are then received by other universities. The receiving universities need to assess if the issuer of the diploma credential is a legitimate university—usually accredited by some government body. In an international context, assessing this legitimacy is a non-trivial task. In our approach, the universities can rely on statements published by their (trusted) peers. While such peer-issued information has no direct legal value, it can assist universities in their assessment process.

6.6.1. Concept

To help verifiers in their assessment process, entities in our system provide statements on the trustworthiness and legitimacy of others—so called "trust statements".

A **Trust Statement** is a tuple of the form $\langle C, \mathsf{LL}, \mathsf{LC}, \mathsf{u}, \sigma, \mathsf{t}, I \rangle$, where C is the trust certifier, I is the credential issuer being certified, $\mathsf{LL} \in [-1, 1]$ is the level of legitimacy C asserts for I, $\mathsf{LC} \in [-1, 1]$ is the level of confidence C has in the statements published by I, u is the identifier of what I is trusted to issue (e.g., type of credential), and t is a timestamp. Additionally, σ is a cryptographic assurance (digital signature) of the certificate's integrity and authenticity. Depending on the system chosen, levels are allowed to be any real value in the interval, one of several discrete values, or even just in $\{-1, 0, 1\}$ (with 1 denoting the highest trust level).

The Web of Trust is a collection of trust statements.

A **Web of Trust** is a directed, edge-labeled multigraph $W = \langle V, E \rangle$, where the vertices V are entities (e.g., educational institutions) and the edges E represent trust statements, which are consequently labeled as stated in the definition above. For edge labels in a web of trust, the trust certifier–credential issuer parameters have to match the edge's vertices, i. e. for each edge e from v_1 to v_2, the corresponding label of e has to be of the form $\langle v_1, \mathsf{LL}, \mathsf{LC}, \mathsf{u}, \sigma, \mathsf{t}, v_2 \rangle$. An example web of trust graph is visualized in Figure 6.8.

The WoT graph is maintained by a **Registry**, a data structure on the DL. The registry is implemented as a smart contract (SC). This results in a single point of contact for trust certifiers and verifiers who want to access or append to either graph by calling the *get* or *add* function on the contract, respectively.

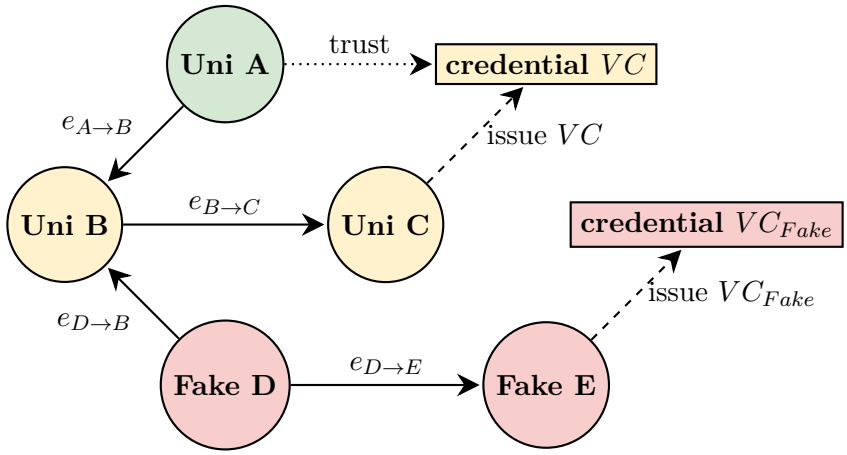

Figure 6.8.: Example Web of Trust with Uni A acting as verifier, Uni B acting as trust certifier, and Uni C acting as legitimate issuer of a credential VC. A credential VC_{Fake} issued by Fake E is not trusted by Uni A, because there is neither a trust path from Uni A to Fake E nor to Fake D. The edge $e_{D\to B}$ was issued by Fake D and is not part of the (directed) trust path.

6.6.2. Process

The first step needed to check a credential is to verify the legitimacy of the credential and its issuer using information published by other entities.

Our approach can be split into four phases: 1. Initially, the registry and its smart contract are established on the DL. 2. The trust certifier identifies a credential issuer and publishes the certification information using the registry. 3. The issuer issues a credential to a holder. 4. The holder shows the credential to a verifier, who uses the information stored in the registry to verify the credential and the credential issuer's legitimacy. These phases are also visualized in Figure 6.9.

Phase 1 **Setup:**

The registry smart contract provides a single point to create or update (*add*) and retrieve (*get*) the lists of edges (trust statements). Additionally, holders, credential issuers, and trust certifiers create their own key pairs and use them to establish their self-sovereign identities, i.e. derive and register Decentralized Identifiers (DIDs). These DIDs with corresponding DID documents are registered on the DL to enable other parties to retrieve the respective public keys for signature verification processes.

Phase 2 **Issue Trust Statement:**

After having assessed an issuer's legitimacy (e.g., in a previously performed tedious manual process or using other channels), a trust certifier may share its decision with others by issuing a trust statement. Based on the previous assessment, the certifier chooses an appropriate level of legitimacy LL, which may also be a negative value if the certifier concludes that the credential issuer is an illegitimate institution.[31] In addition, it adds the level of confidence in trust statements published by the issuer, represented by LC. The certifier publishes its assessment as a trust statement on the registry by setting the variables on the edge accordingly, using its private key to sign the edge. It publishes the signed edge by calling the smart contract through one of the DL's nodes. This creates a new edge in the registry's WoT graph pointing from the trust certifier to the prospective credential issuer.

Phase 2.1 **Revoke Trust Statement:**

As trust in other institutions changes and faulty entries occur, trust certifiers are able to add a new edge between the same vertices (but with a more recent timestamp) to the WoT, thereby altering the stated level of legitimacy or other attributes. Based on the timestamps, verifiers select edges at points in time based on the received credential and their trust policy.

Phase 3 **Issue Credential:**

At some point, a (prospective) holder requests a credential, for example, for some accomplishment. Initially, the holder proves their identity using their DID and private key. Then, the issuer loads the holder's attributes

[31] Optionally, the certifier adds a textual description of its decision to help a verifier with its assessment.

from its information system, encodes those attributes using a suitable JSON schema, and places the encoded attributes into a credential. The credential issuer finally signs the credential and hands it to the requesting holder for their own use. Optionally, the issuer registers the credential by publishing its fingerprint on a separate smart contract.[32] This is done to prove the existence of the credential at a certain point in (ledger) time. Otherwise, the issuing of credentials is independent of the registry, and a DL may only be needed to retrieve the holder's public key via its DID for authentication.

Phase 4 **Verify Credential:**

When a holder presents a credential to a verifier, the verifier needs to verify it according to its trust policy. The verifier first retrieves the registry's WoT graph by calling the smart contract's *get* function using a DL node. In this graph, the verifier performs the path-finding algorithm defined in its policy. The verifier also verifies the signature of each trust statement in the trust paths with public keys obtained from the DL. If there are multiple paths toward a credential issuer, the verifier's trust policy may select which paths are relevant. This enables different verification scenarios, such as "Was the issuer trusted at the time the credential was issued?" and "Is the issuer trusted right now?".[33] On the identified paths, the verifier computes an overall "legitimacy score" and compares this score with the policy's requirements. Next, the verifier verifies the credential's signature with the public key that has been registered in the DL for the issuer's DID, including a revocation check by consulting the respective revocation registry. The verifier concludes the process by checking the credential with regard to a set of rules stated in the trust policy, such as the specific field of study or certain grade requirements.

6.6.3. Discussion

Revocation of trust statements is an essential feature in any trust management system. In our proposed system, a trust certifier can revoke a trust statement at any time by issuing a new one with a reduced legitim-

[32] However, depending on the content of the credential, the hash digest itself might constitute PII and thus publishing it on a DL is not legally possible.

[33] If the signing-time is added to a credential by the issuer (i.e., the credential is not registered on the DL, and there is no trusted timestamp), a compromise of the issuer's private key also destroys the trustworthiness of the issuance information. In that case, a issued credential is never trustworthy.

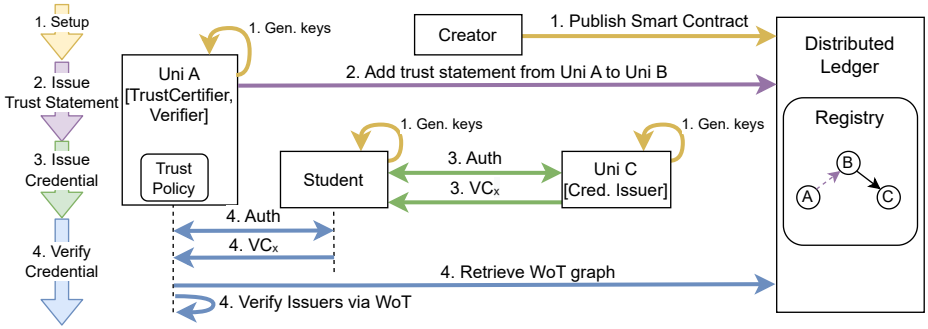

Figure 6.9.: Architecture and example dataflows of our DL-based WoT system. University A, acting as trust certifier, adds an edge stating its trust about University B. Later, University A, acting as verifier, verifies a credential issued by University C. It does so by additionally obtaining a previously registered edge from university B to C from the registry smart-contract.

acy level—even with a negative value to explicitly declare mistrust in the credential issuer. An issuer can also add a trust statement about itself, enabling self-revocation. Additionally, an issuer can revoke a credential after it has been issued, e.g., by publishing a corresponding statement to a revocation registry, which is checked by the verifier. Checks for both types of revocations need to be performed during credential verification by the verifier.

Smart Contract Security: Depending on the access model of the DL, many or even all entities can write data into the registry. Since all information published in the registry is signed by its publisher, this is not a problem for the authenticity of trust statements. Nevertheless, an overfull registry could lead to performance issues during verification. To mitigate this issue, the smart contract serving the registry can be equipped with access control mechanisms. For example, it is possible to enforce that an issuer can only add edges that are outgoing from its own vertex. Preventing an attacker from adding invalid edges starting at the vertex of a legitimate entity keeps the path-finding algorithms from having to verify many invalid signatures to find the valid ones. Furthermore, it is possible to only allow adding edges to trust certifiers already part of the graph. While this keeps attackers away, it also hinders legitimate entities from joining the graph. Thus it depends on the concrete use case which mechanisms should be used.

Since a smart contract is code and code could contain bugs that malicious parties might be able to exploit [Tor+20], it is important to apply secure software development practices when adding access control mechanisms. This is even more important when considering that smart contracts cannot be changed after they have been deployed on a DL [Rod+19; Rod+21].

Operational Costs: To evaluate our concept, we deployed a smart contract that maintains a registry, which manages the graph represented as a list of edges. The smart contract does not perform additional checks to verify the publisher, so a verifier needs to retrieve the complete graphs and filter out any irrelevant or invalid edges. While a public DL enables an open system, all write operations on the ledger have a cost. In the Ethereum context, the cost of a contract call is measured in units called *gas* and depends on the required computational effort. Adding an edge to our registry costs about 78,000 gas, worth about US$ 2 in March 2023.[34] In contrast to write operations, read operations are free. As most operations in our system are of the latter kind, the total costs are still relatively low.

Since the costs of using a public ledger are hard to predict in advance, private or consortium ledgers represent alternatives. In such ledgers, members of a consortium operate all nodes of the ledger, removing the need for an incentive system like gas. This limits the expenses to the costs needed to host the nodes but restricts (write) access to consortium members.

Sybil Attacks and Censorship: The existence of fake issuers issuing (fake) credentials also make the existence of fake trust certifiers issuing trust statements to (fake) issuers plausible. However, since such fake certifiers are neither trusted by a verifier nor have a trust path from the verifier to them, these (fake) trust statements do not influence a credential's verification (see Figure 6.8).

Although the decentralized and distributed nature of DLs provides resistance against censorship and denial-of-service attacks [Ale+17], a node might still provide bogus information to a verifier. Hence a verifier needs to establish a trust relationship with at least one node. One way of doing this is having the verifier operate its own node. Another way is to ask multiple or even all DL nodes to attest a certain set of edges represents the

[34] Computed by multiplying the gas price (`https://etherscan.io/chart/gasprice`) and ether price (`https://etherscan.io/chart/etherprice`) using `https://petertheone.github.io/gasPriceFiat` on 2023-03-21.

full graph and that no edges were censored. While the first case increases the operational effort, the latter case introduces a performance overhead. It is thus important to balance the number of prompted nodes with use case-specific censorship-resistance requirements. To query several nodes of the DL, the ledger attestation system introduced in Chapter 10 can be used.

Chapter 6 Conclusions

In this chapter, we presented the results of our work in the field of trust management in a heterogeneous environment. We introduced our DNS-based architecture that enables trust scheme operators to publish information about their trust scheme. Using this published information and our trust recognition approach, a verifier can automatically verify attestations and transactions, regardless of origin. We extended this trust recognition approach with trust translations, which enables the use of trust information even if the foreign trust scheme uses a different understanding of trust. We also presented a reference implementation of our approach and discussed severally of its aspects. Further, we looked into novel trust management techniques by also discussing the use of a distributed ledger for web of trust-based trust publication. The results of the research presented in this chapter have been published in three academic papers [Wag+19; Mor+21; Mor23].

7

Expressive Trust- and Access-Policies

This chapter is based on the papers *TPL: A Trust Policy Language* by Mödersheim, Schlichtkrull, Wagner, More et al. [Möd+19] and *Adapting the TPL Trust Policy Language for a Self-Sovereign Identity World* by Alber, More, Mödersheim et al. [Alb+21]. Parts of those papers have been copied verbatim, making this chapter a revised version of the original papers. The introduced TPL system was designed in collaboration with DTU Compute. The TPL interpreter was implemented by Lukas Alber. Additionally, the SSI extension was implemented as part of the master's thesis of Bernhard Zenz.

When Service Providers (SPs) receive an electronic transaction, they need to ensure it is trustworthy before processing it further. Such an electronic transaction are documents that are either human-readable (i.e., PDFs) or machine-readable (e.g., XML or JSON files) with business-specific content. To establish trust in these documents, they are accompanied by an attestation, for example, an identity assertion or a certificate. This attestation is cryptographically linked to the document, and issued by a trust scheme. The SPs then uses a trust scheme it trusts to establish trust in the transactions.

But, the specific understanding of trust—and trustworthiness—depends on factors like country, context, and risk. Correspondingly, the rules describing the trustworthiness of a transaction vary for different SPs and use cases. Directly encoding these trust rules in trust management systems' source code is possible, but this increases the effort to customize them to an SPs needs. However, without these customizations, assessing the trustworthiness of diverse transactions remains a manual effort. A more flexible solution is to encode the trust rules in a separate file—a so-called *policy*.

Electronic policies support the automated processing of electronic transactions. In this context, we differentiate between *Trust Policies* and *Access (Control) Policies*. Trust policies enable users to define their own conditions for trusting a transaction and mechanics for discovering trust information needed to authenticate attestation holders. For example, an SP might require a qualified eIDAS signature for purchases of expensive goods. Once trust in a transaction has been established, access policies are used to define business logic-specific acceptance criteria. A simple example of such an access policy rule is an age check. Another example is the limiting of offered products depending on the type of the buyer, who was previously authenticated using the trust policy (we discuss a concrete example of this in Section 7.6).

Introducing TPL: Our global trust management system presented in Chapter 6 enables verifiers to encode their trust and access rules as a policy. To do so, in this chapter we introduce *TPL*—our trust- and access-policy system. TPL can represent complex policies and facilitates the discovery of trust status information. With TPL, an SP only needs to specify the DNS identifier (see Chapter 6) of the schemes it trusts. Information required to check the trust scheme membership and signatures are then automatically retrieved and verified. Further, TPL supports concepts to verify electronic transactions issued in trust schemes other than the local scheme. Besides, TPL can be used to (additionally) formulate access policies. Doing so makes complicated trust assessments easily customizable.

To establish trust in electronic transactions and attestations, TPL supports classical trust management approaches like Public Key Infrastructure (PKI)-based trust infrastructures and X.509 certificates. TPL also supports several Self-Sovereign Identity (SSI) concepts, such as resolving of Decentralized Identifiers (DIDs), retrieval of trust information from a Distributed Ledger (DL), and the verification of World Wide Web Consortium (W3C) Verifiable Credentials. We contribute towards integrating SSI and the interaction with a DL into a trust policy language. TPL further enables using SSI concepts and classical trust schemes in the same policy.

Chapter 7 Goals:

We discuss the overall goals of this thesis in Chapter 5. In this chapter, we focus on Goals 1, 2, and 3. In specific, we tackle the following aspects:

- from Goal 1: Complex Transactions, Different Types of Trust Schemes, Local Trust View

- from Goal 2: Global Trust Scheme Interoperability
- from Goal 3: Novel Models

Chapter 7 Outline:

We start the chapter with a detailed definition of requirements that our thesis goals pose on a trust policy system (Section 7.1). In Section 7.2, we then use these requirements to survey the state of the art.

In Section 7.3, we give a conceptual introduction of our TPL system. In this section we present the TPL language and its syntax and semantics. We also discuss the concept of TPL formats that enables part of TPL's modularity.

In Section 7.4, we present the integration of the TPL system into our global trust management system from Chapter 6. To extend TPL's modularity, we introduce a extensible library of predicates which can be used to query trust schemes from policies. We also discuss how the TPL system can be used to verify transactions from local (Section 7.4.2) and foreign (Section 7.4.3) trust schemes, as well as SSI credentials (Section 7.4.4).

In Section 7.5, we evaluate TPL against the requirements we previously introduced. We also describe how non-technical users can use graphical tools to author TPL policies. Further, we discuss the formal verification of TPL evaluations and consider the privacy of users.

To show the flexibility of TPL, in Section 7.6 we present a case study in which we add the TPL system to a distributed data marketplace. In doing so, we also present a generic access control extension that can be applied to other marketplace systems as well.

7.1. Requirements

In this section, we outline the requirements for our policy system. We start by deriving five requirements from the overall goals considered in this chapter (R1–R5). Additionally, we add 11 general policy language requirements (G1–G11) from Seamons et al. and Coi et al. [Sea+02; CO08]. In the following Section 7.2, we then use those requirements to survey the state of the art. Later in Section 7.5.1, we also use the requirements to evaluate our own approach.

7.1.1. Thesis Requirements

Based on the overall goals (see Chapter 5) relevant for this chapter, we derive the following five requirements for a trust policy system.[1]

R1 Local Trust View: The main reason for this chapter is the need to enable SPs to assess electronic transactions based on their own perception of trust (Goal 1). To fulfill this requirement, we require a expressive trust policy system for distributed trust management.

R2 Support for Trust Scheme Data and Trust Translation: To enable SPs to work with different qualities of trust (Goal 1) in a global setting (Goal 2), in Chapter 6 we introduce concepts like trust scheme memberships, trust translations and recognitions. To support local trust views utilizing this concepts, our policy language must support them. In specific, it needs to support a.) the use of trust scheme data discovered from various sources, b.) the use of trust recognitions to establish trust in foreign schemes, and c.) the possibility to specify rules on trust criteria and the use of trust translations to work with heterogeneous trust schemes.

From the implementation perspective, our proof of concept (Section 6.4) provides the Automated Trust Verifier (ATV) component, which enables the interaction with the mentioned concepts. Hence, the policy system should be integrated with the ATV component, developed in the Java programming language.

R3 Modularity and Extensibility: To ensure that there is no need to hardcode any policy-specific details (Goal 1), we seek that the core components of our system are build in a modular way, allowing to extend or replace them if needed. Core components with which our policy system is interacting are trust schemes and transaction formats. Hence, this must also modularize the trust scheme interface, allowing for novel models to be added later (Goal 3).

Extensibility is also a requirement proposed by Coi et al. to allow that the system can be updated and extended with new features [CO08].

[1] Note: In this thesis, we *don't* use the RFC 2119 [Bra97] convention of the words "must" and "should".

R4 Declarativity and Expressive Power: One goal is provide a policy system flexible enough to represent all kinds of expressive policies (Goal 1). To provide a policy designer with the greatest flexibility possible, we seek for a system build around a Turing-complete language. At the same time, policies should be easy to grasp for a policy designer, and thus semantically as easy as possible. We strive to achieve this by avoiding complex semantic structures and negative constraints, hence formulating all policies in a positive way only.

R5 Accountability: Policy systems deal with transactions of substantial value, so it is crucial that there are no undefined corner cases or bugs in the implementation. In support for Goal 1, it should thus be possible that an independent third party to easily review a policy decision.

7.1.2. General Policy Language Requirements

To fulfill general requirements on a trust policy system, we add the criteria from two review papers by Seamons et al. and Coi et al. [Sea+02; CO08].

G1 Well-defined semantics: The semantics (i.e., the meaning) of a policy must be independent of the implementation (e.g., of the language interpreter) [Sea+02]. This is an important requirement for the understanding of the language, but also for interoperability between different implementations.

G2 Monotonicity: A granted request for a specific resources must also be accepted if accompanied by additional credentials or attributes [Sea+02]. I.e., adding of new information can lead to additional permissions, but must not result in lower privileges. Thus, the policy cannot require the absence of information or attributes. The easiest way to achieve this is by banishing negation from the language [Sea+02].

G3 Evidences: A trust policy system is used to establish trust in data, and hence always deals with untrusted data. Hence, the policy system must be able to validate incoming credentials, i.e., support signed transactions and signature verification.

G4 Credential combinations: The policy system must accept transactions comprising of multiple credentials issued by different issuers. This requirement also supports our goal to support complex transactions (Goal 1).

G5 Condition expressiveness: The policy system must allow policy authors to constrain the type of credentials. Additionally, the system must support constraints on attribute values. In this thesis we also call this conditions *access (control) policies* (in contrast to *trust policies).* Further, the system must allows for different trust rules depending on the value of certain attributes.

G6 Inter-credential constraints: The policy system must support constrains in the attributes of two different credentials. E.g., by requiring that a specific attribute in two credentials has the same value, those two credentials could be linked. Additionally, the value of an attribute in a credential can influence the constraints on attributes in another credential.

G7 Credential chains: The system must provide enough expressive power and constructs to describe attestations that are signed by the holder of other attestations.

G8 Transitive closure: The system must allow for credential chains of arbitrary length. This is an extension of the credential chain requirement. The language should also allow to constrain the length of the chain.

G9 External functions: The system must provide function to operate on dates, numbers, etc. Additionally, it must functions to interact with other parts of the trust management system, i.e., the ATV. This is closely linked to the extensibility requirement, and also needed for our thesis goals, i.e., to interact with trust schemes and transactions. While Coi et al. describes functions that can modify the outside world, we require functions that can only change the state of the policy validation. This is needed to combine functions that retrieve some data with other functions that verify or use it. Apart from that, functions must be side-effect free w.r.t. the outside world (to facilitate the accountability requirement). This limitation does not concern topics like caching, logging and reporting, though.

G10 External trust data: The policy system should support the retrieval and processing of extra trust data in addition to the attestations provided by the user. For example, the system introduced in Chapter 6 queries the DNS to resolve the trust scheme membership of a credential. Additionally, certificate status (i.e., revocation) checks are other common sources for external trust data.

G11 Verifier requirements: Coi et al. also state several requirements for the policy compliance checkers, i.e., the verifier component operated by an SP [CO08, Section 3.2].

The verifier must validate the **credential validity** of any credential involved in the transaction, or provide means to the policy author to do so. Additionally, the verifier must provide means for **credential chain discovery** to retrieve any credentials in a chain between two involved credentials.

Requirements out of scope: Seamons et al. consider the verification of credential ownership, which is not part of our goals. Additionally, they discuss the protection of sensitive policies, which is also out of the scope of our design goals. Nevertheless, we utilize encrypted policies in our use-case implementation in Section 7.6. Coi et al. further propose support for delegation and trust negotiation, which we don't consider in our system.

7.2. State of the Art

Based on the requirements defined in Section 7.1, we now evaluate the state of the art of (trust) policy systems [CO08; Yai19]. Given the architecture introduced in Figure 6.2 and the overall architecture of systems considered by this thesis (see Chapter 2), we focus our analysis on policy languages for *distributed* trust management. This has the implication that we don't consider languages that assume that the issuer and verifier of an authorization token is the same entity. Hence, we require verifiable evidence (i.e., signed attestations). Additionally, we don't consider languages that only support a fixed set of trusted issuers, as this contradicts our trust scheme based approach.

Evaluation by Coi et al.: In their paper on trust, security, and privacy policy languages, Coi et al. evaluate 12 policy languages against 10 criteria [CO08]. Since those criteria are a subset of our requirements (see Section 7.1), we take their evaluation as basis for our state of the art analysis. The languages they evaluate are Cassandra [BS04], EPAL [Ash+03], KAoS [Usz+03], PeerTrust [NOW04; Gav+04], Ponder [Dam+01], Protune [BO05; BOP06], PSPL [BS00], Rei [KFJ03], RT [LM03], TPL (unrelated to our TPL system) [Her+00], WSPL [And04] and XACML [Lor+03]. Of those languages, Cassandra, PeerTrust and Protune fulfill all their criteria. But, there are design decisions that make those languages unsuitable for our approach. In specific, in terms of the requirement of "external functions", PeerTrust is limited to sending evidence during interactive trust negotiation, which is out of our scope. Protune supports "whatever kind of actions, not necessarily side-effect free, as long as a basic assumption holds, namely that action results do not interfere with each other" [CO08]. Cassandra's action execution is limited to side-effect free functions. Additionally, as Yaich states, "the Cassandra trust engine is only available as a proof-of-concept implementation in OCaml" [Yai19, p81]. Those limitations prevent their use for work with trust scheme discovery and the integration with our ATV.

Nevertheless, their design provides a valuable basis for our own investigation. Protune is based on logic programming (extended with an object-oriented syntax [Yai19]), while Cassandra and PeerTrust are based on Constrained DATALOG (a subset of logic programming) [CO08; Mai+18]. All those languages fulfill our requirements on expressiveness and declarative power. This makes the logic programming paradigm a suitable candidate for our goals (cf. [BRS12]).

Other languages: Since the abovementioned survey by Coi et al. is 15 years old, we also explore and evaluate newer trust management systems and their policy languages. We start by reviewing the retrospective study on trust management systems by Yaich from 2019 [Yai19].[2] As it does not involve any systems which we did not discuss above, we then survey the literature for additional trust policy languages. SecPal is a decentralized authorization language based on logic programming [BFG10]. Braghin notes that, as SecPal focuses on authorization, "[it] does not provide a direct way to address credential-based policies, nor it provides an approach

[2]Note: The study by Yaich was the newest survey available when the TPL system was created.

for specifying conditions against credentials" [Bra11, p. 8]. Belchior et al. propose a Self-Sovereign Identity based access control (SSIBAC) for service providers [Bel+20]. It leverages conventional attribute-based access control using the attributes in SSI credentials. SSIBAC uses the XACML standard for policy specification. XACML does not fulfill our requirements on well-defined semantics and does not support verifiable evidence [CO08]. PolicyMan[3] supports simple trust rules, but only allows for static trust rules (i.e., direct specification of an issuer or group of issuers).

Conclusion: After this preliminary evaluation we conclude that there is no candidate that fulfills all our requirements. This is not surprising, as our requirements are very specific and introduce concepts not commonly used by other languages.

Given that circumstance, we face three options: First, extend *and* adapt an existing policy language to our needs. As this would require modifications not only to the implementation but also to the language itself, we do not consider this option. This option is also less attractive since our integration requirements pose a strong constraint on the technology stack used to build the trust language (i.e., Java). Second, develop a new language from scratch. We do not consider that option, as designing a new language is a complex task and likely reinvents the wheel in several aspects. Third, a middle ground between the first two options is to adopt the design of an existing language and build a new system around it. In our evaluation we discovered interesting properties and paradigms that we learn from. For example, the logic programming paradigm supports all our goals specific to core language design. Based on this assessment, we went for the third option. The result is the TPL system, which we present and discuss in the rest of this chapter. In Section 7.5.1, we then evaluate our TPL system against the abovementioned requirements.

7.3. The *TPL* Policy System

We introduce TPL as a system for SPs to formulate their **trust- and access-policies**. Using TPL, SPs decide if and how to rely on existing trust schemes like the European Union's (EU) eIDAS, other trust schemes endorsed by trusted schemes, or DL-based trust schemes.

[3] `https://gitlab.grnet.gr/essif-lab/business/policyman/policyman_project_summary`, accessed on 2023-07-18

TPL is both a trust policy *language* and a trust management *system* geared to support and integrate today's existing trust schemes and enable a global trust infrastructure.[4] In this context, the term **electronic transaction** denotes all data that the SP receives and wants to verify. Examples are attestations, as discussed in earlier chapters. More concrete examples are a bid in an auction house or a login at an online system. Those transactions have in common that an SP needs to establish trust in them to rely on their content. TPL helps to specify and automatically implement the SP's business policy for trust decisions.

TPL is designed in the context of the global trust management architecture introduced in Chapter 6 above. The idea is that there are many trust schemes, but no scheme on which the whole world agrees. To mitigate this, TPL supports different formats of electronic documents and transactions. It also allows authorities behind a trust scheme to define trust recognitions of other schemes. These recognitions can be automatically processed during the execution of a policy. A policy author can then decide to what extent to accept the result of a trust recognition and corresponding trust translation.

7.3.1. Concepts

In this section we introduce several important concepts of the TPL system. The TPL system comprises a language specification (the *TPL language*) and a software component (the *TPL interpreter*). A SP uses the TPL language to encode trust and access rules, forming a *TPL policy*. This TPL policy is then executed by the TPL interpreter, evaluating an electronic transaction. In doing so, the interpreter uses concepts of the language that provide an abstraction layer to file formats (*TPL formats*) and online trust information sources (*built-in predicates*).

TPL Policy: A TPL policy is a list of Horn clauses in a syntax similar to Prolog [ISO95]. A (definite) Horn clause is a rule that specifies a relationship between terms. It consists of a head and a body, separated by an arrow from right to left (←, encoded as `:-`). The head of a definite clause is a predicate, and the body is a conjunction (logical *and*) of predicates called goals.

[4] Even though the name *TPL* originally came from an acronym for the TPL trust policy language, it is now simply a name for the TPL system. TPL supports both *trust*- and *access*-policies.

Each Horn clause is in the form $p(u)$:- $q_1(u), ..., q_n(u)$ meaning: if all the goals $q_i(u)$ are true, then also $p(t)$ is true. A set of clauses for the same p defines the specification for the *TPL predicate* p. As only one clause for p needs to succeed, a set of more than one clause for the same p represents a disjunction (logical *or*).

In TPL, the entry point of a policy is represented by the `accept` predicate. This notation stems from the question the SP is asking the TPL interpreter: the query `accept(Form)` asks whether the policy *accepts* a transaction. The arguments of the `accept` predicate depend on the concrete instantiation but always include the holder's transaction.[5] Listing 7.1 illustrates this syntax.

```
accept(Form) :- extract(Form, age, Age), Age >= 18.
```

Listing 7.1.: Simple policy illustrating the syntax of TPL.

TPL Interpreter: As TPL is an interpreted language, policies encoded in TPL syntax are interpreted by a TPL interpreter. The TPL interpreter is a software component used by a SP as part of its trust verification system.[6] The interpreter handles the syntax-checking, parsing, and execution of a TPL policy. Since the interpreter is focused on the language handling and interpretation of a policy, it does not directly provide functionalities for trust scheme discovery. Instead, it interacts with our ATV to query for trust information (see Chapter 6). We will discuss this integration further in Section 7.4 below.

Formats: In TPL, the concept of *formats* connects policies with parsers that extract values from complex data formats and ensure compliance with data schemata. These values can be concrete numbers, constants, or of some complex format themselves. For example, the `extract` predicate is used in a policy to extract values. For example, `extract(Transaction,`

[5] The argument is commonly called *Form* since it follows the metaphor of a paper form that the holder filled out. An example for a TPL policy with multiple *accept*-arguments is given in Listing 7.8 below.

[6] The source code of our TPL interpreter is available at `https://github.com/H2020LIGHTest/TrustPolicyInterpreter`

`certificate, Certificate)` extracts data from a transaction's field called *certificate* into the variable `Certificate`. Further, the special field `format` is used to access the data format of some data/transaction. For example, `extract(Certificate, format, CertFormat)` retrieves (or constrains) the certificate's file format.

Built-in predicates: TPL features built-in predicates which wrap context-specific discovery and verification logic. That includes all the interactions with trust schemes and format parsing logic. The name *built-in* predicate comes from the fact that this logic is not implemented in TPL syntax but these predicates are built into the system. However, implementing of the predicates happens in an extensible predicate library and can be used to customize the TPL system to an SP's needs. For example, our implementation builds on the infrastructure introduced in Chapter 6. There, the TPL interpreter calls a function of the ATV to retrieve trust information (see Section 7.4 below).

Built-in predicates store their results in the variables passed as output parameters to the call. By directly binding an output parameter to some constant (denoted by an identifier beginning with a lower-case letter), we require that parameter to be that value for the clause to be true. Thus, `extract(Cert, format, x509)` is a shortcut for `extract(Cert, format, CertFormat), CertFormat == x509`.

7.3.2. Syntax

The language of TPL consists of *definite horn clauses*. Its syntax is based on that of Prolog [ISO95; DEC96].

As the basis for our language's grammar, we define four sets of symbols:[7] (1) Variable symbols—starting with upper-case letters. (2) Constant symbols—starting with lower-case letters. (3) Function symbols—starting with lower-case letters, having a fixed number of arguments.[8] (4) Predicate symbols—starting with lower-case letters, having a fixed number of arguments. With this in place, we use a grammar in Extended Backus–Naur form (EBNF) to define the syntax of TPL, as shown in Figure 7.1. A visualization of TPL's grammar is given in Appendix B. In addition to

[7] In the given EBNF grammar, those sets of symbols are used as terminals and not specified.

[8] The number of arguments taken by a relation is also called its *arity*.

this basis, TPL has built-in arithmetic operators which are mapped to function symbols (see example in Listing 7.2).

```
TPLPolicy   ::=  Clause*

Query       ::=  (Predication,)* Predication.

Clause      ::=  Predication.
             |   Predication :- (Predication,)* Predication.

Predication ::=  PredicateSymbol
             |   PredicateSymbol((Term,)* Term)

Term        ::=  VariableSymbol
             |   ConstantSymbol
             |   FunctionSymbol((Term,)* Term).
```

Figure 7.1.: EBNF grammar specifyign the syntax of TPL

Example TPL Policy: An example of a simple policy in TPL syntax is shown in Listing 7.2. This example is an encoding of the following policy:

Example Policy Rule 1. *The auction house accepts any transaction which is of the "Auction house 2023" format and contains a bid up to 100 euros.*

In this example, the variable `Form` is the transaction, here a bidding form in some concrete data format (e.g., Extensible Markup Language (XML)). The built-in predicate `extract` is used to extract the attributes from the form. This predicate represents the interface to the parser for the respective data format. The detailed semantics and parameters of the built-in predicates are described in Section 7.4.1 below.

To ensure format compliance, the first extraction must always be the check for the expected format type. In our example, the format used by the concrete auction house is identified by the `theAuctionHouse2023Format` constant. This constant is defined by the respective format parser in the SP's format library. To load the format, we constrain the value of the

format field to that constant. Next, we extract the `bid` field, bound to the output parameter `Bid`. Finally, we check that the value is below `100`.[9]

```
accept(Form) :-
  extract(Form, format, theAuctionHouse2023Format),
  extract(Form, bid, Bid),
  Bid <= 100.
```

Listing 7.2.: Example simple TPL policy (without trust rules).

7.3.3. Semantics

We formally define the semantics of TPL in two ways: logical semantics and executable semantics.

Logical Semantics:

A logical view of the semantics can be obtained if we consider the Horn clauses as logical formulas of first-order logic. In doing so, `:-` is the arrow $\leftarrow$ (logical implication from right to left), and the comma represents logical conjunction (logical *and*). Further, all variables of every Horn clause are universally quantified. Following this, $p(X, Y)$:- $q(X), r(Y, X)$ becomes $\forall X, Y : p(X, Y) \leftarrow q(X) \wedge r(Y, X)$.

Special care must be taken for built-in predicates, i.e., the interface to the environment. For the semantics, we fix the meaning of these built-in predicates to a snapshot of the world. In particular, we assume that during the policy checking, the world's state does not change. It is also possible to evaluate a historical policy decision by specifying the environment as it was at some point in the past. This allows us to answer whether a policy accepted a given transaction at a previous point in time.

Executable Semantics:

Rules can be evaluated in the same way as in Prolog. To see if a query $p(s)$ succeeds, find a suitable rule; e.g., the rule $p(u)$ fits if s and u can be unified. Then apply the resulting unifier to all $q_i(u_i)$ and evaluate them. If the evaluation for the subqueries evaluates to *true*, then we say that the

[9] In that example, the currency of the value is fixed by the format.

original query was also evaluated to a success. If that is not the case, then try again with the next suitable rule if any such exists. The subqueries are evaluated in the same way by recursion, except for the case where a q_i is a built-in predicate. In that case, the predicate is directly executed, which either results in a success or a failure.

TPL is similar to Prolog but does not include the `!` (cut) operator or negation as failure. Such concepts prohibit interpretation as logical formulas and thus hinder the definition of clear and simple semantics. However, policies are lists of definite Horn clauses and TPL also shares most of Prolog's syntax. Therefore, TPL's executable semantics are the same as that of Prolog; except that in TPL, unification always includes the occurs check.[10]

The semantics of Prolog can be described as an interpreter.[11] But, this interpreter is only responsible for the parsing and executing of the policies, as TPL's built-in predicates (such as `extract` and `lookup`) are not part of TPL's core language but are defined outside of the language directly in the TPL system.

7.3.4. Formats

Policies work on data represented by various concrete data formats, from X.509 certificates and DNS resource records (cf. Chapter 6) to custom data formats for electronic transactions. TPL supports all of these in a flexible way without cluttering the policies with low-level details like parsing. To achieve this, we use an abstract notion of **formats**, similar to abstract syntax. We use the metaphor of a *paper form* with fields to fill in and each field having a unique identifier. This represents an abstraction layer between TPL and concrete measures to structure this information (e.g., XML). Any concrete format can be connected to TPL by providing a parser. A parser performs the transformation between actual byte strings and the abstract syntax. In TPL, these format parsers are extensible and stored in a *format parser library*.

Abstractly, a form is a set of attribute-value pairs, as shown in Listing 7.3.

The actual transaction on the string level could be an XML representation,

[10] An *occurs check* assesses if a variable is part of another variable before attempting to perform unification on them. Occurs checks are disabled in Prolog by default, which can lead to circular/infinite loops.

[11] see, e.g., Deransart, Ed-Dbali and Cervoni's textbook [DEC96, Section 4.2].

```
{(format, the_auction_house_2023), (bidder_name, "Jon Doe"),
 (street, "Dartmouth St"), (city, "Midfarthington"),
 (country, "England"),
 (lot_number, 54678), (bid, 60),
 (signature, ...), (certificate, ...)}
```

Listing 7.3.: Example of an abstract representation of a simple electronic transaction.

as shown in Listing 7.4.

Abstract symbols like `bidder_name` are a sound abstraction of their concrete byte-level format [MK14]. Notice that the XML representation's tree structure and the attribute value pair set representation are different. This forms a layer on top of an XML format, so one does not have to browse the XML parse tree but has a purpose-specific immediate representation of the data. TPL provides the built-in predicate `extract` to connect the interpreter with the appropriate parser. Using these parsers, attributes can be extracted from the format specified by the attribute value pair representation.

```
<?xml version="1.0" encoding="UTF-8"?>
<form format="the_auction_house_2023" xmlns=...>
  <person>
    <name>Jon Doe</name>
    <street>Dartmouth St.</street>
    <city>Midfarthington</city>
    <country>England</country>
  </person>
  <lot_number>54678</lot_number>
  <bid>60</bid>
  <ds:Signature>
    ...
    <ds:X509Certificate> ... </ds:X509Certificate>
  </ds:Signature>
</form>
```

Listing 7.4.: Example of an auction house bid in the form of an eletronic transaction encoded as XML. XMLDsig omitted for clarity.

7.4. Integration

TPL policies are executed by the TPL interpreter. The interpreter is only responsible for parsing and interpreting the policy and has no functionality to access external data. To facilitate communication with the outside world (SP, trust scheme), the TPL interpreter is integrated into our ATV, which we introduced in Figure 6.7. This integration is visualized in Figure 7.2.

Interfaces: The ATV provides the interface to service providers who need to assess the trustworthiness of incoming transactions (e.g., an attestation is loaded using the ATV GUI or API). Thus, the ATV receives a trust policy and an attestation from the SP, and initializes the TPL interpreter.

TPL Interpreter: The interpreter is responsible for executing the policy. To do so, it first parses the policy and checks its syntax. In case of syntax or execution errors, the interpreter halts and directly provides error messages to the SP. For a policy where the execution takes too long, the interpreter rejects the transaction based on a timeout.

Libraries for Built-in predicates and Formats: The ATV also

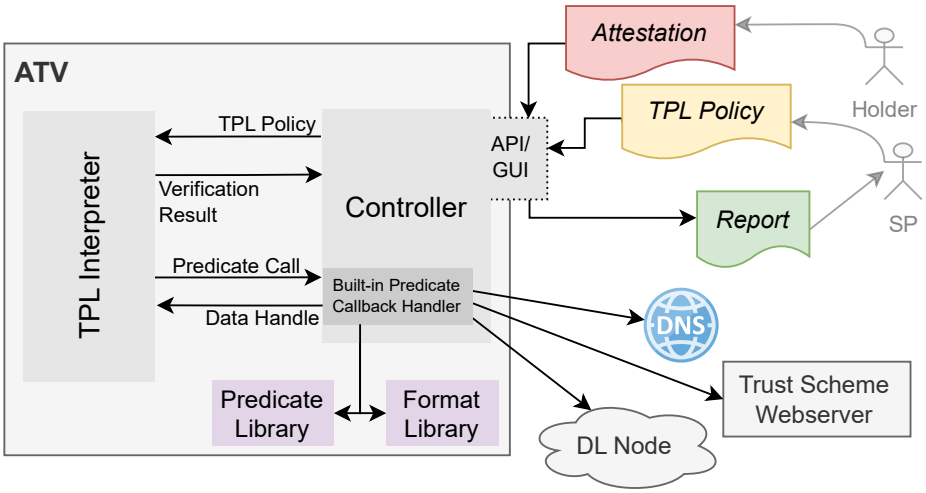

Figure 7.2.: Architecture of the integration of the TPL system into our ATV component. The components enabling TPL's extensibility are highlighted in purple.

provides the functionality for format parsers and built-in predicates to the interpreter. Built-in predicates are the predicates like `extract` whose truth value depends on external facts and actions. These predicates are implemented as functions in the ATV that the TPL interpreter invokes once its execution reaches the call of a built-in predicate. To do so, the ATV provides a callback handler for built-in predicates. Depending on the concrete built-in predicate, the handler discovers or retrieves trust information from the internet (see Section 6.2.4). Or, it uses the library of format parsers to parse some data, e.g., the attestation, the issuer's certificate, or some other data retrieved from the internet. Neither the list of built-in predicates nor the format library are hardcoded in the ATV. Thus, any SP can add their own predicates or formats, adapting the ATV to their needs. Those libraries enable the extensibility of the TPL system.

Data Handles: After finishing the execution of the built-in predicate, ATV handler gives the control back to the interpreter. In our implementation, the callback handler does not directly return the retrieved data to the interpreter. Instead, the handler takes over that data's management, storage, and caching. To allow the interpreter to execute predicates on the data, the handler returns a data handle to the interpreter. This data handle is an internal URI the interpreter uses to address data in a later

call to some built-in predicate. The structure of this handle URI is opaque to the interpreter.

ATV Report: In the end, the ATV returns the final result of the policy execution (as determined by the interpreter). In addition, the ATV logs every call to a built-in predicate in a report. This report also contains the results of all predicates calls. After the execution is completed, this report is returned to the SP. The SP can use the result to learn whether or not the incoming attestation is trustworthy with regard to the specific trust policy. And the report informs the SP about the reasons for this decision. This can for example be used by the SP to automatically request additional information from the user. An example report is displayed in the ATV GUI screenshot in Figure 6.7.

7.4.1. Main Built-in Predicates

This section describes the important built-in predicates of TPL in more detail, as they are specified in our TPL papers [Möd+19; Alb+21].

Built-in Predicate 1 (extract)**.** The `extract` predicate is used to extract information from a document (e.g., a transaction, certificate, or trust list entry). This predicate gives a uniform interface to all kinds of data formats. The interpreter is designed modular so that new data formats can easily be integrated by providing a parser for the respective data structure. For a call

```
extract(From, Field, Out)
```

we have that `Form` is an input document, `Field` is a field of the document, and `Out` is the output, i.e., the value of that field.

The set of document fields that are available depends on the format. When trying to `extract` a field that does not exist in the current format, the predicate fails. For every format, the field `format` is always defined and returns the unique identifier for the document's format.

Built-in Predicate 2 (lookup and trustlist)**.** The `lookup` predicate allows to perform lookups at DNS name servers and HTTP queries authenticated using DNSSEC. The input parameter `Domain` defines the DNS domain to query, while the output parameter `Entry` contains the resulting document. It implements the functionality required to discover trust schemes or trust translation, and to retrieve the information published by them. In a similar

manner, the trustlist predicate is a more specific case, which is used to retrieve a single trust list entry, identified by the parameter `Certificate`.

```
lookup(Domain, Entry)
trustlist(Domain, Certificate, TrustListEntry)
```

Due to the nature of Horn clauses, the same predicate is often called multiple times. Doing so for built-in predicates results in multiple network requests leading to a negative performance impact. To mitigate this, those predicates also implement an internal cache.

Built-in Predicate 3 (trustscheme)**.** The `trustscheme` predicate checks if a trust scheme membership claim (a DNS identifier) represents a trusted scheme. Both parameters are input parameters. A call

```
trustscheme(TrustSchemeClaim, eIDAS_qualified)
```

is true if and only if the trust scheme claim is a claim for an eIDAS membership.

Built-in Predicate 4 (verify_signature)**.** The `verify_signature` predicate has two input parameters. For a call

```
verify_signature(Form, PubK)
```

the TPL interpreter will use the appropriate signature verification function for the format of `Form` and succeeds if and only if the form was properly signed using the given key.

Built-in Predicate 5 (verify_hash)**.** The `verify_hash` predicate checks if an object evaluates to the correct hash value. So for a call

```
verify_hash(Form, Hash)
```

the TPL interpreter will use the appropriate hash function for the format of `Form` and succeeds if and only if the parameter `Form` has the same hash as passed by the parameter `Hash`.

In addition, our implementation comes with the `encode()` built-in predicates to support the encoding and concatenation of scheme identifiers.

For our extension to SSI, we introduce one additional built-in predicate:

Built-in Predicate 6 (resolveDID)**.** The `resolveDID` predicate takes three arguments: The first argument is the DID to resolve. The second argument specifies the minimal age of the block in which the DID was registered. On

calling this predicate, the ATV will try to look up the DID document at the DL and, on success, return the result to the third argument's variable.

```
resolveDID(DIDsubject, min_blockage, DIDdocument)
```

If a DL-based system is used, the block age parameter can be seen as a choice of assurance level regarding the DID document. Since a young block could be dropped as the blockchain grows, an older block is more established and is less likely to drop out.

7.4.2. TPL for Trusting the Local Scheme

In this section, we extend our example policy with trust rules and built-in predicates. So far, our example auction house only accepts bids up to a certain amount but puts no constraints on who may place a bid. For larger bids up to a specific limit, the auction house needs to know who the bidders are. Bidder authentication is achieved by issuing qualified certificates to users. Such a policy represents an example of a policy containing both trust- and access-rules. Therefore, we extend our example:

Example Policy Rule 2. *The auction house accepts any bid up to 1500 euros, if it is signed by an ExampleScheme qualified signature.*

Thus, we need to perform the following checks: (1) Is the bid amount smaller than 1500 euros? (2) Has the bidder's certificate been issued by an ExampleScheme qualified authority? (3) Did the bidder actually sign the bid?

Signatures and Signable Formats: To verify signatures, we use the built-in predicate `verify_signature` and signable formats: A signable format is a format for which a signature verification function is specified. For a form of the specified format and a public key, we can verify if the form is properly signed.

Trust Scheme lookups: We need to verify the trust scheme membership of the bidder's issuer and thus have to obtain the associated trust list. Trust lists are discovered using a trust scheme membership claim which is inside the bidder's or issuer's certificate (see Section 6.2.3). In our approach, this claim is represented by a domain name. For example, the (fictional) DNS identifier `example.com` is a (direct) claim of membership in the trust scheme of qualified ExampleScheme authorities.

The `trustlist` built-in predicate triggers a server lookup. It will succeed if a the given trust scheme exists, the trust list is available, and the desired certificate is on that list. It fails otherwise. It, therefore, acts as a requirement in a policy that the given certificate is on the claimed trust list. To claim a trust scheme membership, a certificate includes a field `trustScheme` that states the DNS identifier of the trust scheme it claims to be in. In order to ensure that the domain actually belongs to our desired trust scheme, we use the built-in predicate `trustscheme` with the `exampleScheme_qualified` constant.

Specifying the policy: We encode Example Policy Rule 2 in a TPL policy, shown in Listing 7.5.

When this TPL code is added to a TPL specification containing Listing 7.2, then any form that lives up to the requirements of either `accept()` rule is accepted. The result of this policy is that the auction house accepts bids up to 100€ without identification, and bids up to 1500€ for bidders authenticated by a qualified issuer.

Listing 7.5 requires that the `format` of the form is the auction house format and extracts the bid to check that it is at most 1500 euros. After that, it extracts the bidder's certificate. Since `Certificate` is also a form, the policy extracts the public key of the bidder, given in the `pubKey` field. Then the verification of the signature of the form is done for the public key using the `verify_signature` predicate. Afterward, the policy checks that the certificate is ExampleScheme qualified. This trust check is done in a separate predicate specified directly in the policy. From the bidder's certificate, it extracts the issuer's certificate, given in `IssuerCert`. From the `IssuerCert` it then extracts the `TrustSchemeClaim`, a DNS identifier used to address the trust scheme and verify the issuer's trust scheme claim. The policy checks that the trust scheme membership claim is really the ExampleScheme trusted by the SP. This is done using the `trustscheme` predicate. A lookup is then done using the `trustlist` predicate, which discovers and retrieves the trust list and verifies that the `IssuerCert` is on the list. If so, it returns the corresponding list `Entry`. Lastly, the issuer's public key is extracted from the list entry and then used with `verify_signature` to verify the signature on the bidder's certificate. The `Entry` must contain at least the issuer's public key, such that it can be verified to be the same as the issuer key recorded in the certificate.

This example shows that policies can be specified on an abstract level.

```
accept(Form) :-
  extract(Form, format, theAuctionHouse2023format),

  % access-policy rule
  extract(Form, bid, Bid),
  Bid <= 1500,

  % verify holder's signature on transaction
  extract(Form, certificate, Certificate),
  extract(Certificate, format, qualified_cert),
  extract(Certificate, pubKey, Pk),
  verify_signature(Form, Pk),

  % establish trust in certificate
  check_qualified(Certificate).

check_qualified(Certificate) :-
  % extract issuer's cert & trust scheme membership claim
  extract(Certificate, issuer, IssuerCert),
  extract(IssuerCert, format, qualified_cert),
  extract(IssuerCert, trustScheme, TrustSchemeClaim),

  % check if claimed scheme is trusted
  trustscheme(TrustSchemeClaim, exampleScheme_qualified),

  % verify if issuer is qualified
  trustlist(TrustSchemeClaim, IssuerCert, Entry),
  extract(Entry, pubKey, PkIss),

  % verify if issuer signed holer's certificate
  verify_signature(Certificate, PkIss).
```

Listing 7.5.: Example TPL policy with trust rules.

TPL frees the policy author from specifying the whole interaction with the internet. It also takes care of authenticating the retrieved data. Since the verification of a certificate often follows the same steps, this sequence of predicates could be collected in a "TPL standard library" and then re-used in different policies (see Section 7.5.4).

7.4.3. TPL for Trusting a Foreign Scheme

In the previous section, we discussed how TPL can be used to formulate complex trust policies. So far, we only consider the verification of attestations that have been issued in the trust scheme that the verifier directly trusts (local scheme). But a verifier can also use TPL to verify trust rules that were issued in a different scheme. As discussed in Section 6.3, trust schemes can define trust recognitions, i.e., an SP might consider other schemes equivalent to their trusted one. These recognitions can then be used in a TPL policy. We extend our example policy accordingly:

Example Policy Rule 3. *The auction house accepts any bid of at most 1000 euros with a signature from a scheme outside ExampleScheme if the scheme is deemed equivalent to ExampleScheme via a regognition published by ExampleScheme.*

We introduce a notation of equivalence with respect to a recognition relying on the trust recognition provided by the authority of the local scheme (see Section 6.3). The used example policy is similar to Listing 7.5, but the `trustscheme` predicate is changed to `trust`, which allows trust recognition and is defined explicitly in the TPL policy: `trust` checks that a trust scheme membership claim belongs either directly to the scheme we are trusting or belongs to an equivalent scheme, as shown in Listing 7.6. A complete example for a TPL policy with trust translation (for non-equivalent schemes) is shown in Listing 6.5 above.

For a claim of a foreign scheme and the name of a trusted scheme, the built-in predicate `encode` generates an identifier for the trust recognition. Suppose `Claim` is a (hypothetical) Swiss scheme located at `example.admin.ch` and the `TrustedScheme` is `example.com`. Then the domain points to the DNS identifier `admin.ch._translation._trust.example.com` (i.e., it should escape the domain of the original scheme and select the corresponding recognition of ExampleScheme). This domain should refer to the entry about the Swiss scheme at ExampleScheme. The entry is then used to discover information that can be used to verify equivalence. In the

```
trust(Claim, TrustedScheme) :-
  trustscheme(Claim, TrustedScheme).

trust(Claim, TrustedScheme) :-
  encode(Claim, TrustedScheme, Domain),
  lookup(Domain, TranslationData),
  extract(TranslationData, recognition_level, equivalent).
```

Listing 7.6.: Example trust policy with enabled trust recognition for equivalent schemes.

example case, we check if the recognition level field is set to `equivalent` (this also means there is no trust translation data, see Section 6.3).

If the two schemes are not equivalent, the ATV additionally retrieves the translation data and hands it to the TPL interpreter, which executes the translation. A TPL policy retrieving and using a trust translation is presented in Listing 6.5.

7.4.4. TPL for Trusting SSI Credentials

In addition to classical PKI and DNS-based trust schemes, TPL also supports several SSI concepts: the resolution of DIDs, retrieval of trust status information from a DL, and verification of W3C Verifiable Credentials. To illustrate this use case, we use the example of a social online platform for teenagers. In this fictional platform, teenagers can join without revealing their legal identity, but they need to prove their age to ensure only teenagers participate in the discussions.

Example Policy Rule 4. *The social platform accepts any registration of a person between the age of 12 and 18, if this is attested by a ExampleScheme qualified identity provider (IdP).*

Since other identity attributes of the teenagers are irrelevant, it is sufficient to provide a birthdate credential. This credential only contains the date of birth of the teenager and no further attributes.

In this example, the birthdate credential is a W3C W3C Verifiable Credential (VC) containing the user's DID as the subject and the date of birth encoded in the credential's `credentialSubject` field. To add (legal) value to the credential, a qualified issuer must issue it. A government

authority or a similar trusted institution can be a legitimate issuer for such claims.

Thus, we need to perform the following checks: (1) Is the holder's age ≥ 13 and ≤ 18? (2) Has the holder's credential been issued by an ExampleScheme qualified issuer? (3) Did the holder sign the registration?

To verify the registration transaction, the verifier checks the holder's signature on the registration. In the SSI world, this is done by resolving the DID of the holder, for example, from a DL [SSP23, Section 14]. The result of this is the DID document of the holder's DID. The verifier then uses the key extracted from the DID document to verify whether the holder signed the credential. For check 2, the issuer's signature on the (age) credential is verified in the same way. After checking whether the issuer's DID was used to sign the credential, the verifier needs to establish trust in the key. This is done by verifying the trust scheme membership claim of the issuer in the same way as above (see also Section 6.2).

The example TPL policy in Listing 7.7 corresponds to Example Policy Rule 4. A user sends a transaction containing a registration request in `registrationFormat` format to the discussion platform. The discussion platform uses the given policy and an automated verification tool to assess if the user is in the right age to be admissible to the platform.

The input parameter passed to the TPL interpreter (`Form`) contains the registration request, signature, and credential of the user. After ensuring the incoming transaction has the correct format, it checks the user's age by extracting the date of birth from the birthdate credential. Then, it uses the predicate `calculateAge` to derive the user's age before it verifies if the age is within the specified range.[12]

Next, the built-in predicate `extract` is used to retrieve the DID of the sender (credential subject) and the credential's issuer. These two DIDs are then used with the built-in predicate `resolveDID` (see Section 7.4.1) to retrieve the DID documents of the two entities. This step also considers the minimum age of the DID document, as specified by the second parameter. Each DID document contains a public key corresponding to DID, which is first used to verify the DID document itself. Further, the sender's key is used to verify the transaction, and the issuer's is used key to verify the credential. If all those checks succeed, there is a valid trust chain

[12] We omit a concrete explanation of the `calculateAge` predicate since it is not of interest for this discussion.

```
accept(Form) :-
  extract(Form, format, registrationFormat),
  extract(Form, birth_credential, Credential),
  extract(Credential, format, w3c_verifiableCredential),

  % access-policy rules
  extract(Credential, date_of_birth, Birthdate),
  calculateAge(Birthdate, Age),
  Age >= 13, Age <= 18,

  extract(Credential, dIDsubject, DIDsubject),
  extract(Credential, dIDissuer, DIDissuer),

  % verify holder's signature on transaction
  get_DIDdoc(DIDsubject, PKu, DIDDocSubject),
  verify_signature(Form, PKu),

  % verify issuer's signature on credential
  get_DIDdoc(DIDissuer, PKi, DIDDocIssuer),
  verify_signature(Credential, PKi),

  % establish trust in credential issuer
  check_issuer(DIDDocIssuer).

get_DIDdoc(DID, PK, DIDDoc) :-
  resolveDID(DID, 30, DIDDoc),
  extract(DIDDoc, format, w3c_diddoc),
  extract(DIDDoc, pk, PK), verify_signature(DIDDoc, PK).

check_issuer(DIDDocIssuer) :-
  extract(DIDDocIssuer, trustScheme, TrustSchemeClaim),
  trustscheme(TrustSchemeClaim, exampleScheme_qualified),
  trustlist(TrustSchemeClaim, DIDissuer, Entry),
  extract(Entry, pubKey, PKi),
  verify_signature(DIDDocIssuer, PKi).
```

Listing 7.7.: Example TPL policy using SSI concepts.

between the issuer and the registration request. The interpreter proceeds to authenticate the issuer itself. In our example, we authenticate the issuer in `check_issuer` using the trust scheme flow described in Section 7.4.2, and our ExampleScheme, showing that both centralized and decentralized world can be used together in one TPL policy.

7.5. Evaluation & Discussion

In this section, we evaluate our approach against the overall goals of this thesis. We then discuss several aspects of our TPL system.

7.5.1. Evaluation

In this chapter, we focused on Goals 1, 2, and 3 of our overall goals, i.e., the support of different qualities of trust, global interoperability, and extensibility (see also Chapter 5). Based on this, in Section 7.1, we formulated the concrete requirements for our trust policy system. From the stated goals, we derived five requirements (R1–R5, see Section 7.1.1). Additionally, we added 11 general trust policy language requirements (G1–G11, see Section 7.1.2), which we take from two survey papers [Sea+02; CO08]. We now discuss the evaluation of our TPL system against the tackled requirements. For each requirement we briefly state the measure or design decision that ensures we comply with it.

R1 Local Trust View: We fulfill this (meta-)requirement by providing a trust policy system tailored to the needs of global and heterogeneous trust management (Section 7.3).

R2 Support for Trust Scheme Data and Trust Translation: We comply with this requirement by providing predicates for working with the necessary trust concepts (Section 7.4.1 – Section 7.4.4). To facilitate this in our implementation, we integrate the TPL system with our ATV component (Section 7.4).

R3 Modularity and Extensibility: The built-in predicate concept and the modular predicate library provide extensibility of TPL's functionality. The extensible format library further enables modularity of supported transaction format.

R4 Declarativity and Expressive Power: TPL's declarativity and expressive power ensures that even complex policies can be represented in TPL. For example, depending on a transaction's payload or assessed risk, a policy could require different credentials or trust levels.

To achieve this, we designed TPL with inspiration from Prolog without the cut operator[13] and negation. Thus, policies are always formulated *positively*, i.e., under which conditions the policy is fulfilled. This is in contrast to policy languages that allow negative constraints, inducing much more complex semantic structures that are often hard to grasp for the policy designer. Nonetheless, TPL is Turing complete, i.e., every computable policy can be expressed. This programming aspect allows generating templates for the most common kinds of policies. Others have also produced graphical interfaces to TPL for users with different degrees of experience with encoding policy specifications (see below).

R5 Accountability: TPL's formal precision and accountability eliminate ambiguity and allows us to achieve verified evaluations. To achieve this, TPL has a strict syntax with defined semantics. Further, the execution of a TPL policy generates a trace that can be independently verified using a theorem prover (see Section 7.5.2 below).

G1 Well-defined semantics: TPL has a well-defined semantic (Section 7.3.3) based on logic programming.

G2 Monotonicity: Compliance with this requirement can be achieved by banishing a negation operator, i.e., by not allowing policies that require the absence of some credential [Sea+02, Section 3.1]. TPL intentionally does not support negation (see Section 7.3.3).

G3 Evidences: TPL supports constructs to work with *electronic transactions*—containers comprised of *signed attestations*. Those attestations can be handled by (signable) format (see Section 7.4.2), validating its integrity and authenticity.

[13] Prolog's cut operator is used to influence execution and optimize backtracking. Its implications introduce too much complexity for our policy system.

G4 Credential combinations: The electronic transactions supported by TPL are collections of one or many attestations, potentially signed by different entities. For example, the policy `accept(Form) :- extract(Form, diploma, GraduationCred), extract(Form, cv, CVCred)` extracts two credentials from a transaction container.

G5 Condition expressiveness: TPL allows (and requires) policy authors to constrain the type of credentials they validate. For example, the predicate `extract(GraduationCred, format, w3c_university_diploma)` uses TPL's format parser system to load the diploma document and verify its type. This predicate fails if the loaded document does not comply to the defined document type.

G6 Inter-credential constraints: TPL supports expressive inter-credential constraints. E.g., the policy `extract(GraduationCred, student_name, Name), extract(CVCred, person_name, Name)` requires that the attribute `student_name` in the first credential is equal to the `person_name` attribute of the second credential.

G7 Credential chains: TPL enables policy authors to verify signatures on any attestation handled by a (signable) format (see Section 7.4.2). Further, it supports the extraction of cryptographic keys from certificates. For example, a key extracted from a certificate can be used to verify the signature on another certificate, as shown in Listing 7.5.

G8 Transitive closure: By supporting recursion, built-in arithmetic operators (see Section 7.3.2), and credential chains (see G7), TPL enables credential chains of arbitrary (but contrained) length.

G9 External functions: TPL's built-in predicate concept and the modular predicate library enable the use of functionality that is *built-in* the system, i.e., not implemented in the TPL policy itself. In the TPL system, those libraries are extensible (see Section 7.4), i.e., the SP can add additional predicates. Examples for a simple external function are the `calculateAge` predicate (see Listing 7.7) and the `verify_hash` predicate (see Section 7.4.1).

The TPL system also supports predicates that modify the state of the policy evaluation, e.g., to pass data from one predicate to another (via data handles). However, TPL cannot prevent built-in predicates from modifying the outside world, with limits the fulfillment of the accountability requirement (R5) to a concrete execution trace.

G10 External trust data: By being integrated with our ATV component (see Section 7.4), TPL can query a trust scheme to retrieve additional trust data. Examples for TPL's support for external trust data are the `trustlist`, `lookup` and `resolveDID` predicates (see Section 7.4.1).

G11 Verifier requirements: In the TPL system, the combination of TPL interpreter and ATV acts as verifier component, i.e., the policy compliance checker. By means of (signable) formats and predicates for signature verification, TPL validates the credentials provided by the user. To accomplish this, TPL uses the rules codified in the trust policy to assess whether the issuer of those credentials is trusted. TPL supports different (extensible) ways to define the trust anchor for such validations, e.g., trust scheme lookups, DL lookups, trust translations, or a hardcoded list of trusted issuers. To build certificate chains from incoming attestations to retrieved trust roots, TPL optionally performs automated credential chain discovery [LWM01]. However, TPL does not support any online credential chain discovery, i.e., it cannot search for a full credential chain on its own.

7.5.2. Formal Verification

TPL has simple, clear, and precise semantics as first-order clauses interpreted with respect to an environment representing TPL's interaction with the outside world. To do so, the TPL interpreter works together with the Automated Trust Verifier (see Figure 6.7)—connecting transaction parsers and server lookups with logical evaluation.

A concern is the reliability of trust decisions, i.e., that bugs in a component might lead to false positives. Thus, TPL's architecture allows for boiling this problem down to the correctness of isolated components. To facilitate the verification of the correctness, our TPL interpreter produces an execution trace and stores it together with the policy decision. This trace is a logical representation of all loaded and retrieved documents and which signatures have been verified for which key. For the logical decision

of whether a decision follows from a policy, our colleagues Schlichtkrull et al. then use this trace to offer a reliable logical *verification*: they take the decision and the policy, together with the trace, and feed it into the automatic theorem prover RP_X [SM20]. The prover then checks if the given decision logically follows from the policy and the given documents. The correctness is double-checked, as the correctness of RP_X was formally proven using the theorem prover Isabelle/HOL [NPW02; Sch+18; SBT19].

7.5.3. Graphical Policy Authoring

The Prolog-inspired syntax of TPL allows the formulation of flexible and expressive policies. It does not require modifying the verifier's source code and hides technical details. But, the syntax of TPL can still challenge non-technical domain experts for business logic and trust rules (following legal regulations), who often formulate policies. To mitigate this usability challenge, two projects provide graphical editors for TPL policies, making policy authoring even easier.

In the LIGHTest project, Alber el al. provide the so-called Trust Policy Authoring Tool (TPAT) [Alb+19a; AW19]. Following the three-layered approach introduced by Weinhardt et al., the TPAT provides means to author TPL policies on three levels: the Prolog-like syntax, a drag-an-drop interface for graphical policy editing, and a (constrained) natural-language editor [WO19]. While each level improves the usability of policy editing, the flexibility decreases. A user-experience evaluation of this tool is further presented in [WP19].

Based on our work, Mödersheim and Ni introduce Graphical TPL (GTPL), the most high-level representation of TPL [MN19]. In GTPL, a policy can be formulated by filling out a template form (generated from a TPL format). The GTPL tool then uses the graphical policy to generate a machine-readable (and executable) TPL policy.

7.5.4. Future Work: TPL Standard Library

Since trust verification of some transactions often follows the same structure, policies often contain the same predicates. For example, to establish trust in a transaction, the SP must always check the trust scheme membership claim, retrieve corresponding trust information, and handle trust translations. To avoid this redundancy and additional work, we suggest a TPL standard library. This standard library could contain a set of

predicates useful for common trust management tasks. In contrast to TPL's built-in predicates, this standard library can be written entirely in TPL syntax. TPL authoring tools could also use the standard library predicates to hide complexity from policy authors.

7.5.5. Privacy

When presenting a credential during an authentication process, a holder often reveals sensitive data. Commonly, only parts of that data are needed by the SP to fulfill its purpose [Eur16, Article 5 (1)(b); Kyi+23]. For example, in our SSI example, even the date of birth reveals more information than needed. The only relevant information is the 1-bit of information on whether a person is in the defined age range. Thus, in part 2 of this thesis, we extend TPL by adding privacy features (see Chapter 9). By integrating privacy-enhancing technologies, we enable selective disclosure and predicates on attributes in trust policies.

7.6. Use Case: TPL for Decentralized Systems

> This section is based on the paper *YOU SHALL NOT COMPUTE on my Data: Access Policies for Privacy-Preserving Data Marketplaces and an Implementation for a Distributed Market using MPC* by More and Alber [MA22]. Parts of this paper have been copied verbatim.

In this section, we present a use case of our TPL system. The focus is on using policies in decentralized platforms such as an online marketplace for (personal) data, like the one we build in the Horizon 2020 project KRAKEN [Koc+22].

Context: Private Data Marketplaces

Personal data has become an attractive source to derive insights for the individual as well as for various companies and institutions. Those data sets can be analyzed using computations like traditional algorithms and novel machine learning-based approaches. The results of such computations have proven valuable for different business and research fields such as medicine, marketing, and more. To enhance the analysis of such data sets, available data must be efficiently brokered to relevant consumers.

Data marketplaces take on this brokerage task. However, the collected personal data is highly sensitive, prompting legislators to protect it well (see Chapter 3). An example is the EU's General Data Protection Regulation (GDPR) [Eur16], which defines the circumstances under which collecting, transmitting, storing, or processing such data is allowed. Especially data sets that might identify a specific person present a unique challenge since misuse can lead to discrimination (e.g., insurance, job market).

Private data marketplaces [Kou+20] try to mitigate these issues by using modern privacy-enhancing technologies. These technologies enable the computation on personal data without revealing the data itself. Recently, multiple approaches relying on this principle have been published [Env22; Kou+20]. One of them is KRAKEN [Koc+20; Koc+22], a marketplace architecture that uses Multi-Party Computation (MPC) to preserve users' privacy. The data is distributed in opaque shares to several nodes for computation. Only the final assembly of all the output shares discloses the result to the computation buyer.

Challenge: Control computations on personal data

A challenge private data marketplaces face is that users have limited ability to control who can buy their data and what buyers can do with this data. Further, users must trust the marketplace to follow the rules they specify for their data. Therefore, data sellers must trust that the marketplace is not covertly performing computations on their data. E.g., on the KRAKEN marketplace, data providers cannot easily control who can buy computations on their data since the marketplace's computation system has no information about the buyer's identity. For the same reason, the marketplace also has access to the computation results, even if a legitimate user launched the computation.

Concept Overview

We tackle the described challenge by adding a policy system to private data marketplaces.

We introduce an architecture for an extension of private data marketplaces. This extension features a flexible access control mechanism based on our TPL system. In the resulting marketplace system, data sellers can define expressive policies to control the usage of their data. An advantage of TPL is that data sellers can build their policies on qualified and thus trustworthy information. They can also rely on trust information from various other sources in their policies, e.g., SSI and distributed ledgers.

Using TPL also enables each seller to define their own policy for their data instead of relying on a set of rules pre-defined by the system.

Those policies are then attached and cryptographically linked to data products offered on a data marketplace. When a buyer purchases a computation on some data products, they are asked to provide credentials certifying their identity and other attributes. The marketplace's computation system then verifies if the credentials fulfill the policy for the selected data. Additionally, the system uses the policy to check if the (now-authenticated) buyer is qualified to execute the concrete computation the buyer requested. The allowed computation and parameters might depend on the identity of the buyer. Only then the system proceeds and executes the computation.

Further, the buyer's credentials are used to encrypt the result of a computation. This ensures that only the legitimate buyer can access the result.

Scope: Our design focuses on *private* data marketplaces that allow a computation on user's data *without the user's involvement* (non-interactive). Thus, we don't consider systems where the user participates in the computation on their data (which does not require this type of policy system).

7.6.1. Architecture

In general, a private data marketplace consists of the following components:

Data Seller: The actor who produces data and wants to offer it on the marketplace. To host this data, the data seller uses some **cloud storage**. Since the data is encrypted at this stage, the seller can also use a public cloud storage. Some models subdivide the data seller further into separate roles, i.e., the data producer/generator, the data subject, and the data provider.

Data Buyer: The actor that wants to buy computations on the data of several data sellers. They select one or multiple data products on a marketplace and decide which computations to execute. The data buyer is sometimes refered to as data consumer.

Marketplace: The online platform which acts as a broker to connect data sellers with relevant data buyers and enables the data trade. The marketplace provides a **catalog of data products** to which a data seller can add their **data records**. In addition, the marketplace helps the data

buyer to find data products of their liking and sells the utilization of the data on its computation infrastructure. Additional tasks the marketplace offers are out of the scope of this paper, e.g., payment processing.

In our work, we focus on private data marketplaces that use a privacy-preserving **computation system** to perform the computation requested by a data buyer. The number of **computation nodes** N involved in this computation depends on the cryptographic technique applied by a marketplace. The KRAKEN system builds on MPC. In MPC, the computation is performed distributed on several nodes ($N > 1$), and each node only receives a part of the user's data [Koc+20]. Alternative techniques like Functional Encryption (FE) and Full homomorphic encryption (FHE) are performed on a single node ($N = 1$, e.g., [Env22]).

In addition, our approach introduces the following additional components:

Policy Interpreter: The marketplace uses the TPL policy interpreter software component to decide if a particular buyer is qualified to acquire (a computation on) some data records. As input the interpreter takes a **TPL policy** defined by the seller for their data, as well as a set of **credentials** from the buyer, alongside some metadata about the requested computation.

7.6.2. Process

In this section, we describe a private data marketplace to which we added our TPL interpreter component. We also add a step necessary to create a policy and adapt the brokerage logic to inform users what credentials they need to provide. A graphical overview of this extended marketplace architecture is shown in Figure 7.3.

Process Overview: We split the flow into the following phases: 0. To trade their data, the seller first creates an account at the marketplace. Additionally, they receive the cryptographic material required to sell data. 1. The seller then prepares the data they want to sell. In our approach, a seller also defines the TPL policy for their data. This policy specifies who can buy the data and what types of computations the seller allows. 2. After encrypting and uploading the data to a public cloud, the seller registers the records together with the policy on the marketplace. They combine the web links to the (encrypted) data and the policy with (unencrypted) metadata describing the data. Publishing this record on the marketplace creates a so-called data product, 3. which the buyer discovers using the

marketplace catalog. The buyer then selects a set of data products from the catalog and specifies which computation they want to perform on those products. In our approach, each of the data products comes with its own policy, so there is now a set of policies that the buyer needs to fulfill. Before purchasing a computation, the buyer provides the required credentials to prove access qualifications w.r.t. the involved policies. The marketplace collects both the computation specification and the buyer's credentials. It then sends it alongside the selected products and policies to the computation system. 4. A computation system with our policy extension uses the policies and the credentials to determine whether the buyer is eligible. 5. On granted access, the system fetches the data from the clouds and performs the specified computations. After completing the computation, the system encrypts the data and returns the result to the buyer.

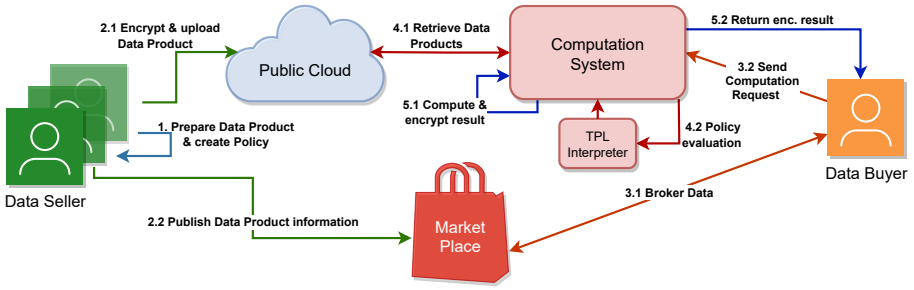

Figure 7.3.: Architecture and dataflows of a private data marketplace extended with our policy system. In addition to several modifications to the brokerage process (see Section 7.6.2), we add step 1 to create and step 4.1 to verify a policy. We also add the policy-related data, which is highlighted in red.

In the following paragraphs, we describe all six phases in more detail.

Phase 0: Setup: As a first step, users who want to become data sellers or data buyers create an account on the online marketplace. Setup steps depend on the concrete marketplace, but usually also involve the establishment of a payment channel. As a result of this phase, a new user obtains cryptographic material enabling them to create data products for the marketplace. A seller can also retrieve parts of this cryptographic material directly from a computation system they trust. Additionally, the user receives some credentials which they can use to reauthenticate at

the marketplace later (e.g., username and password, or a W3C verifiable credential [SLC22]).

Phase 1: Data and Policy Preparation: To provide some data on the marketplace, the data seller first retrieves some data they want to sell, e.g., from their local storage system or IoT devices.

As an additional preparation step before uploading the data, the seller uses a TPL authoring tool (see Section 7.5.3) to formulate their data usage policy. Primarily, this policy contains the rules about who is eligible to buy computations on the corresponding data. While providing a list of qualified buyers is the simplest option, it is not practical for a large set of potential buyers. Thus, the seller could instead restrict access to a category of qualified buyers. For example, they can require that the buyer provides a qualified certificate from a specific trust scheme (e.g., the European Union's eIDAS). In another example, the seller may restrict the type of buyer (e.g., public universities or certified medical research organizations). Further, the policy also contains the types of computations a particular buyer category is allowed to perform on the data. The seller can allow different computations for different sellers. As an alternative to formulating their own policy, the seller can browse the marketplace for existing policies and select one that suits their requirements. To illustrate the syntax and structure of a TPL policy, in Listing 7.8 we give an example seller's policy in TPL

Phase 2: Data Selling: The seller then prepares the data package for selling on the marketplace, i.e., by preparing the data and encrypting it using the cryptographic material retrieved in the setup phase. The details of this step depend on the cryptographic technique used by the specific marketplace and, thus, on the number of computation nodes N. For example, the KRAKEN distributed computation architecture ($N > 1$), the seller first splits the data into N shares. The result of this step is a data package prepared for the respective privacy-preserving computation technique. To restrict who can perform computations on the data, the prepared package is additionally encrypted for the specific computation node(s).

Additionally, to prevent an attacker from replacing the policy (see Section 7.6.5) with their own, the seller cryptographically links the policy to the data. They do so by adding a hash digest of the policy to the encrypted data package.

Afterward, the seller uploads the encrypted data package to a server, e.g.,

to a public cloud. Then, they create a data product on the marketplace by registering the web links to the uploaded data package alongside the policy and some metadata describing the data.

Phase 3: Data Buying: A buyer browses the marketplace's product catalog and selects one or several data products they want to use. Additionally, they specify the computation they want to execute on the data and initialize a computation request.

The marketplace first loads the policies of all selected data products and does an (informal) pre-check to see if the buyer's computation request is possible. The system aborts at this place if the requested computation is not possible on this data. It then computes the list of credentials required to fulfill all involved policies and sends the list to the buyer. The buyer completes the computation request by providing all the requested credentials to the marketplace, which calls the computation system. Alternatively, the buyer discovers the computation system using the marketplace, and forwards their credentials directly to the computation node(s).

Next, the computation system initiates the computation at the computation node(s). Depending on the number of computation nodes N, the computation process on the node(s) looks different. For the special case of architectures with a single computation node ($N = 1$), only this single node performs the computation. For distributed computation architectures ($N > 1$), the marketplace sends the computation request to all nodes. All nodes then perform the same operations but use their own key material and individual part of the input.

Phase 4: Policy Evaluation: After receiving a computation request, the computation node first uses the provided links to download all data packages from the public clouds and decrypts them.

Before the system launches any computation on the data (shares), it checks whether the buyer is entitled to the requested computation. For this it verifies the buyer's credentials using the policies, which it receives for each data product. Before evaluating a policy, the system checks if each policy really belongs to its data. This check is done by calculating the hash of each policy and comparing it with the policy hash inside the corresponding (now-decrypted) data package.

If this precheck is successful, the node launches the TPL interpreter. The inputs to the interpreter are the computation request and all the credentials the node received. Further, the node also provides the number of retrieved

data records to the interpreter. As a first step, the TPL interpreter checks if all involved credentials are linked to the same entity, i.e., by checking if they reference the same subject identifier [Ree+21]. This check prevents an entity from mixing the credentials of several unrelated people to fulfill the policies.

The TPL interpreter then checks if the buyer fulfills the requirements stated by all involved sellers and if the buyer is permitted to perform the requested computation. As part of this check, the TPL interpreter can download additional trust status information. During these verifications, the interpreter also assumes the task of validating the revocation status of the trust data.

Phase 5: Computation: If the interpreter concludes that all rules are fulfilled, the node(s) proceeds with executing the requested computation. After the successful computation, all nodes encrypt the computation result using the buyer's public key. Since the node extracts the public key from the buyer's primary credential, no one but the buyer can view the result.

Finally, the buyer receives the encrypted result and decrypts it using their private key. For distributed systems with $N > 1$, the buyer receives only a part of the result from each node and has to assemble them into the final result.

7.6.3. Evaluation

To evaluate the practicability of our approach, we conducted a performance analysis of our prototype implementation. We measure the impact of the additional TPL interpreter component on the time a data buyer needs to wait for a result. As a baseline, we take the performance of the MPC system used by the KRAKEN marketplace, which utilizes the MPC framework SCALE-MAMBA [Aly+21].

Network: Executing a TPL policy introduces additional network round-trips. Depending on the policy, the interpreter may establish network connections to retrieve the trust status information. The incurring delay depends on the network performance between the interpreter and the trust status registries. As a reference, we measured the latency of some common network actions in trust policies. We used the TPL interpreter's HTTPS client in our office network. Loading the eIDAS root trust status list XML[14] took us 0.3 s (± 0.209 s), while resolving an identifier from

[14] `https://ec.europa.eu/tools/lotl/eu-lotl.xml`

```
accept(BuyerCreds, NumRecords, ComputationType) :-
  extract(BuyerCreds, format, w3c_VP),
  extract(BuyerCreds, mainCredential, BuyerCredential),
  extract(BuyerCredential, format, w3c_VP),

  extract(BuyerCredential, issuer, Issuer),
  check_qualified(Issuer),

  NumRecords > 100,
  extract(BuyerCredential, organization_type, OrgType),
  acceptComputation(OrgType, ComputationType).

acceptComputation(OrgType, ComputationType) :-
  OrgType == public_university,
  ComputationType == machine_learning.

acceptComputation(OrgType, ComputationType) :-
  OrgType == private_research,
  ComputationType == simple_statistics.
```

Listing 7.8.: Example TPL policy formulated by a KRAKEN data seller. Signature and trust chain verification omitted.

the Ethereum ledger[15] takes 0.47 s (± 0.302 s). We note that several of those network lookups are identical for many policies. Thus, the nodes should be able to cache the results. For example, the TPL interpreter, per default, downloads the eIDAS trust status lists of all EU member states during initialization, so no additional network access is required for any further eIDAS trust scheme check.

Policy Execution: We used our TPL interpreter implemented in Java and measured the time for the interpreter to load and evaluate a policy. Apart from the trust status information the interpreter loads from the internet, the performance of a policy execution depends on the complexity of a policy. For benchmarking, we used the Java Microbenchmark Harness (JMH)[16] in version 1.35 and OpenJDK 16. We executed the benchmarks

[15] we use the non-production Universal Resolver at `https://dev.uniresolver.io`, measured with a simple DID lookup on 2022-05-13

[16] `https://github.com/openjdk/jmh`

on the TPL reference implementation using a Intel Core i7-8550U office laptop with 16 GB RAM running Ubuntu 21.10. The results we present in Table 7.1 show the execution of typical policies. We observed that the run-time grows linearly with the amount of executed policies. Thus for computations involving many data products, the nodes must aggregate the policies before execution to improve the run time.

Computations can take a few seconds (for simple statistics) to a couple of hours (for training a simple machine learning model), and even longer for more complex computations or larger datasets.[17]

In contrast, our implementation introduces an overhead from one to ten seconds for sensible policies. We note that our implementation is not optimized for performance.

Since policies are executed sequentially, the memory consumption only depends on the size of the policy, but not on the number of policies. In our benchmarks, each run consumes from ≈22 MB to ≈62 MB of JVM heap memory, depending on the number of predicates in a policy (see Table 7.1). The observed increase in memory consumption is mainly a result of the recursive implementation of the interpreter.

The measured timings are neglectable compared to the latency of a typical MPC computation, especially when assuming faster performance on server hardware and the continuation of Moore's Law [Gus11]. We thus argue that the performance overhead is acceptable. We note that we use the TPL reference implementation, which is not optimized for performance. Thus, instantiating our proposed architecture with an optimized policy system may further increase practicability.

7.6.4. Security Assumptions

The goal of our architecture is that a buyer who fails to fulfill the given policy for some data must never be able to launch a computation on this data. Also, the marketplace must neither be able to launch computations independently nor learn the computation results.

For MPC, the KRAKEN system uses a fully-malicious protocol, which assumes that from the set of N MPC nodes, at least one is honest [Koc+20].

[17] The KRAKEN MPC system source code is available at `https://github.com/krakenh2020/MPCService`. The system's benchmarks [KRA22] were provided to us in private by the authors.

Table 7.1.: Runtime benchmarks of the TPL interpreter.

# policies	# predicates/policy	run-time [s/op]	memory [MB]
1	3	0.08 (± 0.03)	≈22
1	20	0.09 (± 0.03)	≈25
1	100	0.12 (± 0.07)	≈62
100	3	7.68 (± 1.68)	≈22
100	20	8.18 (± 0.59)	≈25
100	100	10.16 (± 1.84)	≈62

Given this assumption, no node must gain access to the plaintext or the computation result.

Our approach builds on that trust assumption. By distributing the policy evaluation to all nodes, we ensure that one honest policy evaluator is enough to ensure that the computation is only initiated if the policy is fulfilled.

Since an honest node would never start the MPC computation if the respective buyer does not fulfill a given policy, this effectively prevents the other nodes from computing anything on the data. Thus, if the stated MPC assumption holds, the goal is achieved, and our approach can be considered secure under the same assumptions as an MPC system. This prevents both illegitimate buyers as well as a curious marketplace from initiating computations without authorization.

7.6.5. Considered Adversaries and Attacks

The goal is to prevent illegitimate entities from accessing personal data, launching a computation, or accessing the result of a computation. As adversaries, we consider (1) a buyer who does not fulfill the policy for a data product, (2) a malicious marketplace, and (3) a malicious computation node. We consider the following attacks:

A marketplace wants to access data, launch a computation, or access the result of a computation. In private data marketplaces, the marketplace platform itself has, by definition, no access to the data since the data is only shared with the platform in encrypted form. The privacy of the computation itself depends on the marketplace's architecture.

In our KRAKEN-based implementation, a marketplace can not launch a computation alone. Since all MPC nodes are needed to perform a computation, the marketplace would have to convince all the nodes that it fulfills the policy. Additionally, the marketplace cannot access the computation result because the MPC nodes encrypt the result (shares) only for the buyer. In some other architectures [Kou+20], a curious marketplace can freely launch computations on the data at will. At the same time, it can view the result of a computation launched by a legitimate buyer without much effort. Thus, such architectures require a marketplace to be trustworthy to some degree.

An adversary replaces the seller's policy with a policy they fulfill. The policies are stored at the marketplace and sent to the computation system in plaintext, together with the links to the (encrypted) data. Thus, any adversary, e.g., a curious marketplace, could send the links to data they are interested in alongside a fake policy they can fulfill. Such an attack is not possible in our design since we cryptographically link the policy to the corresponding data. We let the user add a hash of the policy to the data product before they encrypt the data authentically. If an attacker tries to replace the policy, the hashes do not match, and the computation node aborts the process.

An adversary replaces the buyer's public key to access computation results. In our MPC-based implementation, all result shares are encrypted with the buyer's public key. This public key is sent as part of a credential to the computation system. To gain access to the computation result, a malicious marketplace could try to replace the buyer's public key with their own. Alternatively, they could add a credential with their public key to the computation request. The request would then fulfill the policy using the buyer's real credentials, but the nodes would encrypt the result for the wrong public key. We prevent this attack by including a check in the seller's policy that ensures that all credentials belong to the same identity. Since the public key is extracted from one of the credentials, only the legitimate buyer's public key is used to encrypt the computation result.

Chapter 7 Conclusions

In this chapter, we introduced our extensible trust- and access-policy system TPL. Using TPL, verifiers can define rules about their understanding of trust as well as other business-logic rules in a flexible way. We discussed the integration of the TPL system into our global trust management system, which we introduced in Chapter 6. Further, as a use case for TPL, we presented an access control extension for private data marketplaces. This extension enables data sellers to define expressive policies on their data usage. The results of the research presented in this chapter have been published in three academic papers [Möd+19; Alb+21; MA22].

8

Transforming Credentials between Representations

This chapter is based on the paper *Trust Me If You Can: Trusted Transformation Between (JSON) Schemas to Support Global Authentication of Education Credentials* by More, Grassberger et al. [Mor+21]. Parts of this paper have been copied verbatim. The prototype was implemented as part of the bachelor's thesis of Dominik König.

Upon retrieval of a credential, a Service Provider (SP) needs to assess the credential's authenticity (see Chapter 6); additionally, the SP also needs to be able to interpret the credential's content. This works if the credential issuer and SP share an understanding of an encoding schema, but it becomes a problem when they do not. When credentials are issued and verified in a global and heterogeneous setting, a diverse set of data formats and schemata are used to encode the certified information.

The format and schema of digital credentials play an essential role in their global applicability and compatibility with different systems. **Credential formats** refer to the structure and syntax of the credential data, such as the file type used (e.g., XML). **Credential schemata** refer to the description of the credential data, such as the structure of attributes and their values (e.g., a specific XML schema). The choice of format and schema can impact the interoperability of digital credentials, as different systems may have different requirements and standards.

This potential lack of agreement represents a considerable challenge to automated processing. The SP's automated verification tool needs to understand the credential's format to be able to parse it (see also Figure 6.2). Further, as shown in the example in Figure 8.1, an SP's access policy relies

on a specific credential structure to define access rules (see also Chapter 7). If the incoming credential is of a different format or schema, the policy system cannot check it. The underlying problem is that a verification tool always needs to understand the semantics of a credential to process it.

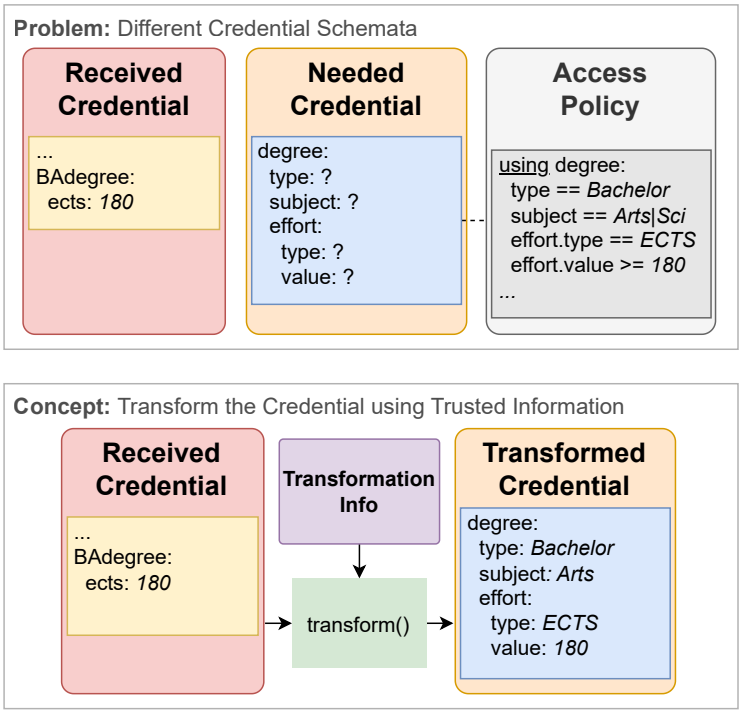

Figure 8.1.: If a SP receives a credential in a format/schema it does not understant, it first needs to transform it.

To support entities who want to verify credentials from various issuers, we introduce a system to interpret credentials issued in an unknown schema. Our system does so by transforming digital credential data between schemata and formats. This transformation is done automatically using transformation information that the SP's verification system executes. The transformation itself is performed using existing transformation languages or template systems. However, transforming a credential invalidates its signature. Hence, our system automatically authenticates the transformation information before executing the transformation. It can easily do so using our global trust management system (see Chapter 6).

Given this then-trusted transformation information, SPs can locally transform a credential from the issuer's schema or format into their preferred schema. This transformation results in a credential in a schema and format that the issuer's policy system understands.

By implementing **trusted credential transformations**, issuers and SPs can exchange digital credentials in an interoperable way, without requiring them to use the same format and schema.

Chapter 8 Goal:

We discuss the overall goals of this thesis in Chapter 5. In this chapter, we focus on Goal 2 (Global Trust Scheme Interoperability). More specifically, we tackle the aspect of Attestation Format Interoperability. From this goal, we derive seven requirements to survey the state of the art and evaluate our approach (see Section 8.2).

Chapter 8 Outline:

We start the chapter with a conceptual introduction into credentials, discussing the difference between credential formats and schemata (Section 8.1).

We then define requirements that our thesis goals pose on a credential transformation system (Section 8.2). In Section 8.3, we use these requirements to survey the state of the art and argue why a new approach is needed.

In Section 8.4, we present our approach for flexible credential transformations. In this section, we also introduce the relevant concepts and the two types of a transformation we consider: the transformation of a credential's schema and the transformation of both the credential's format and schema. Additionally, we discuss the implications of a transformation on the credential's authenticity.

In Section 8.5, we show a proof of concept implementation of our transformation approach. Our implementation builds on our Distributed Ledger (DL)-based trust management system, which we introduced in Section 6.6 as alternative to our DNS-based system. In this implementation, the transformation information is stored on a decentralized storage layer and authenticated using a trust registry stored in a distributed ledger. This registry also enables all involved entities to publish trust statements about each other, announcing to their trusted peers which other entities they consider legitimate publishers of transformation information.

In Section 8.6, we evaluate out trusted credential transformation system against the requirements that we previously introduced. We also discuss our approach and state ideas for future work.

8.1. On Credentials

Credentials are documents commonly used to prove one's identity, capabilities, or the fulfillment of requirements. Examples of credentials are a passport, a driver's license, or a public transport ticket. Other examples are credentials issued for reaching achievements, e.g., course certificates and university diplomas.

A digital credential is the electronic equivalent of a paper-based credential. Simple digital credentials are username and password, e.g., used to authenticate at a website. In the context of this thesis, a credential is a signed data structure issued to some user (the **holder**) [BT11, p. 46]. In more detail, a credential consists of some **attributes** (sometimes also called "claims") about a **subject**. Bertino et al. define an attribute as "a set of data that describes the characteristics of a subject" [BT11, p. 22]. Those encoded attributes are electronically *signed* by an **issuer**. For a credential to have any relevance, it must be issued by an issuer qualified to do so. We also discuss this in Section 2.2.

Many solutions exist to create, manage, and verify digital credentials with different characteristics [BRA23]. In this chapter, we focus on the representation of credentials as authenticated containers for attributes. We define an attribute as a key-value pair. While the key is typically a string (i.e., the attribute's name), there are many potential types of values. Since a credential commonly involves more than a single attribute, it has some structure. This structure can be a flat list of all attributes or a more complex arrangement. For example, some credential systems allow that the value of an attribute consists of attributes again ("nesting"). The list of attributes and the structure of a credential depends on the specific use case.

When we create a credential, we thus first need to represent it on an abstract level, then encode it to a concrete byte string. In that context, we differentiate between a **format**, a **schema**, and the **encoding**:

- A format refers to the specific syntax and structure used to represent the data. Formats[1] commonly used to represent credentials are JSON, XML, and X.509.
- A schema defines the rules and constraints for the data's structure and content. In the context of this thesis, the word *schema* refers to the formal description of a document's structure [Wal01]. A schema itself is specified by some system (in a special format or by a technical standard). For example, "JSON Schema" and "XML Schema Definition" (XSD) are examples of schema systems that specify the structure of a JSON or XML document, respectively. Further, in X.509, "profiles" define the structure and content of X.509 certificates. An example of an X.509 profile is PKIX, which is commonly used in the web Public Key Infrastructure (PKI) for Transport Layer Security (TLS) authentication (profiled in [Coo+08], see also Section 4.2).
- The encoding of a credential is the concrete byte sequence representing some data.[2] While JSON and XML are formats with their own encoding, X.509 is an abstract representation that needs an encoding format like Distinguished Encoding Rules (DER) [Hou+99; ITU02]. This byte sequence can then be stored or transmitted over the network. Additionally, the attributes' encoding is also required for the signing of a credential, as the signature function signs a concrete byte sequence. Some credential systems use the same attribute encoding for signing that they also use to store/transmit the credential (e.g., JWT, X.509, or OpenPGP) [JBS15; Coo+08; Cal+07]. Other systems apply a canonicalization to the credentials to ensure the input of the signature function is always the same (e.g., XMLDsig) [Bar+13, Section 4.4.1].

The choice of encoding mechanism and schema depends on the specific use case.

8.1.1. Common Credential Formats and Schema Systems

Table 8.1 gives an overview of common formats, schemata, and encodings. Relevant credential formats in the context of this thesis are XML and

[1] In this chapter, formats refer to file formats, not to be confused with TPL formats (cf. Section 7.3).

[2] The encoding of data is also concerned with the encoding of characters (e.g., ASCII or UTF-8), but this is out of the scope of this thesis.

JSON. XML is a format and a meta-format (used to define other languages like XHTML and SVG). Common schema systems used with XML are *XML Schema Definition* (XSD) and *document type definitions* (DTD). For JSON, *JSON Schema* is a JSON-based schema for describing the structure of JSON data.

In addition, X.509 is a common format for digital certificates. The structure (schema) of an X.509 certificate is defined using a *profile*. In contrast to XML and JSON, X.509 certificates are represented in the abstract ASN.1 notation. A certificate can then be encoded using different encodings; however, certain X.509 profiles might require a specific encoding. For example, the IETF's PKIX profile requires the DER encoding (or its base64-encoded PEM form). Another relevant format is the OpenPGP message format (which specifies a package format to encode certificates with attributes) [Mor15].

Table 8.1.: Data formats commonly used to represent digital credentials.

Meta	**Format**	**Schema System**	**Encoding** (examples)
XML	XML	XSD, DTD	XML
-	JSON	JSON Schema	JSON, JWT, CBOR
ASN.1	X.509	X.509 Profiles	DER, PEM, CBOR
ASN.1	PKCS, LDAP, ...	...	DER, CBOR, XER
ASN.1	CMS		BER
-	OpenPGP Format	OpenPGP	OpenPGP Packets
-	-	-	XMLDsig, CL-Sigs

8.1.2. Terminology

As some terms in this chapter overlap with similar terms in other chapters, we now provide a clarification:

- In this chapter, we talk about the transformation of **credentials** (as defined above). In other chapters, we use the more general word **attestation** to denote a signed data structure of any kind. Another type of attestation is a **certificate**, which we defined as a data structure binding a key to some identifier. Following this definition,

certificates and credentials are both attestations. This differentiation is also discussed in Section 2.2.

- With **schema** (plural: schemata), we denote a data schema (as described above). In contrast, the word **scheme** is used for trust schemes (see Section 6.1).
- With **format**, we denote a file format (like XML or JSON). In Chapter 7 we also use the word "format" for **TPL formats**, which are parsers from concrete (data) formats into the abstract TPL format used by the TPL interpreter.

8.2. Requirements

This chapter focuses on the goal of Attestation Format Interoperability (see Chapter 5). To fulfill this goal, we require a system that can transform a digital credential from various representations into a representation that the SP can process. This section outlines the requirements for such a credential transformation system. We derive seven requirements from the overall goals considered in this chapter (R1–R7).

In the following Section 8.3, we then use those requirements to survey the state of the art. Later in Section 8.6.1, we also use the requirements to evaluate our own approach.

8.2.1. Thesis Requirements

Based on the overall goal of our thesis (see Chapter 5), we derive the following requirements for a credential transformation system.

R1 Attestation Format Interoperability: The main reason for this chapter is the need to support many credential formats, i.e., signed encodings of attributes (Goal 2). The goal is that an SP can process a credential, even if it was encoded in a format unknown to the SP. To fulfill this requirement, we thus require a method or a system that helps an SP to understand the semantics of such a credential. This is the basis for further processing, e.g., with an access policy system.

R2 Trust: When the SP receives a transaction and attempts to transform it, the transformation system must ensure its trustworthiness. If the

transaction was trustworthy before transforming it, then the system must ensure that it can also be trusted after the transformation. Conversely, if no trust can be established in the incoming transaction, then the result of the transformation must also be marked as untrustworthy. Hence, all entities and artifacts involved in the transformation must be trusted by the SP.[3] Further, all information needed to transform a credential needs to be retrieved from trusted parties.

Since each SP has its own perception of trust (Goal 1), it is desirable to assess the trustworthiness of transactions using trust policies (see Chapter 7).

R3 Local Trust View: In the global context highlighted by Goal 2, we consider a heterogeneous environment with many trust schemes, issuers, and credential formats. Conversely, we aim to enable SPs to enforce their own perception of trust (Goal 1). A likely implication of these goals is that the issuer and the verifier don't know each other and thus cannot agree on a credential encoding. Hence, the system must neither depend on the issuer to issue a credential in the correct format nor assume that the issuer can create such a credential on demand.

R4 No modifications of the issuer: The goal of our approach is to facilitate interoperability (Goal 2) between existing systems as well. Hence, to increase the acceptance of the system,[4] modifications of the issuer's systems must be avoided. Specifically, the system must not assume that the issuer adapts a novel signature scheme.

Given our focus on local perceptions of trust and requirement R3, we *don't* restrict modifications of the verifier's systems.

R5 Extensibility: To comply with the extensibility Goal 3, the system must not limit the credential formats it supports. Further, to facilitate automated processing, the discovery and installation of new credential formats should also happen automatically. Hence, the need for manually installing components for each format should be avoided.

[3] The software security of the local transformation system is also vital for a transformation's trustworthiness but is out of the scope of this thesis.

[4] and to avoid an xkcd 927 situation

R6 Confidentiality & Undetectability: One of the goals of this thesis is to preserve users' privacy (Goal 4). This includes the potentially sensitive content of credentials and information about the user's behavior. To fulfill this goal, the system must not reveal the content of credentials or other identifiable information (i.e., PII) of the user to any third party. This restriction includes the credential issuer and the issuer's trust scheme operator.

R7 No execution of third-party code: A credential transformation might involve a third party supporting the transformation, e.g., by providing credential format information to the SP or fully executing the transformation. Such an entity is trusted by the SP and is relied on for transformations of potentially very important information. Nevertheless, to minimize the attack surface and reduce the impact of a malicious or compromised entity, the SP's system should not execute any code it retrieved from third parties. Here we distinguish between the execution of any (Turing-complete) code and the transformation using, e.g., a less powerful template language that only describes a mapping.

8.3. State of the Art

We now survey the state of the art of credential format interoperability, focusing on credential transformation approaches. We consider techniques and tools that can be used to transform a digital credential (i.e., signed data structure) from one encoding schema to another one. The focus of the survey is on systems that comply with our requirements. To systemize our survey, we first introduce four types of approaches to transform a credential. We argue why only the last type is suitable w.r.t. our core requirements. Hence, we then continue with a discussion of the state of the art of this last type.

8.3.1. Types of Credential Transformation Approaches

Given the typical credential flow (as discussed in Chapter 2 and Section 6.1), we consider four types of credential transformations, depending on which entity is performing the transformation process.

Type 1: Transformation by the Issuer: This is the simplest case for interoperability: If the issuer creates a credential in an encoding that the verifier can understand, then there are no interoperability issues w.r.t. credential formats. However, as we focus on a heterogeneous context in which issuers and verifiers don't agree on a credential format, this type violates R3.

A similar approach is to later entrust an issuer with the transformation of a credential it issued. This raises similar issues (violating R3) and additionally violates the undetectability requirement from R6. Hence, since approaches of this type already conceptually prevent compliance with our requirements, we did not consider issuer-based approaches.

Type 2: Transformation by the Holder: Once the credential is signed and sent to the holder, the following entity in line to transform it is the holder itself. For example, Glaude recently proposed that the holder's digital identity wallet acts as an adapter to transform credentials to the form needed by the verifier [Gla23]. However, as Young points out, this breaks the credential's authenticity [You23], violating the fundamental trust requirement R2.

The W3C Verifiable Credential (VC) data model proposes a mechanism that would allow the transformation of a credential by its holder using zero-knowledge proofs [SLC22, Section 5.8]; this is similar to what we propose in Section 8.6.4. The mechanism discussed in the VC data model requires that the credential issuer uses a suitable signature scheme, e.g., CL signatures [CL02; SLC22, Figure 11], which excludes existing credentials and issuers, and thus violates R4.

Type 3: Transformation by a Trusted Third Party (TTP): If two systems want to communicate but don't agree on a data exchange format, a third system can transform the data for them. In the distributed computing field, this is known as message-oriented middleware (MOM) [CCL08]. While MOMs enable SPs to process incoming data, they do not provide any trust in that data (violating R2).

To mitigate this issue, the concept of MOMs—or, more specifically, data transformation TTPs—can be extended to consider the data's authenticity. For example, Young recommends a proxy issuer approach similar to notaries [You23; Tru21]. Such a proxy issuer takes a credential in

some format, converts it to another format, and attests to the semantic equivalence and the credential's trustworthiness to the SP. The SP then trusts the TTP to only attest credentials with a valid signature and can hence also trust the transformed credential. This approach is conceptually similar to Validation Authorities, i.e., entities to which SPs can outsource the trust verification of a transaction (see Chapter 5 and Chapter 6). The downside of this type of approaches is that sending the full credential to a TTP violates the privacy requirements from R6. It also impedes the requirement of a local trust view R3.

Type 4: Transformation by the Verifier: After the holder transmits the credential to the verifier (SP), it is up to the verifier itself to establish an understanding in the credential, i.e., transform it. This type of approach has the advantage that no modifications to any other component are needed. Further, the verifier can authenticate the credential before transforming it using their own trust perception.

To fulfill the extensibility requirement R5, approaches of this type might download executable code to perform the transformation. Doing so can partially violate the security requirement R7. We discuss this challenge in the evaluation in Section 8.6.1 below.

8.3.2. Results

Since we argued that only approaches of type 4 can fulfill our requirements, we now continue with a survey of approaches of this type.

Type 4 approaches: There exists a large body of techniques and tools to locally transform a document from one form into another.

In the context of our goals, a transformation system is a system that performs the transformation based on some mapping template. A commonly used system for this is the Extensible Stylesheet Language Transformations (XSLT) language[5]. XSLT is designed to transform XML documents into other XML documents (of a different schema) or documents of another format. In XSLT, a transformation is performed by an XSLT processor, which takes a structured (XML) document and an XSLT stylesheet as input, and outputs the transformed document. Systems similar to XSLT

[5] `https://www.w3.org/standards/xml/transformation`, accessed on 2023-04-20

exist for other languages, for example, jsonpath-object-transform[6] (for JSON), or more general approaches like Liquid templates.[7] However, those transformation systems just transform a document and don't consider its authenticity, violating the fundamental trust requirement R2. Similarly, they rely on a pre-configured transformation template, which violates the extensibility requirement R5.

The FutureID project deals with the translation between different eID credential formats using a broker service [Fut19b; Sel+19, Section 8.1]. To do so, FutureID introduces the concept of simple credential transformers (SCTs) that can handle different types of credentials during authentication. A SCT "validates the credential. If successful, the SCT [creates] an authenticated session and sets the identity attributes that were provided with the credential and then redirects the user to the originally requested resource" [Fut14]. While this ensures the trustworthiness of credentials, a separate SCT needs to be installed for each credential format (violating the extensibility requirement R5).

Although all of those approaches violate several of our requirements, they provide a valuable building block for a trustworthy transformation system.

Other approaches: The Resource Description Framework (RDF)[8] is another approach that can be used to solve the credential format interoperability problem. RDF is a graph-based data model used to encode information about resources. This approach is similar to what credentials do. Example encodings for RDF are JSON-LD (JSON for Linking Data)[9] and Turtle.[10] An interesting aspect of RDF is that it uses Uniform Resource Identifiers (URIs) to identify those resources. For example, when applying this idea to credentials, the name of an attribute is encoded as a URI. Hence, an advantage of RDF-based approaches is that no credential transformation system is needed. However, the basis for this is that issuers create credentials with semantic annotations (e.g., a JSON-LD context), i.e., map attributes to URIs. Further, all entities in the system need to agree on a shared context vocabulary for those URIs (e.g., schema.org). Since this assumes a global agreement (at least between the issuer and

[6] `https://github.com/dvdln/jsonpath-object-transform`, accessed on 2023-04-20

[7] `https://learn.microsoft.com/en-us/azure/logic-apps/logic-apps-enterprise-integration-liquid-transform`, accessed on 2023-04-20

[8] `http://www.w3.org/standards/techs/rdf`, accessed on 2023-07-29

[9] `https://www.w3.org/TR/json-ld/#relationship-to-rdf`, accessed on 2023-07-29

[10] `https://www.w3.org/TR/turtle`, accessed on 2023-07-29

verifier), RDF approaches violate our interoperability requirements R1 and R3. Nevertheless, RDF can be used in addition to a transformation system, providing additional semantic information to the SP.

Conclusion: The result of our survey is that no single approach fulfills all our requirements. However, we assembled a collection of valuable building blocks. Specifically, compliance of a local transformation system (e.g., XSLT) with our requirements can be achieved if it is extended to discover transformation mappings automatically and only relies on trusted data. We investigate this idea in the rest of this chapter.

8.4. Trusted Credential Transformation

Issuing digital credentials in a machine-readable form requires the issuing institution to decide on the format and schema it uses to encode the credential's data. Both format and schema need to fulfill various requirements depending on the context. For example, the schema of a university diploma credential depends on the structure of the education system, the contents of a course or study, and legal regulations. Conversely, the SP verifying such credentials needs to understand the formats and schemata used therein. Thus, the issuer and SP have to not only agree on file formats, but also on the schema to represent the credential data, which is a challenge.

To mitigate this issue, we introduce **trusted credential transformations** to enable compatibility between different credential formats and schemata. Credential transformations involve mapping the data from one schema to another, allowing for the exchange of credentials between different systems. This process consists in identifying the attributes in the source schema and mapping them to corresponding attributes in the target schema. This mapping is encoded in machine-readable form—the so-called *transformation information* (TI). The transformation process can then be achieved using existing techniques, such as XSLT or jsonpath-object-transform (see also Section 8.3). In addition to mapping a credential to a different schema, a transformation can also map the credential to another format (e.g., XML to JSON, using XSLT). To trust the result of a transformation, the TI must be retrieved from a qualified entity. Hence, the focus of this chapter is not the transformation itself, but how to perform it in a secure and trustworthy way.

Authenticity and Trust: The signature on the credential protects its integrity and authenticity. However, transforming the credential to another format changes the content, thereby invalidating the signature. Thus, the SP must check the signature—and signer—*before* it transforms the credential. This check is easily possible if the transformation only involves the schema, as the signature mechanism does not change. In that case, the SP can also directly authenticate the credential's issuer using its trusted trust scheme. If the credential's format is also transformed, the signature verification requires additional work. For example, an SP that only knows how to handle credentials in JSON format cannot verify XML signatures.

Additionally, the SP needs to authenticate the transformation information. This information describes how to transform a credential from the issuer's schema into a schema known to the SP. Since the SP wants to trust the result of the transformation, it also needs to trust the transformation information. This trust is enabled by using the trust scheme to authenticate the information.[11] For example, this can be done in a similarly to the authorization of qualified issuers (see Section 6.1 above). The transformation information is then retrieved and authenticated by the SP and used to transform credentials. We visualize the transformation process's conceptual trust model in Figure 8.2.

The authentication of the credential and the TI is possible in an automated way using our global trust management system, which we introduced in Chapter 6. The result of the transformation process is a credential in a schema (and format) that the SP understands, transformed from a trusted credential using trusted transformation information. Afterward, this credential can then be automatically processed based on rules locally defined in the SP's policy (see also Chapter 7).

8.4.1. Concepts

Our approach builds on an existing trust management system. For example, it can be applied to the global system which we introduced in Chapter 6. Hence, we extend the actors defined in Section 6.1 and introduce the following concepts:

The **Transformation Engine** is the software component that performs the transformation process (denoted with `transform()` in Figure 8.2).

[11] If the publisher of the transformation information is a foreign trust scheme, a trust recognition must first be used to establish trust in this scheme (see Section 6.3).

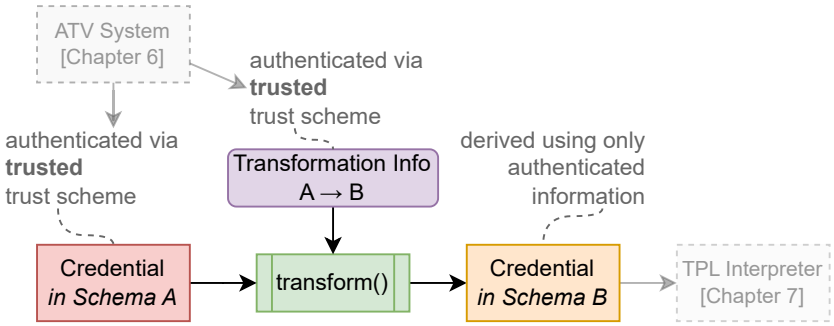

Figure 8.2.: Conceptual trust model of the Credential Transformation process and interaction with the other components of our thesis.

The SP operates it as part of the trust verification process. The concrete choice of an engine depends on the involved schemata and formats.

Transformation Information (TI) is a set of (machine-readable) transformation rules that define how to transform a credential issued using one schema into a credential using another schema. In the same way, TI can describe rules on how to transform a credential between two formats. It is published by a trusted entity, and later retrieved and authenticated by the SP's verification tool.

TI is encoded in an implementation-specific format. The concrete encoding of the TI is specific to the transformation engine and thus needs to be one that the SP understands. But, this does not require a global standard for transformation systems. Instead, only transformation publishers and SPs who want to collaborate need to agree on such a format. This agreement can happen independently of any credential issuers. Additionally, it is even possible that different transformation engines are used for different credential formats.

TI only takes care of the transformation of the actual data but assumes that the data has been authenticated before transforming it. If the format is also transformed, more information is needed to verify the credential's signature.

Verification Information (VI) is an optional extension of transformation information. It describes how to authenticate an incoming credential. To do so, VI consists of functionality to handle public-key material and

verify signatures. VI is published alongside TI by the same publisher.

As an alternative, the TI and VI to transform the format and authenticate the credential can be published by a different entity than the TI to transform the schema. For example, a foreign trust scheme might publish the VI explaining how to verify XML signatures and the TI to transform XML documents into JSON, while the local scheme publishes the schema TI between two JSON schemata.

TI is encoded as a static file that describes the mapping (e.g., a template). The transformation engine uses the TI file to map the attributes between the schemata. In contrast, VI is encoded and published as directly executable code. We show an example TI encoded as a JSON template in Listing D.1 in the appendix.

A **Schema Identifier** is a URI identifying the schema of a credential. It is optionally part of the credential, e.g., as an attribute. In contrast, a **Format Identifier** is not part of the credential itself, but instead derived from the credential's file extension (e.g., `.xml` and `.json`), or from other context information (e.g., MIME type). The SP needs both identifiers to select the correct TI and VI.

Depending on the schema/format, identifying the type without knowledge about it might lead to a chicken-and-egg situation. E.g., extracting the schema identifier from a credential requires knowledge about that credential's schema, i.e., the attribute used to store the identifier. In this case, an SP might use all its available transformation engines to attempt to parse the credential.[12] If successful, the respective engine can extract the identifier and verify if it is responsible for this schema. Alas, this approach does not work if the respective engine is not yet available locally at the SP.

An alternative approach is to agree on parts of the schema (almost making it part of the format) and only leave the credential's *payload* open to the specific schema. For example, this is commonly done in the VC data model [SLC22]. The VC recommendation only specifies the general structure of a credential, most notably the `type` attribute. This attribute contains the schema identifier of the credential. Applications then specify this type/schema identifier, which describes the structure of the credential's payload, stored in the `credentialSubject` attribute (see also Appendix D).

The **Transformation Data Registry (TDR)** is a storage system used

[12]This is supported by our TPL format library, as introduced in Chapter 7.

by all system participants to publish, discover, and retrieve TI and VI. For example, in the case of a DL-based trust scheme (see Section 6.6), the TDR can be a smart contract on a ledger and a suitable storage system. Alternatively, the TDR could be built on our Domain Name System (DNS)-based trust system introduced in Chapter 6. For example, the trust scheme operator (whom all trust scheme members already trust) could operate a TDR for its members. Or, a trust recognition can be used to establish trust in a foreign scheme operator and use its TDR (see Section 6.3). Hence, a single global TDR is unnecessary, in the same way as there is no need for a worldwide agreement on a TI encoding (see above).

8.4.2. Types of Transformations

Credential transformations can be categorized into two types, as also visualized in Figure 8.3.

A **Schema Transformation** transforms the credential from one schema to another schema of the same format. An example is the transformation of a JSON credential into another JSON credential of a different structure. Before verifying the credential, the SP needs to check the signature on the credential (and authenticate its signer). Since the SP in that example already knows how to verify the JSON signature, the schema transformation process relies only on TI.

In contrast, a **Format Transformation** also transforms the format of the credential. This is, for example, needed if the credential is issued in XML format, but the SP can only verify and parse JSON-based credentials. In this example, the SP also needs to gain the knowledge of how to verify the signature on the (unknown) XML format. Hence, to establish trust in such a credential, the SP requires VI in addition to TI. By executing the verification and transformation process, the transformation engine first uses the VI to check the (XML) signature. It then uses the TI to transform the credential into a JSON credential, which the SP can trust and process further.

8.4.3. Process

The transformation process described in this section helps SPs who are unfamiliar with the schema of a credential they receive and, therefore, need support to interpret the credential correctly. We split this process into two

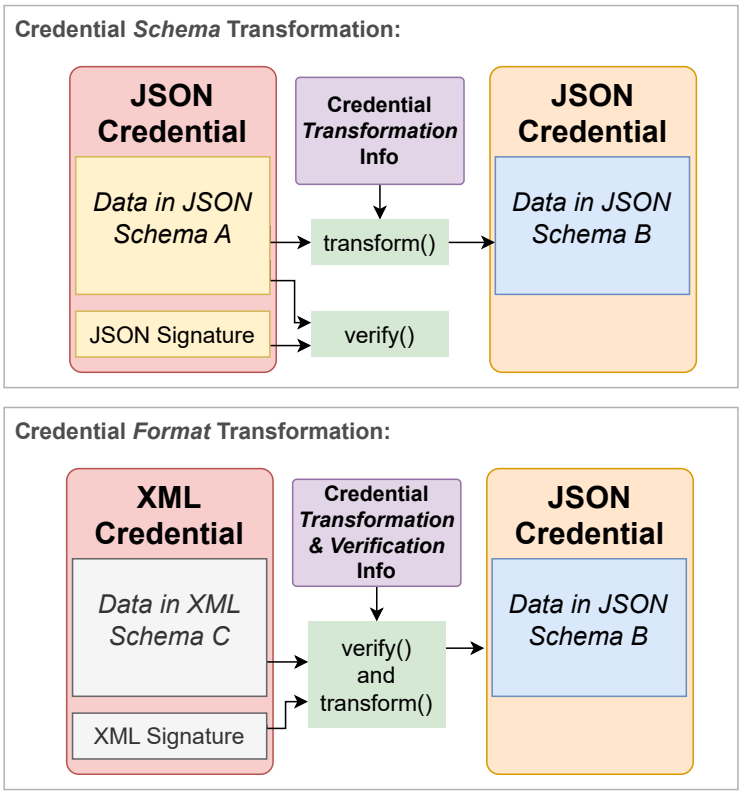

Figure 8.3.: Types of Credential Transformations: transforming just the Schema (e.g., two different JSON schemata), or the full Format (e.g., XML to JSON).

phases. In the first phase, entities knowing how to interpret a credential's schema and transform it into another schema encode this information as TI. The prospective transformation publisher then publishes this information in the registry (TDR). For example, a university having dealt with credentials in a foreign schema before may encode and publish this knowledge as transformation information (TI). If the credential's format also needs to be transformed, the transformation publisher additionally publishes the corresponding verification information (VI). In the second phase, an SP retrieves this TI (and optional VI) from the Transformation Data Registry (TDR) to transform and verify a credential. In doing so, the SP also authenticates the retrieved information.

Phase 1 **Publication Process:**

The preparation of a credential transformation starts when a *transformation publisher* knows the mapping between two credential schemata. The publication can either happen by an entity in the transformation's target or source.[13] In the case of a **publication by a target entity**, this knowledge is about how to map a foreign schema to a local schema. This can, for example, happen when an institution needs to authenticate a digital diploma issued by a foreign university. In doing so, the local institution manually maps the credential's schema into its local schema. To save other institutions from this tedious process, it wants to share this mapping. An example of this publication is shown in Figure 8.4 below. In the opposite case of a **publication by a source entity**, the entity knows how to map from its local schema to a foreign schema. For example, the trust scheme authority of `ExampleCom` scheme might want to facilitate transactions between its members and members of the `FantasyLand` trust scheme. To facilitate outgoing transactions, it can publish TI and VI to a schema and format used in the `FantasyLand` trust scheme. In both cases, the publication happens by an entity trusted by the SP that later wants to use the information.

The transformation publisher then encodes this VI and TI in machine-readable form. After encoding the information, the transformation publisher signs the TI and VI. This essential step enables an SP to verify the authenticity and establish trust in the information.

Finally, the transformation publisher publishes the signed information on the TDR. The concrete publication protocol and registry location depends on the use case and trust scheme (see Section 8.4.1). The identifier of the transformation information is constructed from the identifiers of both source and target of the transformation. The encoding of this identifier also depends on the type of TDR. For example, a use case building on our DNS-based trust scheme publication discussed in Chapter 6 might use a DNS-based TDR. In that case, the TI identifier is constructed by concatenating the DNS encoding of the schemata identifiers and the publisher's DNS identifier.

[13] In this context, "target" and "source" don't necessarily reference separate trust schemes. Entities can have knowledge about different credential schemata (or formats) used in the same trust scheme.

Phase 2 **Transformation Process:**

When an SP receives a credential encoded using a schema or format it does not understand, it must transform the credential. Before transforming the credential, the SP needs to retrieve the corresponding transformation information. To do so, the SP first determines the *format* identifier of the received credential (see also Section 8.4.1). Using this format identifier, the SP determines the type of the required transformation (see Section 8.4.2).

Type 1: Schema Transformation: If the format identifier matches with an identifier that can be directly processed by the SP's verification system, no format transformation is needed. In that case, the SP can directly verify the signature on the credential. While doing so, it also authenticates the signer of the credential using its trust policy (see Chapter 7). After trust in the received credential is established, the SP proceeds by determining the *schema* identifier of the credential (see Section 8.4.1).

The SP then uses this schema identifier to retrieve suitable TI. This retrieval is done using the credential's schema identifier (the *source* identifier) and the identifier of a schema known by the SP (the *target* identifier). The target identifier is selected from a list of schemata that are relevant to the current use case. For example, the registrar's office of a university might choose all diploma schemata its policy system can check. Combining both identifiers results in a "TI identifier" that the SP uses to query the TDR and retrieve the TI. If no such TI exists at the TDR, the SP proceeds with another target schema it understands. If the SP cannot find a suitable TI, it aborts the process and rejects the credential.

Next, before executing the TI, the SP must establish trust in the retrieved TI. This is done by verifying whether a trusted and qualified entity published the information. To check this, we build upon the SP's local trust scheme. The details of this step depend on the concrete use case and TDR architecture. For example, if the system uses a DNS-based trust scheme (as introduced in Chapter 6), the scheme's authority could be qualified to issue such information. The SP then uses the DNS to retrieve and trust the signing keys of the scheme's authority. Or, the authority might authorize other entities in the scheme to publish TI. Another example is the use of a distributed TDR, like on our smart contract-based web of trust (as discussed in Section 6.6). In that example, the WoT is used to authenticate the TI's publisher and retrieve its keys.

After authenticating the TI, the SP proceeds with executing it using the

transformation engine (as shown in Figure 8.3). The result is a (unsigned) credential in a schema that the SP can process further.

Type 2: Format Transformation: If the credential's format identifier does not match an identifier known by the SP, a format transformation is needed. In that case, the SP cannot read the credential before the transformation, so no signature check is possible. Instead, VI is needed to authenticate the credential and its signer.

To discover a suitable TI, the SP uses the credential's format identifier. In addition to TI, the SP also requests VI to verify the credential's signature.

The retrieval of TI and VI works in the same way as with the TI alone (see Type 1 above): the SP generates the information identifier by using the source and target identifiers and queries the TDR. It then uses its local trust scheme to authenticate the TI and VI. After retrieving and authenticating the information, the SP executes it.

During the execution of the TI, the credential is transformed to the target format, and the SP can continue processing it. Additionally, the VI contains executable code to also verify the signature of the credential. This verification code checks the signature before the credential is transformed. The code outputs the verification result as well as the public key of the signer in a format that the SP understands. The SP uses this public key to authenticate the credential's signer using its trust policy.

Type 1 & 2: A special case occurs if both format and schema need to be transformed using different information. This case happens if the format transformation yields a credential in a suitable format but an unknown schema. In that case, an additional TI is needed to transform the schema. This is done by executing Phase 1 after Phase 2, with the (now unsigned) credential in the correct format as the input credential.

8.5. Prototype

To show the feasibility of our concept, we implement a prototype using our DL-based trust management system, which we introduced in Section 6.6. This system deals with credentials in VC format issued in a heterogeneous setting. The data inside the VC envelope is encoded following a schema chosen by the issuer. In Chapter 6 above, we discussed the case where a credential is issued in a JSON schema that the SP's verifier understands. If this is not the case, the verifier needs to first transform the credential

to a schema known by them. To facilitate this transformation, we extend the smart-contract-based registry with a TDR. In our architecture, this is done as a *transformation graph* that stores pointers to TI files. An SP then uses the transformation graph of the decentralized registry to discover suitable TI. This information is first authenticated using the Web of Trust (WoT) graph and then used to transform the incoming credential. Since a) the signature on the credential is verified before the transformation, b) the transformation information is authenticated, and c) the SP itself does the process, the result remains trustworthy. We discuss our system in more detail in [Mor+21].

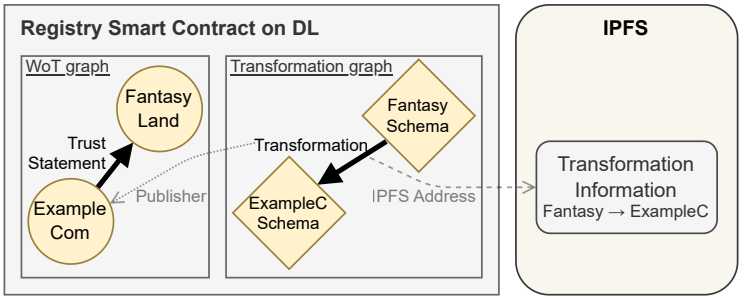

Figure 8.4.: Architecture of our DL-based trust registry with credential transformations following the example from Section 8.4.3. The authority of the `ExampleCom` (target) scheme publishes TI for a transformation from `FantasyLand`'s schema (source) to its own `ExampleC` schema on the transformation graph. The TI itself it stored on IFPS and references from the graph. Additionally, the `ExampleCom` authority publishes a trust statement about the `FantasyLand` trust scheme.

Distributed Storage Layer: To host our decentralized registry, we develop a smart contract in the Solidity programming language[14] and deploy it on a private Ethereum ledger. The contract stores the trust graph and the transformation graph as a list of edges.

Storing large data on a DL is often not practical due to storage constraints and costs. We thus use a decentralized storage system to publish the

[14] `https://docs.soliditylang.org`, accessed on 2023-04-13

actual transformation information. For the decentralized storage of transformation information, we use the InterPlanetary File System (IPFS), since its architecture works well together with a distributed ledger (see also Section 4.3). Furthermore, IPFS provides integrity protection by addressing files based on their content.[15] The transformation graph then only stores the information's IPFS content identifier (CID). This architecture is illustrated in Figure 8.4. Listing 8.1 shows the TI storage part of the registry smart contract.

```
TransformationEdge[] public transformationGraph;

struct TransformationEdge {
  uint id;
  string publisherDID;
  string sourceSchema;
  string targetSchema;
  string ipfsCID;
}
```

Listing 8.1.: The transformation graph storage in Solidity.

Client Components: To help users with adding and retrieving TI edges from the registry, we develop client components using the *web3.js* library. Those components take a user's call to the registry contract, process it using an Ethereum wallet, and send it to a node of the ledger. They also retrieve and visualize the current state of the graphs.

Transformation Engine: TI is discovered using the transformation graph and authenticated using the WoT graph, as explained above. The TI can be encoded by any means as long as the SP's verification tool is able to execute them. In our implementation, we used *jsonpath-object-transform* [Lan+17] and published corresponding templates encoded in JSON to IPFS. Listing 8.2 illustrates the syntax and structure of TI encoded as

[15] In IPFS, a content identifier (CID) is based on the cryptographic hash of the file's content (content-addressable storage). Per default, SHA256 is used, see https://docs.ipfs.tech/concepts/content-addressing, accessed on 2023-09-29

jsonpath-object-transform template. We give a full example in Appendix D. Alternate approaches to transform JSON between different schemata or to another representation (such as XML) include table-based transformations, more generic templating systems using JSONPath [Gös07], and systems based on XML's XSLT (including its experimental variant JsonT [Gös06]).

```
{
  "TransformationInformation": {
    "GraduationDiploma": {
      "PersonDID": "$.credentialSubject.id",
      "UniversityDID": "$.credentialSubject.alumniOf.id",
      "UniversityName": "$.credentialSubject.alumniOf.name.value"
    }
  },
  "TransformationInformationSignature": {
    "type": "RsaSignature2018",
    "created": "2019-01-01T01:00:00Z",
    "proofPurpose": "assertionMethod",
    "verificationMethod": "did:example:ababb1f712ebc6f1c2762ec1",
    "jws": "TVkIEq_PbChOMqsLfRoPsnsgw5WEuts01mq..."
  }
}
```

Listing 8.2.: Example TI, encoded for the *jsonpath-object-transform* transformation engine. This TI transforms a VC into a simple diploma credential. The SP uses the signature to authenticate the TI. Trust in the publisher is established by checking its DID (stored in *verificationMethod*) on the WoT graph.

Evaluation of Prototype: The performance of our approach consists of the retrieval of TI (see Chapter 6) and the execution of the transformation. The performance of the transformation itself depends on the respective transformation engine. In our prototype, we use *jsonpath-object-transform* [Lan+17] as transformation engine. To evaluate the performance of this core component, we run several benchmarks. To do so, we use the example transformation template from Appendix D. The execution of the full transformation process takes 38.8 ms (± 0.6 ms). Since jsonpath-object-transform is a *nodejs* module, this benchmark involves the initialization of the nodejs runtime on each request (which takes 36.5 ms ± 0.8 ms).

To take this into account, we also evaluate the performance of jsonpath-object-transform directly in nodejs (i.e., simulating a more optimized architecture). This results in a performance of 0.46 ms (± 0.05 ms).

8.6. Evaluation & Discussion

In this section, we evaluate our approach against the overall goals of this thesis and the requirements established above. We then discuss several aspects of our trusted credential transformation system.

8.6.1. Evaluation

This chapter focused on facilitating global trust scheme interoperability and consider heterogeneous attestation formats; this is Goal 2 of our overall research goals (see Chapter 5). In that scenario, a SP needs to understand the semantics of attestations issued in schemas that it does not understand. Hence, it needs to transform the attestation to a different format.

Based on that goal, in Section 8.2 we formulate seven requirements for a credential transformation system (R1–R7). We now discuss the evaluation our system against the tackled requirements. For each requirement, we briefly state the measure or design decision that ensures we comply.

R1 Attestation Format Interoperability: We fulfill this meta-requirement by providing a trustworthy credential transformation system (see Section 8.4). By building on existing local credential transformation approaches (e.g., XSLT), we take advantage of established transformation methods while enriching them with trust and flexibility, expanding the state of the art.

R2 Trust: We fulfill this requirement in two ways: When performing a schema transformation using only TI, the SP already has the knowledge to check a transaction's authenticity. Hence, our system directly verifies the authenticity of an incoming credential before transforming it (see Section 8.4). In case of a format transformation, VI is used in addition, which contains the logic to authenticate the credential (see Section 8.4 and Section 8.6.2). In both cases, the SP can use their own trust rules (i.e., trust policy) to decide if they consider a credential's issuer trustworthy.

Further, all transformations are executed only using trustworthy information. In specific, we use the trust scheme already in use by an SP to establish trust in TI and VI before executing it.

R3 Local Trust View: We comply with this requirement since all transformations are performed locally, i.e., without the issuer's involvement. Hence, we don't require that the issuer create a credential in the correct format or provide it at the verifier's request. This enables a verifier to use their local trust perception (in the form of a trust policy) to authenticate the credential and their own access policy to check the credential's attributes.

R4 No modifications of the issuer: Since our system does not involve the credential issuer in a transformation, we comply with this requirement. Specifically, our system supports established signature schemes and credential formats. Thus, there is no need to adopt novel schemes.

R5 Extensibility: Our system uses the trust schemes trusted by the SP to discover and retrieve the TI and VI needed to transform a credential. The system is fully extensible because there is no need to pre-configure any transformation mapping. Further, retrieving and installing new mappings (i.e., TI and VI) happens automatically and on demand. A limitation to this is that only mappings published by trusted entities are available, so there is no guarantee that a specific credential can be processed.

R6 Confidentiality & Undetectability: We comply with these privacy requirements since neither the credential nor any information referencing the holder is transmitted to a third party. This is achieved by retrieving TI/VI and executing it locally—without the involvement of any TTP or the issuer.

R7 No execution of third-party code: We partially fulfill this requirement by differentiating schema and format transformation (see Section 8.4). In the case of a schema transformation, TI is used to transform the credential. Depending on the utilized transformation engine, TI is encoded as a machine-readable mapping template, i.e., not Turing-complete. While

there is no need for TI to be Turing-complete, there are nevertheless Turing-complete mapping templates, e.g., XSLT.

In the case of a format transformation, the used VI also contains executable code to authenticate the credential, which leads to noncompliance with this requirement. For use cases where VI is needed, we recommend limiting the list of trusted VI publishers. Further, we suggest applying sandboxing techniques on the transformation engine as a minimum line of defense.

8.6.2. Local Trust

A challenge that we tackle in this chapter is the authenticity of transformed credentials. To avoid a signature becoming invalid due to transforming the credential, the SP first verifies the signature of the original credential and only then converts the credential (based on trusted TI) locally. Signature verification and credential transformation occur locally in the SP's trust domain; only trusted transformation information is used. Assuming the published TI is correct, this guarantees that the transformed credential has the same meaning as the incoming signed one.

8.6.3. Limitations

Transformations of credentials have limitations. For example, if a target schema requires values that are not available in the source schema, a direct transformation might not be possible. In that case, the TI could map missing values to some defaults. The SP or its policy then must decide whether or not the transformed credential is considered sufficient. Given these missing values, the transformation result could be deemed insufficient for the assessment.

8.6.4. Future Work: Transformation by Holder

In this chapter, we discuss credential transformations that are performed by the SP. In that approach, a credential's holder does not know about the target schema/format the SP needs, and sends the credential in its original encoding. To establish trust in the credential, the SP checks the signature on the credential before the (schema) transformation, or it uses VI to verify the credential's signature.

An alternative approach is credential transformation that is performed directly by the credential's holder. In that case, the SP tells the holder

about its schema requirements, and the holder transforms the credential before sending it to the SP. But, this naive transformation by the holder breaks the signature on the credential, and thus the SP cannot trust it [You23]. To mitigate this, we propose an approach where the holder creates a cryptographic proof showing that the new credential was generated using a credential with a valid signature. This approach can be implemented using our zero-knowledge policy system, which we introduce in Chapter 9. See Figure 8.5 for a comparison of those approaches.

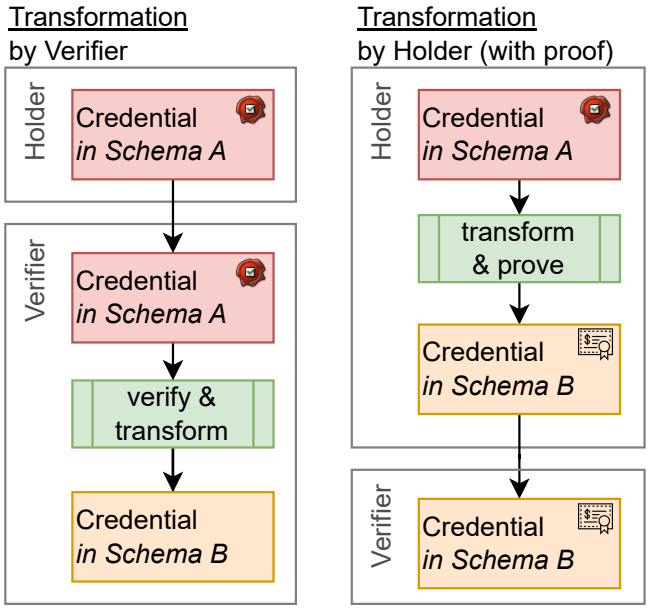

Figure 8.5.: Different approaches to Credential Transformations, differentiated by who is performing the transformation. In a transformation performed by the holder, the holder transforms the credential and proves the relation using a cryptographic proof.

Chapter 8 Conclusions

In this chapter, we presented our approach for flexible and trustworthy credential transformations. Using trusted credential transformations, issuers and SPs can exchange digital credentials without requiring them to agree on a format and schema. Further, we discussed a proof of concept implementation of our transformation approach, demonstrating its feasibility. The research results presented in this chapter have been published as part of an academic paper [Mor+21].

Privacy in Identity- and Trust-Management

Research Area 2 Introduction

Outlook

In this part of the thesis, we present our research in the area of **Privacy**. Our work in this area resulted in two scientific publications (see Section 1.2.1). The presentation of this research is structured into the following two chapters:

In Chapter 9, we present a generic design for supporting **privacy-preserving technologies in access control systems**. In this approach, we mitigate the problem that users commonly need to reveal too much of their sensitive data when presenting credentials to the Service Provider (SP). Our design prevents unnecessary disclosure of sensitive information while still allowing the formulation of expressive rules for access control. For that, we make use of Non-Interactive Zero-Knowledge Proofs (NIZKs). The fundamental idea is to automatically derive the structure of the proof a user needs to create from an access control policy and also integrate the verification of those proofs into the policy system. Our approach allows users to only provide the information the service requires. At the same time, it ensures that SPs can use trustworthy attributes to verify the user's request.

As a reference implementation of this concept, we provide a privacy extension for the TPL system, which we introduced in Chapter 7 above. We also evaluate the resulting *ZK*-TPL language and its associated toolchain. Our evaluation shows that for usual credential sizes, communication and verification overhead is negligible.

In Chapter 10, we present our **Ledger State Attestation** system. Users can utilize this system to retrieve offline-verifiable attestations on data stored on a Distributed Ledger (DL). For example, this data might be the revocation status of some credential or data from other DL-based registries. This approach then enables users to present this data to SPs that are offline. Doing so has the advantage that the DL and its node do not learn about the relationship between SP and the user. This prevents the detectability of the credential presentation and preserves the user's privacy (cf. Section 3.2). Further, it increases the availability of the overall system. We also provide a reference implementation for the Ethereum ledger and discuss our approach.

9

Enhancing Access Policies with Privacy Features

This chapter is based on the paper *Extending Expressive Access Policies with Privacy Features* by More, Ramacher et al. [Mor+22]. Parts of this paper have been copied verbatim. The prototype was implemented by our student Marco Herzl.

In Chapter 7 above, we introduced our extensible and flexible policy system *TPL*. Policy systems like TPL enable the codification and re-use of access policies while decoupling them from the deployed access control systems. Furthermore, policy systems offer a higher level of abstraction that facilitates the design of policies without requiring concrete insights into the implementation of the underlying access control system.

In the *policy-based and credential-based access control* model, a policy is defined by the Service Provider's (SP) domain expert, codifying rules for authentication, authorization, and trust verification [Lee11; SWS07, Section C-3; Mor+22]. Users in the system receive digital credentials and store them in their digital identity wallet. To authenticate, a user later bundles the required credentials to their service request and sends it to the SP (as visualized in Figure 9.1). After receiving the request, the SP uses a *policy interpreter* to verify the incoming user request by applying its access policy.

However, this approach suffers from **privacy issues**. Users are often in possession of credentials that certify numerous attributes (see also Section 8.1). When showing a credential to an SP, users reveal all attributes to the verifier, which is often neither desirable nor necessary to fulfill an authentication request. For example, to pass the TPL policy given in Listing 9.1, a user needs to reveal their date of birth. This is more

information than needed, since the SP is only interested in the predicate whether the user is older than 17.

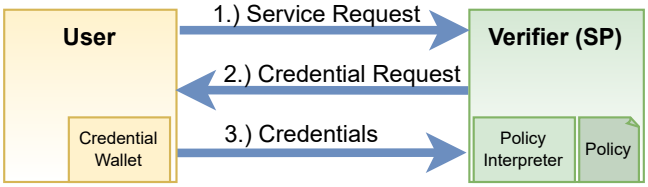

Figure 9.1.: High-level architecture of an access process using a policy system.

By integrating privacy-preserving technologies into the access control process, users are enabled to only reveal a subset of the attributes or even prove a statement without revealing any attribute at all. For example, by introducing Camenisch-Lysyanskaya (CL) signatures [CL02] into W3C's verifiable presentations [SLC22], support for privacy-preserving showings is achieved. Those features are well-understood in the field of attributed-based credentials [Cha81; Cha85] and have been studied for Self-Sovereign Identity (SSI) systems [Müh+18; Abr+19; Abr+21]. While the above-mentioned research and standards are helpful, integrating privacy-preserving technologies into policy-based access control systems is not straightforward.

Several challenges need to be addressed:

- Policy systems offer a method to define access policies and distribute them among all participants of a system, but they currently lack the possibilities to represent (expressive) statements on hidden attributes—both on the language side and the interpreter side. How can privacy-preserving technologies help to overcome these challenges while being flexible enough to preserve the expressiveness of the policy languages?
- With the possibility to have hidden attributes, integrators and policy authors face a new challenge: which attributes should be hidden or revealed to the SP? In addition, which statements/predicates on the hidden attributes need to be revealed?
- Even if these questions are solved on the side of the SP, this information needs to be communicated to users. How is the user informed

on the statements they need to present?

- On the user side, identity wallet implementations need to support a multitude of different services all having their own access requirements. How can a user-side wallet implementation support all the possible proof types different SPs may ask for?

We close the gap by extending policy language systems with privacy features using (non-interactive) zero-knowledge proofs (NIZK-Proofs). We introduce a generalized approach for extending existing policy language-based access control systems. In our approach, the policy's author codifies which statements the user should prove and which information needs to be revealed to receive access as a *policy*. The policy author then uses our *policy compiler* to derive a presentation request they provide to users. The presentation request informs the user about the attributes they need to reveal and statements they need to prove. That enables users to only provide the required attributes and statements, and hide the rest of the credential data. While we focus our implementation and evaluation on the SSI model, the design itself is generic. It can be applied to various policy systems and identity management models, enriching them with privacy features.

```
accept(Cred) :-
 extract(Cred, format, w3c_verifiableCredential),
 extract(Cred, issuerDID, DIDissuer),
 checkQualified(DIDissuer),
 verify_signature(Cred, DIDissuer),

 extract(Cred, credentialSubject, Subject),
 extract(Subject, date_of_birth, Birthdate),
 calculateAge(Birthdate, Age), Age >= 18,

 extract(Subject, username, Username),
 print(Username).
```

Listing 9.1.: Example TPL policy for W3C Verifiable Credentials, with the trust-check omitted for clarity.

Chapter 9 Goals:

We discuss the overall goals of this thesis in Chapter 5. In this chapter, we focus on Goal 4 (Privacy). In specific, we tackle the aspects of confidentiality as well as integrating privacy technology into access control systems.

Chapter 9 Outline:

We start the chapter by defining the requirements for privacy-preserving access control systems in the scope of our thesis (Section 9.1). Further, we give an overview of the state of the art in privacy-preserving access control (Section 9.2).

We then introduce our generic approach of privacy extensions for access control systems (Section 9.3). In this section, we also introduce relevant concepts and describe the process of privacy-preserving access control using our approach.

In Section 9.4, we show the practicability of our approach by applying it to the TPL policy system. The result is an expressive policy system with privacy features that re-uses existing policies. We evaluate our reference implementation in Section 9.4.3.

In Section 9.5, we evaluate our generic concept against the stated requirements. Further, we discuss our approach and state ideas for future work.

9.1. Requirements

In this section, we outline requirements for our privacy-preserving access control system. We derive requirements from the overall privacy goal considered in this chapter (Goal 4). Additionally, we consider the other goals of this thesis, in specific modularity and extensibility (Goals 1 and 3, cf. TPL requirements in Section 7.1). To also respect general requirements on a policy system, we add the relevant criteria from two review papers by Seamons et al. and Coi et al. (cf. Section 7.1.2) [Sea+02; CO08]. In the following Section 9.2, we then use those requirements to survey the state of the art. Later in Section 9.5.1, we also use the requirements to evaluate our own approach.

R1 Data Minimization: Goal 4 of our thesis is to preserve the users' privacy. This goal includes the potentially sensitive content of credentials and information about the user's behavior. This represents a challenge if the verifier needs access to some but not all of the data stored in the credential. To fulfill this goal, the system must support the selective disclosure of attributes without invalidating the authenticity guarantees of the credential.

R2 Expressive Power: We aim to provide a policy system flexible enough to represent all kinds of expressive policies (Goals 1 and 3). To provide a policy designer with the greatest flexibility possible, we focus on Turing-complete policy languages. To enable expressive policies and, at the same time, preserve the user's privacy (Goal 4), the system must support the computation of predicates on private attributes, revealing only the result of the predicate.

R3 Unlinkability: Sometimes a user selectively reveals only some attributes to an SP but later reveals other attributes of the same credential to the same SP. If those presentations a linkable, a malicious SP could combine those presentations, violating the user's privacy. The same issue exists if two or more SPs collude to aggregate credential presentations of the same user. To protect the user's privacy in that case, the system must provide *unlinkability* (see Section 3.2). Specifically, the system should provide unlinkability with respect to two different SPs, and for multiple showings at the same SP (*multi-show unlinkability*).

R4 Simple linking of predicates: Constraints on attribute values, both public and private, are supported (cf. R1 and R2). The system must enable the chaining of multiple constraints, i.e., allow for logical conjunction of constraints.

R5 Expressive linking of predicates: The relevance of some constraints depends on the value of other constraints or attributes, forming a generic boolean equation. Thus, in addition to a simple logical conjunction of constraints (R4), the system should also support expressive chaining of constraints, i.e., also support disjunction.

R6 Inter-credential constraints: The system must support constraints in the attributes of two different credentials. E.g., by requiring that a specific attribute in two credentials has the same value, those two credentials could be linked. Additionally, the value of an attribute in a credential can influence the constraints on attributes in another credential.

R7 No modifications of the issuer: The goal of our approach is to facilitate interoperability (Goal 2) between existing systems as well. Hence, to increase the system's acceptance, modifications of the issuer's systems must be avoided. Specifically, the system must not assume that the issuer adopts a novel signature scheme.

R8 Practicability: To ensure the practicability of our approach, we consider several aspects. First, the system must minimize the setup effort imposed on the user. Second, the system must provide realistic performance, i.e., consider common usability goals. Following the advise from Nielsen, "1 second is about the limit for the user's flow of thought to stay uninterrupted", and "10 seconds is about the limit for keeping the user's attention focused" [Nie97, p. 135]. This limits concern the time it takes to calculate a credential presentation and the duration the user has to wait for a result from the verifier. Further, the credential presentation must be transmitted from the user to the SP, e.g., over the Internet or Bluetooth. As this impacts the performance, the system should also limit the size of transmitted data, i.e., of the credential presentation.

9.2. State of the Art

We now survey the state of the art of privacy-preserving access control systems (ppACS) with a focus on the policy-based and credential-based access control model. In that model, a ppACS comprises a policy-based access control system and privacy-preserving technologies.

The ABC4Trust project focused on privacy-enhancing attribute-based credentials (ABC) and can be seen as preliminary work for the W3C VC standard [SLC22] and modern SSI [SKR12; Bic+15a; Bic+15b]. In their approach, the verifier sends a so-called presentation policy to the user. This policy states the conditions the user has to fulfill to access the service. On the user side, the ABC engine then matches the needed attributes, and finally, a presentation token is created consisting of cryptographic

evidence that the user satisfies the policy. This proof can later be verified for access control purposes. The calculation of tokens relies on credentials signed using specific cryptography (e.g., CL signatures [CH02; CL01] or Brands blind signatures [Bra93; Bra95]; cf. [Cam+14]). This requires modifications of the issuer (violating R7). Further, since the whole policy must be proven through the presentation token, it can not contain any conditions that are difficult or impossible to translate to a cryptographic proof (e.g., online lookup for trust verification). Additionally, the project supports a limited list of functions for use in predicates on private attributes.[1] As ABC4Trust policies build on XACML, they only support the conjunction of predicates and no expressive linking of constraints (violating R5). Thus, ABC4Trust enables interesting privacy features, but its policies are limited in flexibility and expressiveness (impacting R2).

Belchior et al. propose a Self-Sovereign Identity based access control (SSIBAC) for service providers [Bel+20]. It leverages conventional attribute-based access control using the attributes in SSI credentials, profiting from its decentralized nature. SSIBAC uses the XACML standard for policy specification (impacting R5). Their implementation is based on W3C Verifiable Credentials [SLC22], Hyperledger Indy[2], and Aries[3] for communication and distributed storage. While they mention the possibility of introducing privacy-enhancing technologies, they do not discuss its integration into an access policy language.

Conclusion: Above, we presented a selection of privacy-preserving access control systems. While the state of the art shows that there is ongoing research in this field, none of the presented systems is suitable for the goals of this thesis. In specific, existing systems use credentials based on cryptographic techniques that prevent the critical expressiveness requirement R2. For the same reason, they only allow simple linking of predicates and attributes (enabling R4 but violating R5) and require modifications to the issuer (violating R7). However, the conceptual technique applied by ABC4Trust inspires a valuable building block, namely the derivation of presentation request and token from an access control policy. Hence, we advance this approach and investigate it with novel and more expressive cryptographic techniques in the rest of this chapter.

[1] https://abc4trust.eu/download/Deliverable_D2.2.pdf, Section 4.4.3
[2] https://github.com/hyperledger/indy-sdk, accessed on 2022-07-03
[3] https://github.com/hyperledger/aries, accessed on 2022-07-03

A second aspect of the thesis Goal 4 is the integration of privacy techniques into *existing* access control and policy systems (see Section 5.4.2). Additionally, we introduced the TPL system to provide a foundation for trust management in a heterogeneous setting, as discussed in Section 7.2. This research resulted in a policy system that fulfills our thesis' goals but lacks privacy features. Thus, in this chapter, we also demonstrate our concept's feasibility by applying it to the TPL system.

9.3. Compiling Access Policies into NIZK-Proofs

In this section, we describe the design of our access policy system with privacy-preserving features. In Section 9.4, we discuss a concrete instantiation of this design.

9.3.1. Concept

Before describing the different components, actors and the process, we will present the high-level idea of our concept.

Preliminary: Commit-sign-proof Credentials One common approach (cf. [CL02; FHS19]) to design attribute credentials is to first commit to the attributes. This commitment is then signed by the issuer (i.e., identity provider). We show the simple form of this process in Figure 9.2. For privacy-preserving showings, the user later proves consistency of any revealed attribute with respect to the commitment. The latter proof is combined with a proof of knowledge of a signature on the commitment, or by directly providing the signature to the verifier.

Compiling Access Policies into NIZK Proofs As in other systems with access control, rules that have to be satisfied must be represented as program logic, forming a policy. Hence, we extend the concept of commit-sign-proof credentials with the possibility for a policy designer to codify such access rules. For any committed-to attribute, we enable the policy designer to decide whether the user needs to reveal an attribute to the service provider or whether it is sufficient to convince the verifier that an attribute satisfies a policy rule without revealing it. Our system automatically informs the user about the policy and compiles it into the corresponding Non-Interactive Zero-Knowledge Proof (NIZK). That is,

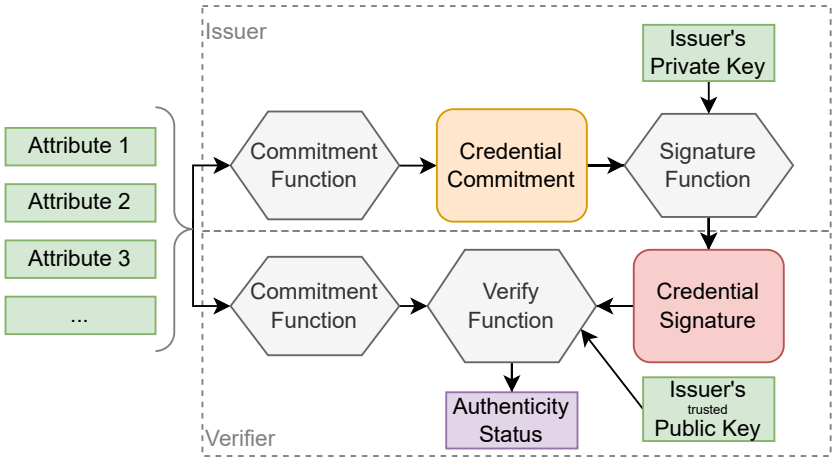

Figure 9.2.: Data flow of a commit-sign-proof credential.

the user proves the consistency of revealed attributes with the public commitment. Similarly, for all rules involving hidden attributes, the user also generates proofs of knowledge of these attributes and that they satisfy the specified rules.

In our system, we allow the policy to express rules with respect to any credential format: attributes that are encoded in some form of data structure that has some public reference value. The latter may consist of a signed commitment or a signature directly over the attributes. The statement for the proof system is then built accordingly.

9.3.2. Components

Our system consists of the following actors/components:

User (Holder): The actor that wants to access a resource or consume a service at the SP, and needs to authenticate to do so. Their identity attributes are stored in form of **digital credentials** in a **digital identity wallet** [PAZ22]. Part of the wallet is also a **policy client**, which prepares a **presentation token** that satisfies a **presentation request**.

Service Provider (Verifier): The SP is the actor (or their system) which provides access-protected services or resources to users. To control who

can access a resource and how users are authenticated, the administrator of the SP creates a **policy** that encodes access control and trust rules. The SP uses a **policy compiler** to generate the presentation request, which they provide to the user. After receiving a service request (which includes the presentation token), the SP uses a **(extended) policy interpreter** to verify the request.

9.3.3. Process

We now discuss the entire process of our concept. We split the process into the following three phases: 1. The setup of the cryptographic system, authoring of a policy, and publishing of the presentation request, 2. computing of a presentation token, and finally 3. the verification of the presentation token.

Phase 1 **Setup System and Policy:**

Setup cryptography: If the employed proof system requires a specific setup, performing it is the first step. For example, when using Succinct Non-Interactive Arguments of Knowledge (SNARKs), a trusted third party generates a Common Reference String (CRS) (we discuss this in Section 9.5.2 below). The so-obtained common material is published and retrieved by all system participants.

Author policy: During the setup phase, a *policy* is authored by the SP. This policy encodes the rules a user of the SP needs to fulfill to use the service or access a resource. Depending on the nature of an SP, there can be different policies for different services or resources.

Authors of this policy are either technical personnel or domain experts of the SP. An author without technical knowledge but with domain expertise can use a graphical policy authoring tool to create the policy (cf. Section 7.5.3). A policy is, in the end, always encoded in machine-readable form, e.g., as an executable program or in a specific data format.

Define public and private attributes: As part of the policy, the author specifies which attributes a user has to provide and what credential types the SP accepts [OM23]. Additionally, the policy author defines two types

of rules the user attributes have to satisfy, which are differentiated by whether they operate on revealed attributes or private attributes. The first set of rules operates on attributes the user must reveal to the SP. They are used when the SP requires the attributes for further processing or trust management. The second set of rules are instead transformed into statements for the NIZK proof system. Thus, the user can prove that their credentials fulfill these rules without revealing the values of the attributes.

Compile policy and publish presentation request: Depending on the NIZK proof system, the SP at this stage also compiles parts of the policy into an intermediate representation for the policy client.

The encoded policy, together with metadata about the service, forms the *presentation request* for a specific service. Before initiating a service request, users need to know what data they are required to provide. Therefore, the presentation request is published by the SP alongside the service description.

Phase 2 **Authentication at SP:**

Retrieve presentation request: When a user want to access a service or consume a resource, they have to authenticate with the SP. To do so, they first retrieve the corresponding presentation request from the SP's website or another form of a service catalog.

Execute policy: The user then extracts the policy from the presentation request and executes it using the policy client. While doing so, the client retrieves the respective credentials from the user's identity wallet [Cam+15].

Handle public attributes: For the rules on public attributes, the client extracts the attributes from the credential, thereby revealing their values. It then computes a NIZK statement that proves that the value was indeed extracted from the credential, i.e., to prove the consistency of the values with the commitment. This statement proves that the revealed attributes are linked to their credential. As a special case, if all attributes of a

credential are specified as revealed in a policy, the client adds the full credential to the response instead of a proof.

Handle private attributes: For the policy rules on private attributes, the client computes a NIZK statement which proves that the attributes fulfill the rules. Again, the client adds a statement to the proof that the attributes were indeed extracted from the credential.

For the computation of proofs, the client uses the common material retrieved in the setup phase and the NIZK statements. The client also appends the metadata (e.g., issuer information) of all involved credentials to the response.

Create presentation token: After executing the policy, the client encodes all proofs, revealed credentials, and credential metadata into a *presentation token*. An example presentation flow is shown in in Figure 9.3.

Then, the user adds service-specific data and sends it alongside the token to the SP.

Phase 3 **Verification of Presentation Tokens:**

Load policy: On receiving a request, the SP loads the policy for the respective service. The SP's policy interpreter then uses a *NIZK verifier* and a *policy verifier* to check the presentation token.

Execute NIZK verifier: The SP extracts the NIZK proofs from the presentation token and verifies them using the NIZK verifier. As inputs for the NIZK verifier, the policy and its proof system-specific representation, respectively, need to be provided. Additionally, all public reference values, i.e., the commitments and all revealed attributes, need to be known to the NIZK verifier. Hence, the SP extracts these values from the presentation token and provides them to the NIZK verifier. After this step, the verifier is convinced that the proven statements match the requested statements.

Execute policy verifier: As next step, the SP initializes the policy verifier and executes the remaining rules of the policy. All rules that work only on revealed or public data are validated by the policy verifier.

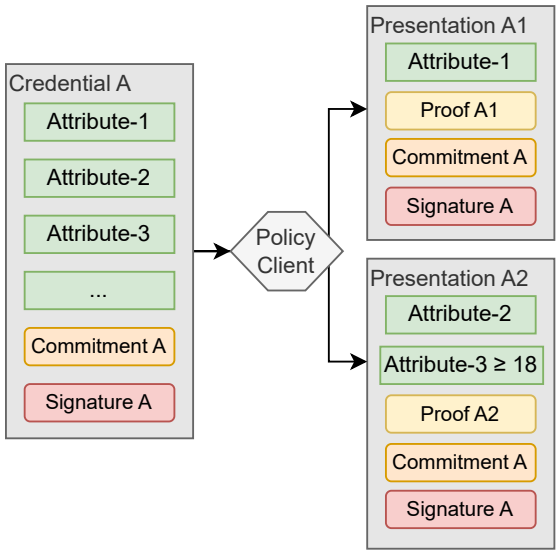

Figure 9.3.: Deriving different presentations from the same credential. `Presentation A1` only reveals `Attribute-1`, while `Presentation A2` reveals `Attribute-2` and a predicate on `Attriibute-3`, but not its value.

Ensure trustworthiness of tokens: Besides evaluating the rules on the revealed attributes, this phase verifies the token's trustworthiness. To do so, the policy verifier uses the trust rules encoded in the policy to check the issuer of the credential (i.e., the signer of the attribute commitment). That forms a trust chain from the issuer along the signature to the commitment, which is in turn linked with the proof and consequently the attributes. This trust chain is also visualized in Figure 9.4.

The trust rules specify which issuers are trustworthy for which type of credentials. That can, for instance, be done by providing a list of trusted issuers. A more flexible method is to define a trusted trust scheme (cf. Section 7.4.2): One example of a possible trust scheme is Europe's eIDAS trust framework. Another example are SSI trust schemes established using distributed ledgers (cf. Section 7.4.4). Depending on the defined trust scheme, the policy verifier automatically retrieves trust status information about the credential issuers from online registries. This process ensures that public and private attributes can be trusted. Therefore, all NIZK statements on these attributes are trustworthy w.r.t. the user's rules.

Conclude verification: After the NIZK verifier and the policy verifier conclude that the presentation token is trustworthy and fulfills the user's policy, the SP grants the user access to the service.

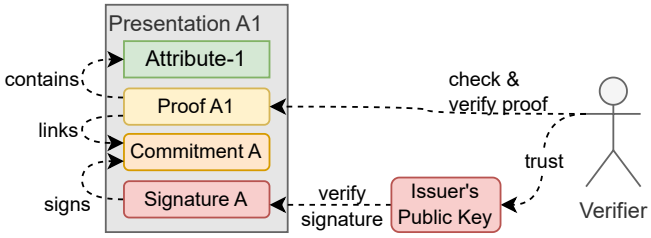

Figure 9.4.: The tust chain of our approach. After establishing trust in the issuer's public key (e.g., by using our approach from Chapter 6) and verifying the signature, the verifier establishes trust in the proof by comparing the signed commitment with the commitment in the proof. This authenticates the proof, which is then used to authenticate the (revealed) attribute.

9.3.4. Benefits

In our approach, we derive the presentation request from an access control policy. We then send it to the user, who automatically generates a suitable presentation token using their identity wallet and existing credentials. Our design properties result in several benefits for the involved actors.

Benefits for users: Our approach builds on commit-sign-proof credentials and standard cryptographic schemes. By doing so, we allow the use of existing credentials in a privacy-preserving way, without the need to change credential specifications or modify issuers. Specifically, only relevant attributes need to be revealed, and different attributes an be revealed for different SPs or policies (see also Figure 9.3). Additionally, user can use credentials they already received in their identity wallet, enhancing their privacy. Those processes are performed automatically, as the presentation request is executed by the policy client.

Benefits for SPs: In our approach, we derive privacy-preserving access requests from high-level access control policies. Our design further serves

as a basis for the use of cryptographic techniques that allow the use of more expressive policies (e.g., SNARKs), as discussed in the next section. This enables service providers and their policy authors to express complex business rules using existing tools and languages they know. By always linking the derived presentation request with the signature on the original credential, we retain the authenticity guarantees of the attributes. The SP can then use their own trust policy to establish trust in this signature, retaining a local trust view.

9.4. Implementation

While our concept is described on a generic level, the concrete choice of policy system and proof system is important to assess the feasibility and to evaluate the performance and security. Thus, we provide a concrete instantiation of our concept.

Our implementation builds on our TPL policy system (as introduced in Chapter 7), which we extend with privacy features. Since TPL uses a logic-based syntax, we need to compile the rules encoded in form of TPL predicates to suitable statements for NIZK proofs.

As NIZK proof system we use SNARKs, since it enables small proofs (see also Chapter 4). We instantiate our SNARKs with the Groth16 proving scheme, which we execute with the help of the Bellman library.

Furthermore, we integrate the ZoKrates zero-knowledge toolbox [ET18] as an intermediate layer in the transformation process. Thus, we compile TPL policies into ZoKrates programs, which are then mapped to circuits for bellman. An advantage of this intermediate step is that the SP can already compile the policy into a ZoKrates program and directly share that with the user as part of the presentation request. As an alternative, the presentation request can be directly bundled with the identity wallet or shipped to the user by other means prior to authenticating. This makes sense for large proof requests which could take too long to transmit to the user directly (e.g., via bluetooth). For example, when applying our SNARK-based system on credentials that use SHA256 as commitment function, proof request size can be on the scale of 100MB.

During authentication, the user only needs to provide their credentials to the ZoKrates client and execute the ZoKrates program. Then, they send the resulting proof alongside the selected revealed credentials to the

SP as part of the presentation token. Finally, the SP uses the ZoKrates verifier to ensure the proof is valid. In the next step, they forward the NIZK verification result and the rest of the presentation token to the TPL verifier. In addition to executing the policy on the revealed credentials, the TPL verifier also checks whether all data is trustworthy. An overview of this process is shown in Figure 9.5.

In our implementation we enable the privacy-preserving showing of attributes originating from credentials as well as private statements on these attributes driven by TPL policies. We did not focus this implementation on other aspects of the credentials, in specific we don't hide the issuer or other parts of the trust chain. In the following sections, we discuss the integration of our concept (from Section 9.3) into the TPL system in more detail. Specifically, Section 9.4.1 presents the extensions of TPL to ZK-TPL from the point of view of the policy author. And Section 9.4.2 covers aspects of compiling ZK-TPL into NIZK statements using ZoKrates. Finally, Section 9.4.3 presents the evaluation of our implementation.

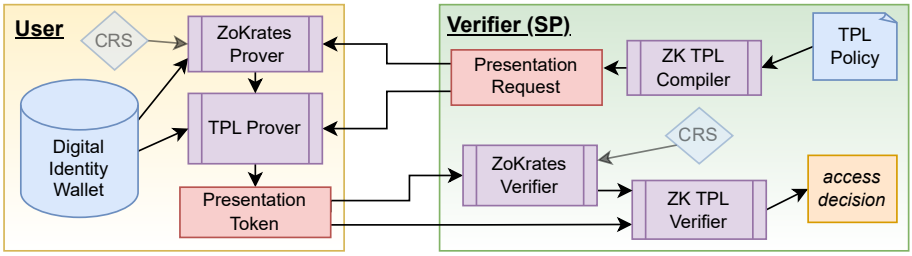

Figure 9.5.: Architectural overview of our implementation including the actors and process flow.

9.4.1. Extending TPL with Zero-Knowledge Rules

We now focus on the concrete changes to the TPL syntax to express ZK rules. A policy author defines in a policy which attributes a user needs to reveal. There are multiple options to denote this in a TPL policy:

Option 1: Naming Convention: In TPL, the type of terms such as atoms and variables is defined by their name. Any term starting with an uppercase letter followed by letters, numbers or underscores represents

a variable. Whereas terms starting with lowercase letters refer to atoms (constants). Hence, in the same way we could refer to attributes that are not revealed via a naming convention. However, as adding new conventions to the TPL specification would lead to ambiguities, we consider this approach to be error-prone and unintuitive.

Option 2: Privacy Predicate: Another approach is to represent the hidden and revealed nature of attributes explicitly via a special predicate. As with other domain-specific predicates that are available for TPL, a new predicate can be defined that only evaluates to true if the corresponding attribute is hidden. With this approach, all predicates related to this attribute need then to cope with a potentially hidden attribute value.

Option 3: `zkaccept`-Predicate: Finally, the third (and chosen) option is to add a new `zkaccept` predicate in addition to the `accept` entrypoint-predicate for TPL programs. A policy is satisfied if and only if both `accept` and `zkaccept` evaluate to true. Consequently, `accept` and `zkaccept` are the two main predicates that represent a TPL rule set. With this approach, the meaning of the `accept` predicate is untouched and interpreted as before. In the `zkaccept` predicates, all statements are interpreted with respect to hidden attributes and cause the creation and verification of the corresponding NIZK proofs. Our approach is exemplified by the TPL policy in Listing 9.2. It requires the owner of the credential to be of 18 years or older, whereas neither the calculated age nor the `date_of_birth` attribute are revealed to the verifier. The example policy also contains a `semester` attribute, which is revealed to the verifier since it is only used in the `accept` predicate, but not in the `zkaccept` predicate.

We opted to implement the third approach since it provides a clear differentiation between predicates applied to public and hidden data points. As such, we consider it easier for the policy designer to design and reason about the policy. From a technical perspective, we expect all three approaches to be implementable with reasonable effort.

Consistency Checks: When evaluating the policy on the prover or verifier side, one needs to take care of multiple issues. First, an attribute can only appear as either hidden or revealed attribute. If a revealed attribute is also used in the `zkaccept` predicate, this is likely a mistake of the policy designer. Thus, the compiler needs to check if this invariant is satisfied

```
zkaccept(Presentation) :-
    extract(Presentation, format, w3c_VP),
    extract(Presentation, verifiableCredential, Cred),
    extract(Cred, format,  w3c_VC),
    extract(Cred, credentialSubject, Subject),

    % range proof on private attribute
    extract(Subject, date_of_birth, Birthdate),
    calculateAge(Birthdate, Age), Age >= 18.

accept(Presentation) :-
    extract(Presentation, format, w3c_VP),
    extract(Presentation, verifiableCredential, Cred),
    extract(Cred, format,  w3c_VC),
    extract(Cred, issuerDID, DIDissuer),

    checkQualified(DIDissuer),
    verify_signature(Cred, DIDissuer),

    extract(Cred, credentialSubject, Subject),

    % revealed attribute
    extract(Subject, semester, Semester),
    print(Semester).
```

Listing 9.2.: TPL policy from Listing 9.1 extended with privacy-preserving features: from the full credential, only the statement about age (derived from `Birthdate`) and the `Semester` attribute are revealed.

and yield an alert if not. Secondly, when two or more distinct attributes of the same data structure, e.g., a credential, are used in `accept` and `zkaccept`, consistency needs to be ensured. Hence, the zero-knowledge proof needs to be extended with statements ensuring consistency of all publicly revealed attributes.

9.4.2. Compiling TPL policies to NIZK circuits

ZK-TPL Compiler: As ZoKrates provides a domain-specific yet high-level language that closely resembles the syntax of Python, the TPL rules need to be compiled to this language. ZoKrates itself then compiles the corresponding code to suitable circuits for the underlying NIZK library, i.e., bellman. Hence, we provide the ZK-TPL compiler to transform policies from TPL syntax into ZoKrates' proof program syntax, as shown in Figure 9.5. The ZoKrates standard library already provides several cryptographic primitives such as the compression function of SHA256, and SHA256 for a fixed number of calls to the compression functions, i.e., SHA256 for fixed input lengths, or Pedersen commitments. Therefore, parts of the functionality we require are covered by the standard library. Comparison operators for primitive types are also provided by ZoKrates.

When compiling TPL policies to ZoKrates programs, we consider the following challenges:

1) Constant-length Attributes: When generating the ZoKrates proof program, a challenge is to map attributes to either private or public variables, and how to encode their lengths. As lengths can already be sensitive information for various data points, they are encoded as fixed-length string with `0` to pad to the maximal length. Thus, there is a compromise between runtime costs for the additional padding, security, and functionality. Length restrictions may be problematic for field types with international variations such as the use of first and last names.

2) Arithmetic: While integer types are available in various forms (8 bit to 64 bit), ZoKrates also provides a native `field` type representing $\mathbb{Z}_p$ (the integers modulo p) where the prime p depends on the choice of elliptic curve used by ZoKrates. In general, p will be large (≥ 256 bit) and all the arithmetic of the smaller types is implemented as $\mathbb{Z}_p$ arithmetic. Hence, when compiling arithmetic involving hidden attributes, arithmetic

is best represented using the `field` type unless specific features of the fixed bit-width types are needed.

3) Representing Strings as Numbers: Third, parsing arbitrary strings as integers is a complex and expensive task when performed inside ZoKrates. Conversion of an array of `u8` into a `field` involves arithmetic and potentially additional checks of well-formedness, e.g., that the individual bytes are ASCII digits, or that the full string is valid UTF-8. Hence, we perform the parsing outside the ZK component as much as possible. To ensure the integrity of the proof, this pre-processing step uses the same encoding of attributes than the issuer of the credential. Thus, we require that the hash of the parsed data matches the hash used in the credential as commitment.

Note that our ZK-TPL compiler together with ZoKrates define the encoding of data and its representation in the rank-1 constraint system of the underlying SNARK. Therefore, any change to our compiler or in ZoKrates may render old proofs unverifiable. For short-lived or interactive scenarios, we thus require compatible encodings for both prover and verifier.

Example ZoKrates Program: Listing 9.3 presents an example ZK program, generated by our ZK-TPL compiler from the policy in Listing 9.2. For the inputs to the hash function, we opted to directly use `u32` arrays as expected by the provided implementation of SHA256. When using different hash function designs with ZK-friendly permutations such as GMiMC [Alb+19b] or Poseidon [Gra+21], the inputs and outputs can be represented as `field` elements. Thereby, we would be able to significantly improve the performance of the ZK program. With the goal in mind to be as widely usable as possible, we consider support for common hashes such as SHA256 essential.

9.4.3. Evaluation of Prototype

To evaluate our prototype implementation, we perform several benchmarks and compare them with the evaluation of the TPL system without any privacy features. Existing TPL benchmarks focus on the verification phase, which takes one to ten seconds for realistic policies and includes the retrieval of trust information from online registries (see also Chapter 10)

```
import "hashes/sha256/sha256" as sha256

def compare(u32[8] h1, u32[8] h2) -> bool:
  return h1[0] == h2[0] && h1[1] == h2[1] &&
         h1[2] == h2[2] && h1[3] == h2[3] &&
         h1[4] == h2[4] && h1[5] == h2[5] &&
         h1[6] == h2[6] && h1[7] == h2[7]

def main(u32[8] pub_hash, u32 currentYear,
         private u32 birthYear, u32 semester) -> bool:

   // Encode full credential, append SHA256 padding:
   u32[1][16] enc_cred = [[birthYear, semester,
                           2147483648, 0, 0, 0, 0, 0,
                           0, 0, 0, 0, 0, 0, 0, 64]]

   // Calculate age using private attribute:
   u32 age = currentYear - birthYear

   // Proof that attributes fulfill the age check
   // and the consistency of data used for the proof:
   return age >= 18
          && compare(pub_hash, sha256(enc_cred))
```

Listing 9.3.: ZoKrates program generated by compiling the TPL policy from Listing 9.2. Contains private `birthYear` attribute, revealed `semester` attribute, and (simplified) age check. It also proves the consistency of the attributes w.r.t. `pub_hash` commitment of the credential. The magic-numbers for the encoded credential are SHA256 padding-constants.

[MA22]. Standard TPL needs no setup phase, and the authentication phase is a trivial process for the user.

To measure the impact of the privacy extension to the existing TPL toolchain in Java, we run hyperfine[4] benchmarks on a Intel Core i7-8550U office laptop with 16 GB RAM running on Ubuntu 21.10. Our prover and verifier tools use ZoKrates, which we configured to use the bellmann backend with Groth16 [Gro16]. We evaluate the performance on the `ALT_BN128` [BN05] and `BLS12_381` [Bow17] curves; while the former provides 100 bit of security, the latter is slower but provides ≥ 117 bit of security [BD19; YKS19]. Additionally, we compare the performance of SHA256 with the ZK-friendly Poseidon hash function [Gra+21]. While Poseidon is faster and produces smaller proofs, SHA256 is commonly used in existing credential systems.

Evaluation Results: To illustrate the impact of our privacy extension, we divide our process (from Section 9.3.3) in two categories:

- The setup phase (compilation of policy, generating of keys/CRS) is a one-time phase and only performed once for each policy. Since those steps are only performed once, their duration is of less importance. The results of this steps are the policy program and thee ZoKrates keypair (verifier key and prover key), which need to be distributed to the corresponding parties. The size of this artifacts is of relevance, as some use cases might deal with users that are connecting to a service using a constrained connection (e.g., bluetooth [Abr+20]).
- The authentication phase (computation of witness and generation of proof by the user) and verification phase (verification of proof and execution of policy by the SP) are repeated phases and executed for each authentication process. Hence, the performance of these steps is important. This is also the case for the proof artifact, since it needs to be transmitted from the user to the SP. Further, the proof verification duration is of special interest, as the verifier deals with many users at the same time.

Table 9.1 shows the results of our evaluation. We observe that the verification is fast, and that the duration is independent of the utilized commitment function and curve. This is an important advantage, as the verifier has a need to verify a large amount of policies at the same

[4] `https://github.com/sharkdp/hyperfine`

Table 9.1.: Evaluation results of the example policy in Listing 9.2 for different commitments and curves.

		bn128		bls12_381	
		Poseidon	SHA256	Poseidon	SHA256
	security	100 bit		$\geq$ **117 bit**	
setup	compile policy [s]	1.80	**1.15**	1.83	1.16
	gen. keys [s]	**0.74**	9.11	1.23	16.09
authenticate	gen. witness [s]	**0.02**	0.31	**0.02**	0.33
	gen. proof [s]	**0.18**	1.28	2.04	27.73
	verify proof [s]	0.01	0.01	0.01	0.01
	policy prog. [kB]	659.10	17475.2	**643.99**	17468.74
	prover key [kB]	**452.76**	6406.62	658.29	9601.85
	verifier key [kB]	**1.88**	3.02	2.58	4.18
	proof [kB]	**0.95**	1.48	1.21	1.73

time. Also, the size of the proof transmitted to the verifier is around 1 kB in all cases. In our experiments we further observe that proof size and verification duration are also independent of the complexity of the policy. This is because the size of SNARK proofs always amounts to only 3 elliptic curve points and is thus independent of the witness size. Hence, the size of the transmitted proof only depends on the choice of commitment function and curve, and size of the revealed attributes.

The performance of the setup and authentication phases depend on the number of attributes that are part of the credential. This is because all attributes are part of the credential's signature and thus need to be part of the commitment in the proof (see Listing 9.3). The performance of calculating the commitment in the proof program is linear in the size of the attributes.

We also observe that the higher security of `BLS12_381` has a cost on artifact size and on performance. While this is especially the case for SHA256, the difference is potentially neglectable for Poseidon. Figure 9.6 visualizes the performance differences and further highlights the performance advantages of Poseidon. This shows why ZK-friendly commitment functions are needed. At the same time, our evaluation shows that existing credentials using SHA256 can be used in a privacy-friendly way without the need for changes at the credential issuer. However, performance is slow on the `BLS12_381` curve, and the distribution of the larger key material and policy program requires greater operational effort.

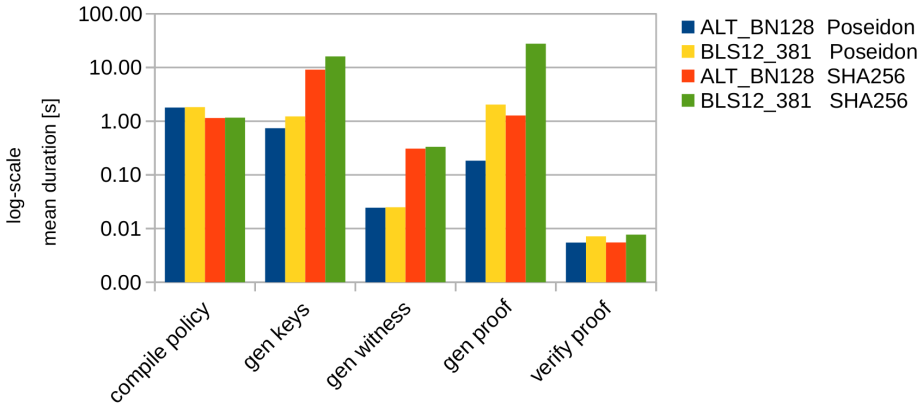

Figure 9.6.: Performance overhead evaluation results of the example policy in Listing 9.2 for different commitments and curves.

9.5. Evaluation & Discussion

9.5.1. Evaluation

This chapter focused on Goal 4 of our overall goals (see Chapter 5). Goal 4 is concerned with the privacy aspects of authentication. Based on this goal, in Section 9.2 we formulated the concrete requirements for our privacy-preserving access control system. Additionally, we added general requirements for policy systems and considered the other goals of this thesis, in specific modularity and extensibility (Goals 1 and 3). We now discuss the evaluation of our approach against those requirements. For each requirement, we briefly state the measure or design decision that ensures we comply with it (or note why we don't comply).

R1 Data Minimization: We provide a flexible system to disclose attributes of credentials selectively. Further, we automatically derive a proof that revealed attributes are part of the (signed) credential. By doing so, we ensure that the authenticity guarantees of the attributes are retained.

R2 Expressive Power: We fulfill this requirement by enabling policy authors to specify expressive predicates. Those predicates can be applied to attributes of the credential without revealing the attributes' value to the SP.

We note that a malicious policy author could define a predicate that reveals the attribute's value. In the same way, a malicious policy can reveal all attributes if the user does not pay attention. To mitigate this, we propose to authenticate and accredit SPs and establish a liability framework. Further, an SPs accreditation could be used to automatically restrict what data the SP can request.

R3 Unlinkability: Our approach always sends the credential's signature and commitment to the verifier. These values are always the same for presentations derived from the same credential – even if different attributes are revealed or no attributes are revealed. As this enables malicious SPs to link multiple showings, we don't comply with this requirement.

Mitigating this problem in our approach is not easily possible, as it would require modifications of the issuer (violating R7). Different types of commitment schemes are needed, as the hash digest enables a link between two presentations even if no attributes are revealed. However, the used SHA-2 family does not support the randomization of the hash digest (in contrast to, e.g., [Ped91]). As an alternative, we propose further research into approaches that also hide the hash digest. Doing so would require adding the verification of the issuer's signature on the hash digest to the presentation proof. Further research and implementation work in that direction is needed. Another alternative is to use a different signature scheme, e.g., [PS16; FHS19].

R4 Simple linking of predicates: We fulfill this requirement by allowing logical conjunction of constraints in the form of `checkQualified(DIDissuer), verifySignature(Cred, DIDissuer)`.[5]

R5 Expressive linking of predicates: We fulfill this requirement by using ZoKrates' operators[6] to allow logical disjunction of constraints as well as a combination of conjunction and disjunction. In TPL, a disjunction is defined by specifying multiple clauses for the same predicate name (see also Section 7.3); for example `allowed(Student) :- extract(Student, age, Age), Age >= 18.` and `allowed(Student) :- extract(Student, approved, yes).`

[5] The comma is the TPL operator for a logical *and*.

[6] `https://zokrates.github.io/language/operators.html`, accessed on 2023-08-18

R6 Inter-credential constraints: Our approach supports expressive inter-credential constraints. We achieve this by allowing to combine attributes of multiple credentials in the same constraint, e.g., `extract(GraduationCred, student_name, Name), extract(CVCred, person_name, Name)`.

R7 No modifications of the issuer: Since our system works with existing credentials and no specific schemes are needed, we comply with this requirement. In specific, we support established commitment schemes.[7] Additionally, since we reveal the credential's commitment digest and verify the signature separately, our approach is independent of the applied signature scheme.

R8 Practicability (Authentication): We provide the evaluation of our prototype implementation in Section 9.4.3. To evaluate the practicability, we focus on the authenticate phase. The goal was to initialize the presentation (calculate the proof) in around 1 second and deliver the verification result (and access decision) to the user in under 10 seconds [Nie97].

For the calculation of the credential presentation, the results depend on the selected parameters, i.e., the commitment scheme of the credential and the curve of the SNARK proof. Using the `ALT_BN128` curve (100 bits of security), the process takes less than 1 second for Poseidon and 1.5 seconds for SHA256, achieving our stated goal. On the `BLS12_381` curve (117 bits of security), the process takes around 2 seconds for Poseidon. However, with `BLS12_381`, the process takes 28 seconds for SHA256, which is close to being impractical.

For the verification of the presentation proof, the performance is 0.1 seconds, independent of the selected parameters, achieving our goal. The size of the data transmitted to the verifier is between 1–1.7 kB, which is a practical amount to transfer. Most importantly, both the verification duration as well as the size of the presentation are independent of the credential's size.

Hence, if support for SHA256-based credentials is needed (see also R7), and 100 bit-security is enough [Bar20, Section 5.6.1], we recommend using the `ALT_BN128` curve for practical performance. Otherwise, Poseidon-based credentials provide a practical performance on both curves.

[7] `https://zokrates.github.io/toolbox/stdlib.html`, accessed on 2023-08-18

R8 Practicability (Setup): Further, the compiled policy (presentation request) must be transmitted from the verifier to the user, a process that happens once per policy per user. During the first interaction between a user-verifier pair, the user must retrieve the prover key. To assess the impact of these costs, we consider the size of the presentation request and prover key (see Table 9.1).

For SHA256-based credentials, the request size is around 17 MB and the prover key is 6–9 MB of size, clearly representing a practical limitation. By using Poseidon, the request is approximately 0.6 MB large, while the size of the prover key is about 0.4–0.7 MB. However, the practicability of our approach can be improved by including common policies directly in the identity wallet, downloading the request over a faster channel, or applying caching. See also Section 9.5.2 for other alternatives.

Integration of privacy technology into access control systems: The goal was to integrate privacy technologies into an access control system. Further, we wanted to derive a privacy-preserving authentication system from an *existing* (policy-based) system. This focus ensures that we can re-use existing trust- and access-policies as much as possible instead of replacing them with a policy language specific to privacy. We achieve this goal by deriving a privacy-preserving presentation request from an access policy. In terms of implementation, we derive a presentation request for the SNARK zero-knowledge proof system from a TPL policy. The only modifications to the TPL policy are to mark the attributes and predicates that the user needs to reveal. The system then automatically derives privacy-preserving proofs from the user's credentials to probe that the user fulfills the SP's policy.

9.5.2. NIZK Setup

From a deployment point of view, policy-dependent setup phases may hamper the adoption of such a system. As this dependency is mainly influenced by the underlying proof system, an efficient proof system with a universal CRS is of paramount importance for more flexible applications.

Also, the need for a trusted third party (TTP) for the CRS generation is not ideal in some use cases. However, it has already been shown that the TTP can be replaced using secure multi-party computation for the setup algorithm [BGG18]. Alternatively, it is also possible to employ

transparent, subversion-resistant, or updatable proof systems [Ben+19; BFS20; Fuc18; Gro+18; ARS20], where knowledge of secret trapdoors no longer poses a threat.

9.5.3. Constant-length Attributes

We observe multiple restrictions inherent to the use of attributes with arbitrary types. Specifically, when dealing with string attributes, all the strings need to be encoded with a constant length. Otherwise, the length of the strings could reduce the anonymity set and the mere knowledge of the string lengths leaks sensitive information. This also extends to primitives that consume these strings, e.g., hash functions, as the number of compression function evaluations depends on the size of the input.

9.5.4. Future Work on NIZK Toolchains

While NIZKs and SNARKs are known for languages in all of NP (cf. [GOS06] and others), for practical purposes the situation is significantly different. Yet, as implementations of SNARKs gain better toolchains with support for higher-level abstractions, SNARKs can be applied to solve more interesting challenges. These toolchains need to abstract technical details such as rank-1 constraint systems and other arithmetization techniques to be useful for a wider audience. With ongoing scientific and engineering effort, these abstractions are rendered more efficient, less costly, and more expressive.

9.5.5. Future Work on Policy Authoring Tools

Since our work extends the capabilities of the TPL policy language, the GUI-based authoring tools (as discussed in Section 7.5.3) would need to be updated by future work. Attributes should be hidden by default and only be revealed when indicated. Non-technical policy authors should be able to use zero-knowledge features in a graphical manner without being familiar with any underlying details.

9.5.6. Communicating Privacy Implications to Users

While we extend the capabilities available to a policy designer, the consequences of revealing certain attributes also need to be explained to the

user. Some works have developed interfaces that highlight the revealed attributes and data flows to the user. Examples in various directions include Angulo et al.'s approach [Ang+11], which provides visualizations of policies, and Mikkelsen et al. [Mik+15] presenting a user interface to disclose attributes of a credential selectively. Alternatively, privacy metrics [WE18] offer tools to attach scores based on the processed data and the type of performed computations. Using these techniques may help visualize a user's potential privacy risks based on the policy in question.

Chapter 9 Conclusions

In this chapter, we discussed the topic of privacy-preserving (and enhancing) access control. Specifically, we focused on the context of our thesis: policy-based and credential-based access control. We conceptualized a generic way to extend access control systems with privacy features. We do so by automatically deriving a privacy-preserving presentation request and token from a high-level access policy.

The concrete privacy properties that can be achieved intrinsically depend on the policy and the data involved in evaluating this policy. Thus, we consider it of high importance that both the designer and the users interacting with such a policy system clearly understand the risks to the privacy of their personal data. Therefore, our system raises an interesting question on how the policies can be presented to the user intuitively so that they can perform an informed decision before presenting, for example, their credentials.

Further, we evaluated our approach, elaborating its strengths (i.e., flexibility and adaptability) and limitations (most notably, the user's likability). We also discussed the system's parameters (commitment scheme and elliptic curve for proofs) and practical parameter choices. The research results presented in this chapter have been published in one academic paper [Mor+22].

10

Privacy for Verification of DL-based Credentials

This chapter is based on the paper *Offline-verifiable Data from Distributed Ledger-based Registries* by More, Heher and Walluschek [MHW22]. Parts of this paper have been copied verbatim. The prototype was implemented by our student Clemens Walluschek.

Trust management systems often use registries of various forms to authenticate data or form trust decisions. Examples are revocation registries, accreditation registries, schema transformation lists, and trust status lists. Such registries are usually implemented in the form of a list or API published using a centralized server. By introducing Distributed Ledgers (DLs), it is also possible to create decentralized registries, for example using a smart contract as storage and interface (see Section 4.3). A verifier then queries the respective contract during verification of a credential or execution of a trust policy. While this ensures up-to-date information, the process requires the verifier to be online. Additionally, the connection from the verifier to the registry poses a privacy issue, as it leaks information about the user's behavior. To reduce this limitation, we extend existing ledger APIs to support results which are verifiable even in an offline setting.

Challenge: Availability & Privacy To retrieve DL-based data, the verifier communicates with the API of a DL node it trusts. While this ensures the freshness of the data, a network connection to this node is required. If the verifier is offline, it cannot retrieve a trustworthy copy of the data [Abr+20]. The same is true if the particular DL node used by the verifier is unavailable [Li+21].

User privacy poses an additional challenge. Such approaches don't provide undetectability of interactions—in other words, the contacted DL node learns about the showing of a credential, and about which verifier the credential was shown to (see also Section 3.2). Since verifiers are typically operated by the individual Service Provider (SP), this correlates with the user's associations [Chu+18]. Often, sensitive information such as physical location can be derived.

A solution to this problem would be to move the retrieval of DL data to the user, and then forward it to the verifier. However, as data provided by the DL API is currently not signed, trust in it is derived solely from the authenticity of the underlying connection with a trusted node. This is in contrast with comparable technologies, for example, OCSP stapling in TLS [III11].

Chapter 10 Goal:

We discuss the overall goals of this thesis in Chapter 5. In this chapter, we focus on the privacy Goal 4. More specifically, we tackle the aspect of Undetectability.

Chapter 10 Contributions & Outline:

Offline-verifiable Data from DL-based Registries: In this chapter, we extend the ideas of Abraham, More et al. [Abr+20] to solve the described problem with a generic *Ledger State Attestation (LSA)* system. Using this LSA system, a user can retrieve data from the DL and prove its provenance to an offline verifier. Since our approach provides a generic interface to the data stored on the ledger, the system can be used in different use cases. We introducing our approach in Section 10.1. To summarize our contribution, we differentiate between two types of attestations:

Node Attestations: As basis for ledger state attestations, we enable DL nodes to issue signed *node attestations* to users. Such an attestation contains the result of some specified operation on the DL, such as retrieving the current block hash or the result of a smart contract invocation. Additionally, it attests in an offline-verifiable way that the result matches the node's current view of the DL state. We achieve this without modifications to the code of the ledger clients but instead provide a wrapper around the node API. This approach is also transparent to the consensus protocol

used by the ledger; hence, it does not touch the ledger's trust model. The wrapper provides a generic attestation functionality and thereby supports all kinds of current and future use cases. Although this wrapper needs to be hosted directly on the nodes' servers, this only needs to be done once.

Aggregated Attestations: Additionally, we enable an user to retrieve such node attestations from multiple nodes aggregated into a single *aggregate attestation*. By retrieving node attestations from an appropriate set of DL nodes, the user can be reasonably sure that the aggregate attestation also includes node(s) that an unknown verifier trusts. The verifier can then verify the attestation without needing to communicate with the node(s) in question. As it can now trust the provided result, it can then use it to authenticate the user's credentials while remaining fully offline and without leaking information to the node or other third parties.

Implementation and Discussion: We demonstrate the feasibility of our approach using two implementations, which we present in Section 10.2. While our general architecture is independent of a concrete DL technology, in this proof of concept implementation we focus on the Ethereum stack. Our first implementation variant enables DL nodes to attest the current block hash, which then allows an offline verifier to establish trust in any DL-based data that the user provides. The second variant issues attestations of returned data from smart contract function calls. This enables users to specify custom queries or filters for the data they want to retrieve. In Section 10.3, we discuss trust assumptions and operational concerns of our approach. Further, we state the limitations and ideas for future work.

10.1. Concept

In this section, we introduce our approach on a conceptual level. First, we give an overview of the architecture of our system and how it connects with the existing DL architecture. Afterward, we discuss how the adapted architecture forms a **Ledger State Attestation (LSA)** system, which enables users to retrieve attestations of data stored on a DL.

10.1.1. Architecture Overview

There are several main components in our system, which we describe below. A high-level and generic overview of these components and how they are connected is shown in Figure 10.1, while Figure 10.2 offers a more detailed and Ethereum-focused picture.

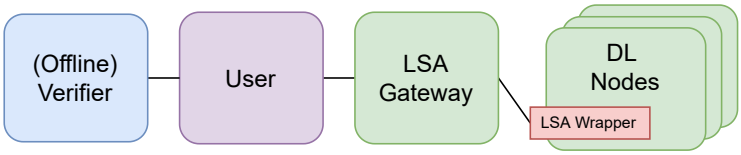

Figure 10.1.: High-level architecture of our Ledger State Attestation system.

DL Nodes: The distributed ledger (DL) is represented by its DL Nodes. These nodes communicate peer-to-peer, using a consensus protocol to agree on a shared state. DL Nodes provide an HTTP API to allow other entities to access the state of the DL.

LSA Wrapper: We add the LSA Wrapper component to each DL node. It wraps the node API, providing access to the same data but enriches the API functionality, adding a proof of provenance to the returned data. The wrapper provides the same API endpoints as the node API, so it is compatible with existing API clients.

LSA Gateway: We introduce the LSA Gateway stand-alone component to support the user by retrieving data from multiple (or all) DL nodes. It can be part of a node, run by the user, or be operated by a third party. We discuss the implications of this choice in more detail in Section 10.3. The gateway provides the same API endpoints as the node API and our LSA Wrapper. A user can then send a query to the gateway instead of to each node separately. The gateway forwards queries it retrieves from a user to the DL nodes, and aggregates their answers into a single response for the user. By doing so, it acts as a helper to collect node attestations from a set of DL nodes, and combines them into an aggregate attestation.

The LSA Gateway only forwards requests and aggregates signatures issued by the DL nodes, but does not sign anything. However, the user must trust the gateway to forward the request to all nodes, and trust it not to censor any replies.

User: The User wants to retrieve data from a DL and present it to a verifier. To retrieve data directly from the DL, the users interact with the API of a DL node. For data that can be presented to an offline verifier later, they instead interact with the LSA Gateway.

(Offline) Verifier: The (Offline) Verifier operated by a SP receives data from the user and needs to establish trust in this data. We consider a scenario where this verifier is offline, i.e., it cannot connect to any of the DL's nodes during verification.

Attestation: In DL-based trust management, the verifier is interested in the trustworthiness of a **claim** about the DL's state. This claim can, e.g., be an assertion of the current block hash, or of the return value of a smart contract function. We define an **attestation** as data combined with proof of authenticity. There are two types of attestations: A **node attestation**, created by an individual DL node, attests that the data reflects the data in its local storage. Combining such attestations of several nodes yields an **aggregate attestation**, attesting an agreement between all involved nodes.

10.1.2. Ledger State Attestation (LSA)

When a user shows some data to a verifier, this verifier needs to make sure that it can trust this data before relying on it for further processing. Trusting data coming from a DL-based registry means verifying that this data matches the data stored in the DL. An online verifier could check this by contacting a trustworthy DL node and comparing the data, or even by doing additional lookups on its own.

Since we consider the scenario where a verifier is offline, this online check is not possible. The underlying challenge is that an offline verifier has no reason to trust data that a user claims to have retrieved directly from a node's API. For example, a verifier cannot be sure if this data was really

retrieved from a DL, that the user did not alter the data, or that the data represents the latest state of the DL.

Attestation of state by a node: To mitigate this problem, we add the LSA Wrapper component to all nodes of the DL, wrapping the node API. This is the only modification to a DL node our approach requires, and it only needs to be done once and not for every use case. The wrapper enables DL nodes to issue attestations of data stored on the DL. While the default node API answers user queries by returning plain data, the wrapper additionally adds a proof to the response. To ensure the authenticity of the data, the wrapper creates this attestation proof using a private key of the node. To enable a verifier to decide if the presented information was fresh enough, the number of the current block and a low-resolution timestamp are also added to the attestation.

The attestation can then later be presented to an offline verifier which checks it to ensure that the presented data was returned by a specific DL node and has not been altered. After receiving some attestation (data and proof) from a user, the verifier uses the node's public key from a local trust store to verify the attestation, before processing the data.

Attestation by the whole DL: While this process ensures authenticity (and integrity) of the data with respect to one node, it means the user needs to select a node the verifier trusts. This is a problem since a user does not know, at the time of retrieving the attestation, which node(s) a verifier trusts. Additionally, this limits which verifier the user can present an attestation to, since different verifiers trust different nodes. To avoid this, the user would need to retrieve an attestation by all DL nodes.

In our LSA system, the gateway is used to provide users with an easy way to retrieve data, alongside a proof from all nodes. Since the data stored by each node was agreed upon using the DL's consensus protocol, the data returned is also the same for each node. But the proof returned by the nodes is different since it proves authenticity for a certain node. This allows that the gateway returns the data only once, and aggregates the proofs of the nodes into a single proof.

Protocol 1 shows all the required steps in this attestation process, while Protocol 2 does the same for an interactive showing to an offline verifier.

Our protocol relies on a multi-signature scheme MSIG for the attestation proof (see Section 4.6).

Attestation Phase:

on User

1. encode the user's claim:
   $\mathsf{query} \leftarrow \mathsf{Client.Encode}(\mathsf{call}, \mathsf{parameters})$
2. send encoded query to the LSA Gateway

on LSA Gateway

3. forward query to n nodes

on Node i of the DL (in parallel)

4. retrieve data by executing the user's query on the DL's state:
   $\mathsf{data} \leftarrow \mathsf{Node.Execute}(\mathsf{call}, \mathsf{parameters})$
5. create a node attestation statement:
   $\mathsf{epoch} \leftarrow$ low-resolution timestamp
   $\mathsf{NA}'_i \leftarrow (\mathsf{call}, \mathsf{parameters}, \mathsf{data}, \mathsf{epoch})$
6. create a multi-signature: $\sigma_i = \mathsf{MSIG.Sign}(\mathsf{sk}_i, \mathsf{NA}'_i)$
7. issue node attestation NA to the LSA Gateway:
   $\mathsf{NA}_i \leftarrow (\mathsf{NA}'_i, \sigma_i, \mathsf{pk}_i)$

on LSA Gateway

8. receive node attestations from n nodes and aggregate them:
   $\sigma = \mathsf{MSIG.ASigs}(\{(\sigma_i, \mathsf{pk}_i)\}_{\forall i \in [n]})$
9. issue ledger attestation LSA to the user:
   $\mathsf{LSA} \leftarrow (\mathsf{call}, \mathsf{parameters}, \mathsf{data}, \mathsf{epoch}, \sigma, \{\mathsf{pk}_i\}_{\forall i \in [n]})$

on User

10. receive LSA and store it for the specified call and $\mathsf{parameters}$
11. (*Optional:* verify LSA using local trust store)

Protocol 1: Attestation Protocol

Showing Phase (Offline):

on User

1. receive the verifier's request for DL data, which specifies the required call_V and the values for some $\mathsf{parameters}_V$ while leaving the values of other parameters $\mathsf{parameters}_U$ to the user
2. retrieve stored LSA for the specified call and send it to the verifier

on Verifier

3. verify the attestation $\mathsf{LSA} = (\mathsf{call}, \mathsf{parameters}, \mathsf{data}, \mathsf{epoch}, \sigma, \{\mathsf{pk}_i\}_{\forall i \in [n]})$:
   - check if enough signer keys $\{\mathsf{pk}_i\}_{\forall i \in [n]}$ are part of the trust store and that epoch is fresh enough
   - aggregate the signer keys: $\mathsf{pk} = \mathsf{MSIG.APKs}(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$
   - verify the aggregated signature: $\mathsf{MSIG.AVerify}((\mathsf{call}, \mathsf{parameters}, \mathsf{data}, \mathsf{epoch}), \sigma, \mathsf{pk}) = 1$
   - verify that $\mathsf{call} = \mathsf{call}_V$ and $\mathsf{parameters} = (\mathsf{parameters}_V, \mathsf{parameters}_U)$
   - check if $\mathsf{parameters}_U$ and data fulfill the verifier's policy
4. use the now-trusted data for further processing

Protocol 2: Offline Showing Protocol

10.2. Implementation

To show the feasibility of our LSA concept, we implement a prototype for the Ethereum stack. In this section and the following subsections, we describe how we applied our approach to the API of Ethereum nodes, and how the implementation can be used in practice. We focus on permissioned DLs, such as those used in the European Blockchain Services Infrastructure (EBSI) [Eur22].

On the server-side, we provide a wrapper for Ethereum's RPC API (see Section 4.3), which extends the API of Ethereum nodes to support signed responses. The LSA Gateway first forwards the user's query to the wrapped API of all applicable nodes. Then, it aggregates the received node attestations into one aggregate attestation. For the authenticity proofs, we use digital signatures issued individually by each node using their private key.

To be able to aggregate the signature of the individual nodes into one combined signature, we use the BLS signature scheme for the authenticity

proofs (see Section 4.6) [Bon+22]. An additional benefit of using BLS is its efficiency and small storage requirements, minimizing the overhead.

On the client side, we extend the *web3.js* library[1] to retrieve node and aggregate attestations. This allows a user to call contracts using well-established high-level calls, but retrieve the response value in a signed and offline-verifiable form.

To demonstrate the flexibility of our design, we implement two different variants, which differ in the type of data that gets attested.

Variant 1: Attestation of Data In this variant we enable an offline verifier to trust any raw data retrieved from the DL by the user. This then allows the verifier to execute a locally stored smart contract, or work with the data by other means.

Ethereum ledgers protect the integrity of their data using merkle trees: The block hash is the root hash of a merkle tree, formed by all transactions, smart contract code, and data stored in the ledger at a certain point in time [Woo+22]. A trustworthy attestation of the block hash therefore allows any data in that block, and any previous block, to be trusted. Another advantage of the attestation of the root hash is that it can be pre-computed. Since this needs to be done only once per block, this significantly reduces the load on the DL nodes while still allowing to establish trust in all data on the DL.

Thus, we create a mechanism to retrieve an attestation of the current root hash. This allows a user to retrieve any raw data, and the supplementary parts of the merkle tree, called merkle proof [JJ18], from any (single) node. Afterward, they send this data, the merkle proof, and the attestation of the root hash to the verifier. The verifier can then use this trusted hash to establish trust in the data.

Variant 2: Attestation of Smart Contract Response In our second variant, we enable users to query for and retrieve trustworthy data from the DL. To do so, we extend the functionality of the node API in a way that nodes can issue attestations for the return value of smart contract functions. Part of this attestation is also the address of the called smart contract, the executed function, and the call parameters. This enables

[1] `https://github.com/web3/web3.js`, see Section 4.3

users to send queries to a smart contract and get an attestation of the query result. To provide more flexibility, we do this by extending the `call` function of the web3 client library with the ability to request and handle signed attestations.

A user can simply execute the function in the same way as without the LSA system. Each node then executes the function call, creates a node attestation of the return value, and sends the signature to the gateway, which aggregates the signatures. The result is an aggregate attestation that contains the call and return value of a certain contract function, and proves the consensus of the nodes about this state. This attestation credential can be shown to an offline verifier and authenticated using the verifier's truststore. Since the attestation contains the needed data, the verifier does not need to execute a smart contract or evaluate a merkle proof.

10.2.1. Attestation & Showing Process

The process works as follows in both variants. The structure of this process is also shown in Figure 10.2.

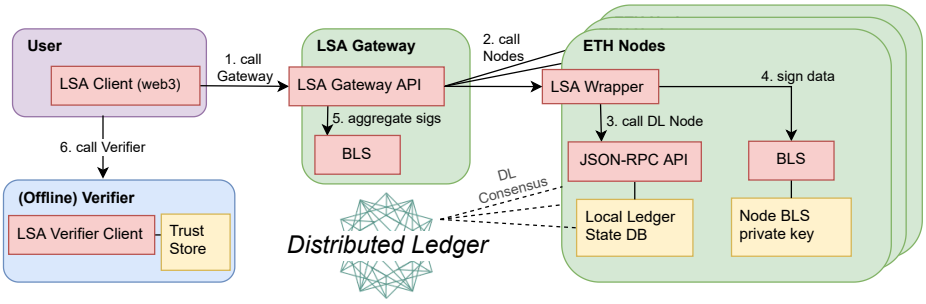

Figure 10.2.: The architecture of our Ledger State Attestation system, extending the functionality of Ethereum nodes and web3.js.

Attestation Phase

1. To fetch the attestation of some data, the user utilizes our modified web3 library to send the request to the LSA Gateway. In variant 1,

this is a request for the block hash, while in variant 2 this is a call to a smart contract function including parameters.

2. The LSA Gateway has a list of all nodes of the DL. It forwards the request to all nodes, which run Ethereum's RPC API with our wrapper.

3. Each node's wrapper first forwards the request to the RPC API of the node itself, which retrieves the requested data from its local version of the ledger state. In variant 2, it also retrieves and executes the called smart contract function using the EVM.

4. After retrieving the result, each of the nodes creates a node attestation using its own BLS private key and sends the result back to the gateway. In variant 1, the nodes attest the retrieved root hash of the current block, while in variant 2 they attest the result of the smart contract call.

5. The LSA Gateway then aggregates all signatures and sends the aggregate attestation back to the user, who stores it, e.g., in a digital wallet.

Showing Phase

6. Later, the user shows the aggregate attestation to an offline verifier, for example by sending the attestation by Bluetooth to the verification device. The offline verifier then uses their trust store to authenticate the attestation and thereby establishes trust in the attested data.

    - In variant 1, the verifier can now use the now-trusted block hash to authenticate the merkle proof the user also sent. It then checks the merkle proof to authenticate the rest of the data, which is only then used to locally execute a locally stored smart contract.

    - In variant 2, the verifier checks that the smart contract address and call specification contained in the attestation are the expected values.

10.2.2. Evaluation of Prototype

We consider the performance of our proof of concept implementation. To do this, we contrast it with traditional online verification. We identify the following additional

Attestation Phase

Attestation retrieval requires an additional network round trip compared to a traditional online query, in scenarios where the LSA Gateway is not co-located on the user device. Quantifying this overhead exactly is difficult, as it varies based on the physical location of, and connection topology between the various entities. However, given that even a transatlantic round trip typically takes only around 90 ms [Ver22], we consider this to be negligible.

Data attestation requires each node to create a signature over the data retrieved from the DL. Using BLS signatures with 128-bit security, as in our implementation, signature creation takes ≈0.3 ms on a typical consumer laptop [Bon+22].

Data retrieval of DL data by any one individual node incurs no additional overhead compared to the traditional online flow. As the LSA Gateway sends queries to all nodes in parallel, the query time in the worst case should be no worse than for a single node. However, it is worth noting that, when viewed across the entire ledger, our scheme induces additional load. While in the traditional model the online verifier only sends its query to a single trusted node which has to perform data retrieval, in our case, the LSA Gateway sends this query to many different nodes, each of which has to perform the operation.

Showing Phase

During verification the user needs to transmit the retrieved LSA to the verifier. Since this communication could happen on constrained devices and a slow channel, we also consider the storage size of an attestation. In BLS, both signature and public key are encoded as single group elements [Bon+22]. Thus, an aggregated signature uses 48 bytes, with an additional 48 bytes per public key. To evaluate the impact of this storage overhead, we measure transmission of an attestation over 10 kB of data and 20 public keys using mid-range smartphones: a Samsung Galaxy XCover Pro

and a Google Pixel 1. This results in a transmission time of ≈150 ms, even using Bluetooth 4.2.

Then, the verifier needs to **verify the LSA's signature**. For BLS signatures with 128-bit security, the verification takes ≈2.7 ms on a typical laptop [Bon+22].

We note that transmission time, scaling with the size of the transmitted data and number of involved nodes, appears to be the primary driver of verification time. This presents potential optimizations by reducing the size of the transmitted LSA. For example, the public key space requirement could be removed by also aggregating the BLS public keys. On a permissioned ledger with a stable node membership, public keys could be outright omitted from the attestation, and verification could be performed using a complete trust store located at the verifier. Regardless, we consider a total duration overhead of ≈153 ms to be negligible for an interactive showing [Nie97].

10.3. Discussion

10.3.1. Evaluation

In this chapter, we focused on Goal 4 of our overall goals (see Chapter 5). In specific, we focused on the aspect of Undetectability (see Section 3.2).

The goal was to hide the behavior of users from credential issuers, i.e., to make sure the issuers don't learn when and where a uses presents their credentials. In the context of DLs, the issuer is represented by a node of the DL. We achieve this by removing the DL nodes from the interactive authentication process between a user and the SP. By doing so, we also achieve our side-goal to enable the verification of credentials in the offline setting. As opposed to the state of the art [Abr+20], we provide a simple query system to enable the attestation of any data stored on the ledger, without the need to modify the system for each use case.

10.3.2. Related Work

Various systems and methods in the blockchain world use data stored in a DL, but all of those systems have online components that directly interact with the ledger. E.g., Layer 2 protocols [Gud+20] move a large number of transactions from a ledger to an off-chain service to increase

performance and reduce cost. Systems based on the layer 2 approach interact with a smart contract and thus require a connection to the DL. The same requirement exists for Ethereum's Light Client, which fetches a state root from a trusted node [CBC21]. Another example is inter-ledger communication [Zam+21], which is used to transfer assets from one ledger to another. This transfer requires a trusted third party with a connection to both ledgers.

To prove the provenance of data, TLS-N[2] uses a more generic approach by extending the TLS handshake to enable a server to notarize a TLS session. TLSNotary [TLS14] and DECO [Zha+20] are concerned with the attestation of access protected web data to a third party. This is realized by involving a third party (oracle) trusted by the verifier in the TLS session with the server.

Some revocation systems work without a direct connection between the verifier and the revocation authority. One common example of this is the verification of TLS certificates: OCSP stapling is a TLS extension that allows a certificate subject (web server) itself to acquire the status information of its certificate [III11]. This information is signed by a status authority, which ensures that a verifier can trust the information. The subject can then provide it to the verifier (web browser), and the verifier does not require a direct connection to the status authority.

The system by Abraham, More et al. [Abr+20] allows the offline verification of DL-based revocation information. However, it is limited to only the revocation use case. Modifying this system for other use cases is possible, but each additional use case requires adaptions on all of the DL's nodes. Hence, we extended this approach to our generic ledger attestation system.

10.3.3. Trust Assumptions

The user must trust the verifier's trusted DL nodes to provide truthful attestations. This assumption is also made in the online case, and is not unique to our work.

Additionally, heading into an offline scenario the user relies on the provided attestation being valid. It is not a negligible concern that the LSA Gateway returns a bogus attestation. In order for the user to verify the provided

[2] `https://github.com/tls-n`, accessed on 2023-08-22

attestation, they would need to have a list of all DL nodes and their keys on their device. In general, this is not trivial (see also Section 10.3.4). Therefore, the user must trust the LSA Gateway to provide a valid attestation. They may also retrieve attestations from multiple different LSA Gateways. As long as at least one returns a valid attestation, the user device can successfully complete the LSA process.

The verifier has some trust policy that relies on the truthfulness of some subset of DL nodes. To verify the attestation proof, the verifier also has a store with the public keys of the nodes it trusts. This does not require additional assumptions beyond those already made in the online scenario. The LSA Gateway simply retrieves attestations from all DL nodes, including the nodes trusted by the verifier, and aggregates them. The verifier's trust in the aggregate attestation derives from the inclusion of attestations by nodes it trusts. It does not need to trust the LSA Gateway. Indeed, the existence of the gateway is transparent to the verifier.

10.3.4. Operational Concerns

To forward requests, the LSA Gateway requires an up-to-date list of all DL nodes.

Maintaining such a list is, in general, not a trivial task on permissionless ledgers. It thus intuitively makes sense to separate the LSA Gateway from the user device and, for example, to include it into one (or more) well-known nodes of the DL. This co-location might enable use of DL information already-available to the node to contact the other nodes.

For some DL setups like permissioned DLs, the set of nodes is well-known, static, or otherwise can be easily obtained. In that case, it makes sense to instead include the LSA Gateway into the user's client device. This eliminates the trust considerations towards the LSA Gateway outlined in Section 10.3.3.

Availability: In practice, it may not be possible for the LSA Gateway to reach all DL nodes. This can happen due to network issues, maintenance, or as the result of a DoS attack [Li+21]. In this situation, it may only be possible for the LSA Gateway to provide an incomplete attestation. At what point it should give up and do so, and how this would be communicated to the user, are open questions.

Additionally, if the verifier expects an attestation was signed by *all* nodes of a certain trust subset, an availability issue arises: if the LSA Gateway was not able to reach all of these nodes, the resulting incomplete attestation will be rejected.

To mitigate this, verifiers could use a threshold policy. Using the list of public keys that are part of an attestation, the verifier first verifies the aggregated signature on the data. It then checks if at least k nodes from its trust store signed the provided aggregate, and accepts it if so.

Required Modifications: Modifications to existing systems are always a challenge, especially to nodes in a distributed system. An advantage of our approach is that the only such modification is the addition of the LSA Wrapper to the nodes, which provides generic attestation and can thus be employed in various use cases. Such a modification could be for example performed during the setup of the system, and only the nodes considered by any verifier need to be modified. This is in contrast to the state of the art, where each use case requires an additional modification to the DL nodes, which is often not feasible during operation.

10.3.5. Limitations & Future Work

Attestation Freshness: A limitation of both stated approaches is the fact that information authenticated using such attestations is less up-to-date than information directly retrieved from a registry. While this is an acceptable trade-off for some use cases, other use cases require more timely information. Depending on the concrete requirements on freshness, we can mitigate the limitation up to some extent by utilizing the network connection of the user. In a scenario where the user is online during or shortly before the interaction with the (offline) verifier, they can retrieve a fresh attestation. This makes sense for a verifier operated on a constrained device and is especially useful for the user's privacy since it facilitates *unobservability* of the interaction.

Synchronized Time-stamping: In our scheme, we assume that the nodes queried by the LSA Gateway will typically agree on the state of the DL. This results in the returned node attestations having identical content, allowing the node attestations to be aggregated into a single aggregate attestation.

However, the inclusion of a timestamp, low-resolution as it might be, in the attested claim complicates this. For any variety of reasons, the attestations returned by two nodes may end up in two (adjacent) epochs. Since the epoch is part of the signed data, this difference makes it impossible to aggregate the signatures.

One potential workaround would be for the LSA Gateway to include an epoch derived from its local timestamp in its query to the DL nodes. The DL nodes could then verify that the epoch is within an acceptable interval of their local clock time, and issue their attestation with the requested epoch. This ensures that all nodes issue their attestations for the same epoch, thereby also for the same claim.

Node Discovery: Our approach works well for permissioned DLs such as consortium ledgers, but node discovery is a challenge in open architectures. In permissioned DLs like the European Blockchain Services Infrastructure (EBSI), the set of nodes is known and changes relatively rarely [Eur22]. On the other hand, permissionless ledgers like mainnet Ethereum have a large and unstable set of nodes, and no node knows all other nodes. In our naive implementation, as all nodes perform the attestation process, the LSA Gateway needs a list of those nodes. This presents a significant challenge when applying our approach to such a permissionless ledger.

While some node discovery systems exist [MM02],[3] future work is needed to access if they are applicable for our system and in what way they can be used by a verifier to create the required trust store.

Chapter 10 Conclusions

In many previous decentralized trust systems, an implicit always-online requirement significantly hinders practical applicability. We resolve this issue by applying the battle-tested concept of OCSP stapling to the distributed ledger ecosystem.

After collecting signed attestations of the ledger's current state from a sufficiently large subset of DL nodes while online, the user can present this aggregate attestation to a verifier later in an offline setting. The verifier can use its local trust store to verify that the claimed state was attested by

[3] e.g., `https://github.com/ethereum/devp2p/blob/master/discv4.md` and `https://eth.wiki/en/ideas/kademlia-peer-selection`, accessed on 2022-07-01

nodes it trusts, establishing trust in the data itself. This allows the data to be used to make informed decisions regarding the user's credentials.

In this chapter, we introduced Ledger State Attestations, which allow arbitrary queries to DL nodes' HTTP API to retrieve attested results. This serves as the basis for almost any imaginable use case with only a single adjustment to the underlying DL's nodes and is a significant improvement over the state of the art. Additionally, our LSA approach enables the undetectability of interactions with the verifier, which is essential in ensuring users' privacy.

Furthermore, we provided a proof of concept implementation for Ethereum-based ledgers. We evaluate this implementation, demonstrating the practical feasibility of our scheme. Finally, we discussed the implications of the LSA concept in terms of performance impact, added trust requirements, and operational concerns.

The research results presented in this chapter have been published as part of an academic paper [MHW22].

Thesis Conclusions

This thesis focused on improving the trust verification aspect of electronic transactions in a heterogeneous context.

To satisfy diverse use cases, we considered electronic transactions that consist of multiple digital credentials. Each of those credentials provides a different trust aspect to the verification and is issued by a different issuer. These issuers can be qualified in a different trust scheme, i.e., authorized by a separate entity following a different governance framework. So far, manual effort is needed to set up and maintain trust information retrieval for each scheme. We addressed this challenge by proposing a **trust management infrastructure** based on the Domain Name System (DNS). Using this infrastructure, a verifier establishes trust in a previously unknown trust scheme by simply configuring a single human-readable identifier. Additionally, we enabled trust schemes that recognize other trust schemes to publish this recognition in our system. To account for different understandings (and accordions) of trust, we proposed automated trust translations.

When an SP relies on a trust scheme to assess and certify the trustworthiness of some information, it depends on that scheme's understanding of trust. However, the rules about the trustworthiness of data on hand are specific to the concrete SP or a business use case and are usually unknown to a trust scheme operator. To mitigate this and to allow verifies to define their own view of trust, we introduced a **expressive trust policy system**. To be future-proof, we added concepts that enable the easy extensibility of our policy system without requiring modifications to the policy language itself. We showed how this trust policy system can be used with our DNS-based trust infrastructure and distributed ledger-based trust management in the novel self-sovereign identity model.

The large variety of credential formats and schemata is another challenge in the global context. A verifier needs to understand the semantics of the data encoded in a credential but cannot do so if the credential is encoded in an unknown credential schema. We tackled this challenge by introducing a generic framework for **trustworthy credential transformations**. Using this framework, a verifier automatically retrieves the data needed to transform a credential into a format and schema it can parse.

We further considered privacy aspects when presenting credentials with sensitive attributes. Ensuring privacy in complex systems presents a considerable challenge. This thesis considered privacy aspects of an electronic transaction's content and the user's behavior.

Various privacy- enhancing/-preserving technologies exist to protect the user's privacy. However, integrating these technologies into existing access control systems is not straightforward. We tackled this challenge by extending **expressive access control systems with privacy features**. Our approach re-uses existing access control policies, requiring only minor modifications to define which attributes need to be revealed. We then automatically derive a zero-knowledge-based access-control flow from the policy. In doing so, we effectively turn an existing access control system into a privacy-preserving access control system.

During the transaction verification, the verifier queries various registries, e.g., to retrieve trust status information of the credentials. While this ensures trustworthy information, the connection from the SP to the registry poses a privacy issue, as it leaks information about the user's behavior. In the case of distributed ledger-based registries, this leaks information about the user's behavior to the ledger node. We resolved these issues by extending existing ledger APIs to support **ledger state attestation**: generic results that are trustworthy without directly communicating with the DL node during the interaction with a verifier. By introducing a simple query system, we enabled the attestation of any data stored on the ledger without the need to modify the system for each use case. This attestation enables a user to prove the provenance of any DL-based data to an offline SP.

We also demonstrated the practicability of our proposals and discussed their limitations.

Bibliography

[Abd+23] Behzad Abdolmaleki, Noemi Glaeser, Sebastian Ramacher, and Daniel Slamanig. *Universally Composable NIZKs: Circuit-Succinct, Non-Malleable and CRS-Updatable*. IACR Cryptol. ePrint Arch. (2023), page 97 (cited on page 40).

[ARS20] Behzad Abdolmaleki, Sebastian Ramacher, and Daniel Slamanig. *Lift-and-Shift: Obtaining Simulation Extractable Subversion and Updatable SNARKs Generically*. CCS. ACM, 2020, pages 1987–2005 (cited on pages 40, 210).

[Abr17] Andreas Abraham. *Whitepaper: Self-Sovereign Identity*. Technical report. online, accessed on 27 April 2022. 2017. https://technology.a-sit.at/en/whitepaper-self-sovereign-identity/ (cited on page 47).

[Abr22] Andreas Abraham. *Qualified Self-Sovereign Identity: Addressing the gaps between Self-Sovereign Identity and traditional Identity Systems*. PhD Thesis. Sep 2022. doi:10.13140/RG.2.2.29266.22728 (cited on page 13).

[Abr+19] Andreas Abraham, Felix Hörandner, Olamide Omolola, and Sebastian Ramacher. *Privacy-Preserving eID Derivation for Self-Sovereign Identity Systems*. ICICS. Volume 11999. LNCS. Springer, 2019, pages 307–323 (cited on page 184).

[Abr+21] Andreas Abraham, Karl Koch, Stefan More, Sebastian Ramacher, and Miha Stopar. *Privacy-Preserving eID Derivation to Self-Sovereign Identity Systems with Offline Revocation*. TrustCom. IEEE, 2021, pages 506–513 (cited on pages 8, 184).

[Abr+20] Andreas Abraham, Stefan More, Christof Rabensteiner, and Felix Hörandner. *Revocable and Offline-Verifiable Self-Sovereign Identities*. TrustCom. IEEE, 2020, pages 1020–1027 (cited on pages 8, 39, 50, 204, 213–214, 225–226).

[ASM21] Andreas Abraham, Christopher Schinnerl, and Stefan More. *SSI Strong Authentication using a Mobile-phone based Identity Wallet Reaching a High Level of Assurance*. SECRYPT. SCITEPRESS, 2021, pages 137–148 (cited on page 8).

[ATK18] Andreas Abraham, Kevin Theuermann, and Emanuel Kirchengast. *Qualified eID Derivation Into a Distributed Ledger Based IdM System*. TrustCom/BigDataSE. IEEE, 2018, pages 1406–1412 (cited on page 10).

[Ala19] Alain Durand, ICANN Office of the CTO. *Digital Object Architecture and the Handle System*. https://www.icann.org/en/system/files/files/octo-002-14oct19-en.pdf. online, accessed on 13 September 2023. 2019 (cited on pages 94–95).

[Alb+21] Lukas Alber, Stefan More, Sebastian Mödersheim, and Anders Schlichtkrull. *Adapting the TPL Trust Policy Language for a Self-Sovereign Identity World*. Open Identity Summit. Volume P-312. LNI. Gesellschaft für Informatik e.V., 2021, pages 107–118 (cited on pages 7, 103, 121, 147, 262).

[AMR20] Lukas Alber, Stefan More, and Sebastian Ramacher. *Short-Lived Forward-Secure Delegation for TLS*. CCSW@CCS. ACM, 2020, pages 119–132 (cited on pages 8, 38).

[Alb+19a] Lukas Alber, Stefan More, Stephanie Weinhardt, and Olamide Omolola. *LIGHTest D6.5 Open Source Tool for Visualization and Editing of Trust Policies*. https://www.lightest.eu/static/deliverables/D6.5.pdf. online, accessed on 15 March 2023. LIGHTest Consortium, 2019 (cited on page 134).

[AW19] Lukas Alber and Stephanie Weinhardt. *LIGHTest D6.9 Usability and Interaction Design*. https://www.lightest.eu/static/deliverables/D6.9.pdf. online, accessed on 15 March 2023. LIGHTest Consortium, 2019 (cited on page 134).

[Alb+19b] Martin R. Albrecht, Lorenzo Grassi, Léo Perrin, Sebastian Ramacher, Christian Rechberger, Dragos Rotaru, Arnab Roy, and Markus Schofnegger. *Feistel Structures for MPC, and More*. ESORICS (2). Volume 11736. LNCS. Springer, 2019, pages 151–171 (cited on page 202).

[Ale+17] Nikolaos Alexopoulos, Jörg Daubert, Max Mühlhäuser, and Sheikh Mahbub Habib. *Beyond the Hype: On Using Blockchains in Trust Management for Authentication*. TrustCom/BigDataSE/ICESS. IEEE, 2017, pages 546–553 (cited on pages 34, 101).

[Aly+21] Abdelrahaman Aly, K Cong, D Cozzo, M Keller, E Orsini, D Rotaru, O Scherer, P Scholl, N Smart, T Tanguy, et al. *SCALE–MAMBA v1.14 : Documentation*. 2021 (cited on page 142).

[And04] Anne H. Anderson. *An Introduction to the Web Services Policy Language (WSPL)*. POLICY. IEEE, 2004, pages 189–192 (cited on page 110).

[Ang+11] Julio Angulo, Simone Fischer-Hübner, Tobias Pulls, and Ulrich König. *HCI for Policy Display and Administration*. In:

Privacy and Identity Management for Life. Springer, 2011, pages 261–277 (cited on page 211).

[Are+05a] Roy Arends, Rob Austein, Matt Larson, Dan Massey, and Scott Rose. *DNS Security Introduction and Requirements*. RFC 4033 (2005), pages 1–21 (cited on pages 36, 66).

[Are+05b] Roy Arends, Rob Austein, Matt Larson, Dan Massey, and Scott Rose. *Resource Records for the DNS Security Extensions*. RFC 4034 (2005), pages 1–29 (cited on page 66).

[Ash+03] Paul Ashley, Satoshi Hada, Günter Karjoth, Calvin Powers, and Matthias Schunter. *The Enterprise Privacy Authorization Language (EPAL)*. Report. W3C, 2003. https://www.w3.org/2003/p3p-ws/pp/ibm3.html (cited on page 110).

[BD19] Razvan Barbulescu and Sylvain Duquesne. *Updating Key Size Estimations for Pairings*. J. Cryptol. 32.4 (2019), pages 1298–1336 (cited on page 204).

[Bar20] Elaine Barker. *SP 800-57. Recommendation for Key Management: Part 1 – Generals*. Technical report. Gaithersburg, MD, USA, 2020 (cited on page 208).

[BN05] Paulo S. L. M. Barreto and Michael Naehrig. *Pairing-Friendly Elliptic Curves of Prime Order*. SAC. Volume 3897. LNCS. Springer, 2005, pages 319–331 (cited on page 204).

[Bar+13] Mark Bartel, John Boyer, Barb Fox, and Brian LaMacchia Ed Simon. *XML Signature Syntax and Processing*. W3C Recommendation. W3C, 11 Apr 2013. https://www.w3.org/TR/2013/REC-xmldsig-core1-20130411/ (cited on page 153).

[BFG10] Moritz Y. Becker, Cédric Fournet, and Andrew D. Gordon. *SecPAL: Design and semantics of a decentralized authorization language*. J. Comput. Secur. 18.4 (2010), pages 619–665 (cited on page 110).

[BRS12] Moritz Y. Becker, Alessandra Russo, and Nik Sultana. *Foundations of Logic-Based Trust Management*. IEEE S&P. IEEE, 2012, pages 161–175 (cited on page 110).

[BS04] Moritz Y. Becker and Peter Sewell. *Cassandra: Distributed Access Control Policies with Tunable Expressiveness*. POLICY. IEEE, 2004, pages 159–168 (cited on page 110).

[Bel+20] Rafael Belchior, Benedikt Putz, Günther Pernul, Miguel Correia, André Vasconcelos, and Sérgio Guerreiro. *SSIBAC: Self-Sovereign Identity Based Access Control*. TrustCom. IEEE, 2020, pages 1935–1943 (cited on pages 111, 189).

[Ben+19] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. *Scalable Zero Knowledge with No Trusted Setup*. CRYPTO (3). Volume 11694. LNCS. Springer, 2019, pages 701–732 (cited on page 210).

[BBR85] Charles H. Bennett, Gilles Brassard, and Jean-Marc Robert. *How to Reduce Your Enemy's Information (Extended Abstract)*. CRYPTO. Volume 218. LNCS. Springer, 1985, pages 468–476 (cited on page 3).

[Ber17] Hal Berghel. *Equifax and the Latest Round of Identity Theft Roulette*. Computer 50.12 (2017), pages 72–76 (cited on page 33).

[Ber20] Hal Berghel. *The Equifax Hack Revisited and Repurposed*. Computer 53.5 (2020), pages 85–90 (cited on page 33).

[BT11] Elisa Bertino and Kenji Takahashi. Artech, 2011 (cited on page 152).

[Bic+15a] Patrik Bichsel, Jan Camenisch, Maria Dubovitskaya, Robert R. Enderlein, Stephan Krenn, Ioannis Krontiris, Anja Lehmann, Gregory Neven, Christian Paquin, Franz-Stefan Preiss, Kai Rannenberg, and Ahmad Sabouri. *An Architecture for Privacy-ABCs*. In: *Attribute-based Credentials for Trust*. Springer, 2015, pages 11–78 (cited on page 188).

[Bic+15b] Patrik Bichsel, Jan Camenisch, Maria Dubovitskaya, Robert R. Enderlein, Stephan Krenn, Anja Lehmann, Gregory Neven, and Franz-Stefan Preiss. *Cryptographic Protocols Underlying Privacy-ABCs*. In: *Attribute-based Credentials for Trust*. Springer, 2015, pages 79–108 (cited on page 188).

[Bit+12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. *From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again*. ITCS. ACM, 2012, pages 326–349 (cited on page 40).

[BFL96] Matt Blaze, Joan Feigenbaum, and Jack Lacy. *Decentralized Trust Management*. IEEE S&P. IEEE, 1996, pages 164–173 (cited on page 15).

[Blo81] Arthur Bloch. *Murphy's Law Book Two: More Reasons why Things Go Wrong!* Book 2. Magnum, 1981. ISBN 9780417064505 (cited on page 12).

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. *Non-Interactive Zero-Knowledge and Its Applications (Extended Abstract)*. STOC. ACM, 1988, pages 103–112 (cited on page 40).

[BRA23] Ricardo Bochnia, Daniel Richter, and Jürgen Anke. *Lifting the Veil of Credential Usage in Organizations: A Taxonomy*. Open Identity Summit. Volume P-335. LNI. Gesellschaft für Informatik e.V., 2023 (cited on page 152).

[Bol03] Alexandra Boldyreva. *Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme*. PKC. Volume 2567. LNCS. Springer, 2003, pages 31–46 (cited on page 39).

[BO05] Piero A. Bonatti and Daniel Olmedilla. *Driving and Monitoring Provisional Trust Negotiation with Metapolicies*. POLICY. IEEE, 2005, pages 14–23 (cited on page 110).

[BOP06] Piero A. Bonatti, Daniel Olmedilla, and Joachim Peer. *Advanced Policy Explanations on the Web*. ECAI. Volume 141. Frontiers in Artificial Intelligence and Applications. IOS Press, 2006, pages 200–204 (cited on page 110).

[BS00] Piero A. Bonatti and Pierangela Samarati. *Regulating service access and information release on the Web*. CCS. ACM, 2000, pages 134–143 (cited on page 110).

[BDN18] Dan Boneh, Manu Drijvers, and Gregory Neven. *Compact Multi-signatures for Smaller Blockchains*. ASIACRYPT (2). Volume 11273. LNCS. Springer, 2018, pages 435–464 (cited on page 39).

[Bon+22] Dan Boneh, Sergey Gorbunov, Riad S. Wahby, Hoeteck Wee, Christopher A. Wood, and Zhenfei Zhang. *BLS Signatures*. Internet-Draft draft-irtf-cfrg-bls-signature-05. https://www.ietf.org/archive/id/draft-irtf-cfrg-bls-signature-05.txt. IETF Secretariat, 2022. https://www.ietf.org/archive/id/draft-irtf-cfrg-bls-signature-05.txt (cited on pages 221, 224–225).

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. *Short Signatures from the Weil Pairing*. J. Cryptol. 17.4 (2004), pages 297–319 (cited on page 39).

[Bow17] Sean Bowe. *BLS12-381: New zk-SNARK elliptic curve construction*. 2017. https://electriccoin.co/blog/new-snark-curve/ (cited on page 204).

[BGG18] Sean Bowe, Ariel Gabizon, and Matthew D. Green. *A Multi-party Protocol for Constructing the Public Parameters of the Pinocchio zk-SNARK*. Financial Cryptography Workshops. Volume 10958. LNCS. Springer, 2018, pages 64–77 (cited on page 209).

[Bra97] Scott O. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. RFC 2119 (1997), pages 1–3 (cited on page 106).

[Bra11] Stefano Braghin. *Advanced languages and techniques for trust negotiation*. PhD Thesis. University of Insubria, Italy, 2011 (cited on pages 110–111).

[BW90] Louis Brandeis and Samuel Warren. *The right to privacy*. Harvard law review 4.5 (1890), pages 193–220 (cited on page 24).

[Bra93] Stefan Brands. *Untraceable Off-line Cash in Wallets with Observers (Extended Abstract)*. CRYPTO. Volume 773. LNCS. Springer, 1993, pages 302–318 (cited on page 189).

[Bra95] Stefan Brands. *Restrictive Blinding of Secret-Key Certificates*. EUROCRYPT. Volume 921. LNCS. Springer, 1995, pages 231–247 (cited on page 189).

[BL16] Bud P. Bruegger and Peter Lipp. *LIGHT^{est} - A Lightweight Infrastructure for Global Heterogeneous Trust Management*. Open Identity Summit. Volume P-264. LNI. GI, 2016, pages 15–26 (cited on pages 42–43, 48, 94, 261–262).

[BÖ14] Bud P. Bruegger and Eray Özmü. *A DNSSEC-based trust infrastructure*. Open Identity Summit. Volume P-237. LNI. GI, 2014, pages 133–139 (cited on page 94).

[BKW13] Johannes Buchmann, Evangelos G. Karatsiolis, and Alexander Wiesmaier. *Introduction to Public Key Infrastructures*. Springer, 2013 (cited on pages 12–14, 18–19).

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. *Transparent SNARKs from DARK Compilers*. EUROCRYPT (1). Volume 12105. LNCS. Springer, 2020, pages 677–706 (cited on page 210).

[But+14] Vitalik Buterin et al. *A next-generation smart contract and decentralized application platform*. white paper 3.37 (2014) (cited on page 35).

[Cac+16] Christian Cachin et al. *Architecture of the hyperledger blockchain fabric*. Workshop on distributed cryptocurrencies and consensus ledgers. Volume 310. 4. Chicago, IL. 2016 (cited on page 35).

[Cal+07] Jon Callas, Lutz Donnerhacke, Hal Finney, David Shaw, and Rodney Thayer. *OpenPGP Message Format*. RFC 4880 (2007), pages 1–90 (cited on page 153).

[Cama] Cambridge University Press. *Identifier*. In: *Cambridge Dictionary*. https://dictionary.cambridge.org/dictionary/english/identifier (cited on page 13).

[Camb] Cambridge University Press. *Identity*. In: *Cambridge Dictionary*. https://dictionary.cambridge.org/dictionary/english/identity (cited on page 13).

[Camc] Cambridge University Press. *Privacy*. In: *Cambridge Dictionary*. https://dictionary.cambridge.org/dictionary/english/privacy (cited on page 24).

[CH02] Jan Camenisch and Els Van Herreweghen. *Design and implementation of the* idemix *anonymous credential system*. CCS. ACM, 2002, pages 21–30 (cited on page 189).

[Cam+14] Jan Camenisch, Stephan Krenn, Anja Lehmann, Gert Læssøe Mikkelsen, Gregory Neven, and Michael Østergaard Pedersen. *ABC4Trust D3.1 Scientific Comparison of ABC Protocols*. online, accessed on 21 August 2023. 2014 (cited on page 189).

[CL01] Jan Camenisch and Anna Lysyanskaya. *An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation*. EUROCRYPT. Volume 2045. LNCS. Springer, 2001, pages 93–118 (cited on page 189).

[CL02] Jan Camenisch and Anna Lysyanskaya. *A Signature Scheme with Efficient Protocols*. SCN. Volume 2576. LNCS. Springer, 2002, pages 268–289 (cited on pages 158, 184, 190).

[Cam+15] Jan Camenisch, Sebastian Mödersheim, Gregory Neven, Franz-Stefan Preiss, and Alfredo Rial. *A Prolog Program for Matching Attribute-Based Credentials to Access Control Policies* (2015) (cited on page 193).

[CY10] Yuan Cao and Lin Yang. *A survey of Identity Management technology*. 2010 IEEE International Conference on Information Theory and Information Security. 2010, pages 287–293. doi:10.1109/ICITIS.2010.5689468 (cited on page 16).

[CEF21] CEF eSignature. *MRA Cook-book*. Technical report. European Commission: DIGIT, 2021 (cited on pages 76, 93).

[Cha05] Patrick Chappatte. *Internet And Censorship*. International Herald Tribune. 2005 (cited on page 23).

[CBC21] Panagiotis Chatzigiannis, Foteini Baldimtsi, and Konstantinos Chalkias. *SoK: Blockchain Light Clients*. IACR Cryptol. ePrint Arch. (2021), page 1657 (cited on page 226).

[Cha81] David Chaum. *Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms*. Commun. ACM 24.2 (1981), pages 84–88 (cited on page 184).

[Cha85] David Chaum. *Security Without Identification: Transaction Systems to Make Big Brother Obsolete*. Commun. ACM 28.10 (1985), pages 1030–1044 (cited on page 184).

[CH96] Bruce Christianson and William S. Harbison. *Why Isn't Trust Transitive?* Security Protocols Workshop. Volume 1189. LNCS. Springer, 1996, pages 171–176 (cited on page 91).

[Chu+18] Taejoong Chung, Jay Lok, Balakrishnan Chandrasekaran, David R. Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, John P. Rula, Nick Sullivan, and Christo Wilson. *Is the Web Ready for OCSP Must-Staple?* Internet Measurement Conference. ACM, 2018, pages 105–118 (cited on page 214).

[CO08] Juri Luca De Coi and Daniel Olmedilla. *A Review of Trust Management, Security and Privacy Policy Languages*. SECRYPT. INSTICC Press, 2008, pages 483–490 (cited on pages 105–111, 130, 186).

[Coo+08] David Cooper, Stefan Santesson, Stephen Farrell, Sharon Boeyen, Russell Housley, and W. Timothy Polk. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280 (2008), pages 1–151 (cited on pages 19–20, 33, 67, 153).

[CCL08] Edward Curry, Desmond Chambers, and Gerard Lyons. *Extending Message-Oriented Middleware using Interception*. Third International Workshop on Distributed Event-Based Systems (DEBS '04) at ICSE '04 (Jul 2008), pages 32–37 (cited on page 158).

[Dam+01] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. *The Ponder Policy Specification Language*. POLICY. Volume 1995. LNCS. Springer, 2001, pages 18–38 (cited on page 110).

[Dav+19] Matthew Davie, Dan Gisolfi, Daniel Hardman, John Jordan, Darrell O'Donnell, and Drummond Reed. *The Trust over IP Stack*. IEEE Commun. Stand. Mag. 3.4 (2019), pages 46–51 (cited on page 93).

[DEC96] Pierre Deransart, AbdelAli Ed-Dbali, and Laurent Cervoni. *Prolog - the standard: reference manual*. Springer, 1996 (cited on pages 114, 117).

[Dou+19] Konstantinos Douloudis, Maria Siapera, Gerasimos Dimitriou, and Andriana Prentza. *Application of Automated Trust Verification and Delegation Mechanisms in PEPPOL eProcurement Network*. EMCIS. Volume 381. Lecture Notes in Business Information Processing. Springer, 2019, pages 448–457 (cited on page 85).

[Dri+19] Manu Drijvers, Sergey Gorbunov, Gregory Neven, and Hoeteck Wee. *Pixel: Multi-signatures for Consensus*. IACR Cryptol. ePrint Arch. (2019), page 514 (cited on page 39).

[Dur+13] Zakir Durumeric, James Kasten, Michael Bailey, and J. Alex Halderman. *Analysis of the HTTPS certificate ecosystem*. Internet Measurement Conference. ACM, 2013, pages 291–304 (cited on page 92).

[DC03] Jeffrey H. Dyer and Wujin Chu. *The Role of Trustworthiness in Reducing Transaction Costs and Improving Performance: Empirical Evidence from the United States, Japan, and Korea*. Organization Science 14.1 (2003), pages 57–68. ISSN 10477039, 15265455. http://www.jstor.org/stable/3086033 (cited on page 12).

[ET18] Jacob Eberhardt and Stefan Tai. *ZoKrates - Scalable Privacy-Preserving Off-Chain Computations*. iThings/GreenCom/CPSCom/SmartData. IEEE, 2018, pages 1084–1091 (cited on pages 40, 197).

[Env22] Enveil. *Enveil: ZeroReveal® Machine Learning*. 2022. https://www.enveil.com/products/%5C#zeroreveal-machine-learning (cited on pages 136, 138).

[ETS16] ETSI. *TS 119 612 V2.2.1: Requirements for Trusted Lists*. Standard. European Telecommunications Standards Institute, 2016 (cited on pages 32, 93).

[ETS20] ETSI. *TR 103 684 V1.1.1: Electronic Signatures and Infrastructures (ESI); Global Acceptance of EU Trust Services*. Report. European Telecommunications Standards Institute, 2020 (cited on page 93).

[Eur22] European Commission. *EBSI: European Blockchain Services Infrastructure*. https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/EBSI. online, accessed on 2022-03-16. 2022 (cited on pages 220, 229).

[Eur16] European Parliament. *Regulation (EU) 2016/679 of the European Parliament and of the Council: General Data Protection*

Regulation. 27 Apr 2016. https://eur-lex.europa.eu/eli/reg/2016/679 (cited on pages 25, 135–136).

[Eur14] European Parliament and Council of European Union. *Regulation (EU) No 910/2014 on electronic identification and trust services for electronic transactions in the internal market ("eIDAS")*. https://eur-lex.europa.eu/eli/reg/2014/910/oj. 2014 (cited on pages 19, 42, 45, 62, 88–89, 92–93).

[Fri20] Lothar Fritsch. *Identity Management as a target in cyberwar*. Open Identity Summit. Volume P-305. LNI. Gesellschaft für Informatik e.V., 2020, pages 61–70 (cited on page 33).

[Fuc18] Georg Fuchsbauer. *Subversion-Zero-Knowledge SNARKs*. PKC (1). Volume 10769. LNCS. Springer, 2018, pages 315–347 (cited on page 210).

[FHS19] Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. *Structure-Preserving Signatures on Equivalence Classes and Constant-Size Anonymous Credentials*. J. Cryptol. 32.2 (2019), pages 498–546 (cited on pages 190, 207).

[Fut14] FutureID Consortium. *FutureID D44.3 Technical specification of interfaces, modules and documentation for the application integration services*. https://web.archive.org/web/20180831171146/http://futureid.eu/downloads. online, accessed on 14 July 2023. 2014 (cited on pages 46, 160).

[Fut17] FutureTrust Consortium. *FutureTrust D4.2 Global Trust List - Software architecture document*. https://cordis.europa.eu/project/id/700542/results. online, accessed on 20 June 2023. 2017 (cited on page 93).

[Fut19a] FutureTrust Consortium. *Global Trust Service List*. https://pilots.futuretrust.eu/gtsl. online, accessed on 22 January 2021. 2019 (cited on page 93).

[Fut19b] FutureTrust Consortium. *pan-European eID-Broker*. https://pilots.futuretrust.eu/eid. online, accessed on 16 May 2023. 2019 (cited on page 160).

[Gab+22] Silvia Gabrielli, Silvia Rizzi, Oscar Mayora, Stefan More, Juan Carlos Pérez Baun, and Wim Vandevelde. *Multidimensional Study on Users' Evaluation of the KRAKEN Personal Data Sharing Platform*. Applied Sciences 12.7 (2022). ISSN 2076-3417. doi:10.3390/app12073270 (cited on page 8).

[GL23] Tarek Galal and Anja Lehmann. *Privacy-Preserving Outsourced Certificate Validation*. Proc. Priv. Enhancing Technol. 2023.4 (2023), pages 322–340 (cited on pages 44, 92–93).

[Gal+11] Laura Galluccio, Alessandro Leonardi, Giacomo Morabito, and Sergio Palazzo. *Context Privacy in the Internet of Things*. In: *Trustworthy Internet*. Edited by Luca Salgarelli, Giuseppe Bianchi, and Nicola Blefari-Melazzi. Milano: Springer Milan, 2011, pages 61–73. ISBN 978-88-470-1818-1. doi:10.1007/978-88-470-1818-1_5. https://doi.org/10.1007/978-88-470-1818-1_5 (cited on pages 27, 30).

[Gas+89] Morrie Gasser, Andy Goldstein, Charlie Kaufman, and Butler Lampson. *The Digital Distributed System Security Architecture*. 12th National Computer Security Conference. NIST/NCSC. 1989, pages 305–319. https://www.microsoft.com/en-us/research/publication/the-digital-distributed-system-security-architecture/ (cited on page 72).

[Gav+04] Rita Gavriloaie, Wolfgang Nejdl, Daniel Olmedilla, Kent E. Seamons, and Marianne Winslett. *No Registration Needed: How to Use Declarative Policies and Negotiation to Access Sensitive Resources on the Semantic Web*. ESWS. Volume 3053. LNCS. Springer, 2004, pages 342–356 (cited on page 110).

[Gla23] Mathieu Glaude. *The Digital Universal Credential Adaptor*. May 2023. https://northernblock.io/blog/the-digital-universal-credential-adaptor/ (cited on page 158).

[GB08] Shafi Goldwasser and Mihir Bellare. *Lecture notes on cryptography*. Course "Cryptography and computer security" at MIT (2008) (cited on page 38).

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. *The Knowledge Complexity of Interactive Proof-Systems (Extended Abstract)*. STOC. ACM, 1985, pages 291–304 (cited on page 40).

[Gös06] Stefan Gössner. *Transforming JSON*. https://goessner.net/articles/jsont. online, accessed on 22 January 2021. 2006 (cited on page 172).

[Gös07] Stefan Gössner. *JSONPath - XPath for JSON*. https://goessner.net/articles/JsonPath. online, accessed on 22 January 2021. 2007 (cited on page 172).

[Gra+21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. *Poseidon: A New*

Hash Function for Zero-Knowledge Proof Systems. USENIX. USENIX Association, 2021, pages 519–535 (cited on pages 202, 204).

[GJ19a] Hans Graux and Edwin Jacobs. *LIGHTest D3.7 Cross-Border Legal Compliance and Validity of Trust Scheme Publication*. https://www.lightest.eu/static/deliverables/D3.7.pdf. online, accessed on 17 February 2023. LIGHTest Consortium, 2019 (cited on page 90).

[GJ19b] Hans Graux and Edwin Jacobs. *LIGHTest D6.8 Cross-Border Legal Compliance and Validity of Trust Policy and Trust Decisions*. https://www.lightest.eu/static/deliverables/D6.8.pdf. online, accessed on 17 February 2023. LIGHTest Consortium, 2019 (cited on page 90).

[Gro16] Jens Groth. *On the Size of Pairing-Based Non-interactive Arguments*. EUROCRYPT (2). Volume 9666. LNCS. Springer, 2016, pages 305–326 (cited on pages 40, 204).

[Gro+18] Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. *Updatable and Universal Common Reference Strings with Applications to zk-SNARKs*. CRYPTO (3). Volume 10993. LNCS. Springer, 2018, pages 698–728 (cited on pages 40, 210).

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. *Perfect Non-interactive Zero Knowledge for NP*. EUROCRYPT. Volume 4004. LNCS. Springer, 2006, pages 339–358 (cited on page 210).

[Gru+18] Daniel Gruss, Michael Schwarz, Matthias Wübbeling, Simon Guggi, Timo Malderle, Stefan More, and Moritz Lipp. *Use-After-FreeMail: Generalizing the Use-After-Free Problem and Applying it to Email Services*. AsiaCCS. ACM, 2018, pages 297–311 (cited on page 8).

[Gud+20] Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. *SoK: Layer-Two Blockchain Protocols*. Financial Cryptography. Volume 12059. LNCS. Springer, 2020, pages 201–226 (cited on page 225).

[Gus11] John L. Gustafson. *Moore's Law*. In: *Encyclopedia of Parallel Computing*. Edited by David Padua. Boston, MA: Springer US, 2011, pages 1177–1184. ISBN 978-0-387-09766-4. doi:10.1007/978-0-387-09766-4_81. https://doi.org/10.1007/978-0-387-09766-4_81 (cited on page 144).

[Hal20] Harry Halpin. *Vision: A Critique of Immunity Passports and W3C Decentralized Identifiers*. SSR. Volume 12529. LNCS. Springer, 2020, pages 148–168 (cited on page 33).

[Haw12] Katherine Hawley. *Trust: A Very Short Introduction*. Oxford University Press, Aug 2012. ISBN 9780199697342. doi:10.1093/actrade/9780199697342.001.0001. https://doi.org/10.1093/actrade/9780199697342.001.0001 (cited on pages 11–12, 15).

[Her+00] Amir Herzberg, Yosi Mass, Joris Mihaeli, Dalit Naor, and Yiftach Ravid. *Access Control Meets Public Key Infrastructure, Or: Assigning Roles to Strangers*. IEEE S&P. IEEE, 2000, pages 2–14 (cited on page 110).

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

[Hou+99] Russell Housley, Warwick Ford, W. Timothy Polk, and David Solo. *Internet X.509 Public Key Infrastructure Certificate and CRL Profile*. RFC 2459 (1999), pages 1–129 (cited on pages 90, 92, 153).

[Hüh+16] Detlef Hühnlein, Tilman Frosch, Jörg Schwenk, Carl-Markus Piswanger, Marc Sel, Tina Hühnlein, Tobias Wich, Daniel Nemmert, René Lottes, Juraj Somorovsky, Vladislav Mladenov, Cristina Condovici, Herbert Leitold, Sophie Stalla-Bourdillon, Niko Tsakalakis, Jan Eichholz, Frank-Michael Kamm, Andreas Kühne, Damian Wabisch, Roger Dean, Jon Shamah, Mikheil Kapanadze, Nuno Ponte, Jose Martins, Renato Portela, Cagatay Karabat, Snezana Stojicic, Slobodan Nedeljkovic, Vincent Bouckaert, Alexandre Defays, Bruce Anderson, Michael Jonas, Christina Hermanns, Thomas Schubert, Dirk Wegener, and Alexander Sazonov. *FutureTrust - Future Trust Services for Trustworthy Global Transactions*. Open Identity Summit. Volume P-264. LNI. GI, 2016, pages 27–41 (cited on page 93).

[HC09] Patrick C. K. Hung and Vivying S. Y. Cheng. *Privacy*. In: *Encyclopedia of Database Systems*. Edited by LING LIU and M. TAMER ÖZSU. Boston, MA: Springer US, 2009, pages 2136–2137. ISBN 978-0-387-39940-9. doi:10.1007/978-0-387-39940-9_274. https://doi.org/10.1007/978-0-387-39940-9_274 (cited on pages 24, 26).

[IANa] IANA. *Delegating or transferring a country-code top-level domain (ccTLD)*. https://www.iana.org/help/cctld-delegation (cited on page 89).

[IANb] IANA. *Root Zone Management*. https://www.iana.org/domains/root (cited on page 89).

[III11] Donald E. Eastlake III. *Transport Layer Security (TLS) Extensions: Extension Definitions*. RFC 6066 (2011), pages 1–25 (cited on pages 214, 226).

[Inm16] Matthew Inman. *Trust*. The Oatmeal. 2016. https://theoatmeal.com/comics/trust (cited on page 11).

[ISO20] ISO. *ISO/IEC 9594-8:2020: The Directory: Public-key and attribute certificate frameworks*. Standard. Geneva, CH: International Organization for Standardization, 2020 (cited on page 32).

[ISO95] ISO. *Information technology – Programming languages — Prolog*. Standard. Geneva, CH: International Organization for Standardization, 1995 (cited on pages 37, 112, 114).

[ITU88] ITU. *X.509: The Directory: Public-key and attribute certificate frameworks*. Recommendation. International Telecommunication Union, 1988 (cited on page 32).

[ITU02] ITU. *X.690: ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*. Recommendation. International Telecommunication Union, 2002 (cited on page 153).

[ITU05] ITU. *The root zone file and the root server system*. Jun 2005. https://www.itu.int/itunews/manager/display.asp?lang=en&year=2005&issue=06&ipage=root&ext=html (cited on page 89).

[ITU09] ITU. *Y.2720: NGN identity management framework*. Recommendation. International Telecommunication Union, 2009 (cited on page 13).

[ITU13] ITU. *X.1255 : Framework for discovery of identity management information*. Recommendation. International Telecommunication Union, 2013 (cited on page 95).

[JJ18] Simon Jentzsch and Christoph Jentzsch. *EIP-1186: RPC-Method to get Merkle Proofs - eth_getProof*. https://eips.ethereum.org/EIPS/eip-1186. online, accessed on 22 April 2022. 2018 (cited on page 221).

[JBS15] Michael B. Jones, John Bradley, and Nat Sakimura. *JSON Web Token (JWT)*. RFC 7519 (2015), pages 1–30 (cited on page 153).

[JIB07] Audun Jøsang, Roslan Ismail, and Colin Boyd. *A survey of trust and reputation systems for online service provision.* Decis. Support Syst. 43.2 (2007), pages 618–644 (cited on page 15).

[JKD05] Audun Jøsang, Claudia Keser, and Theodosis Dimitrakos. *Can We Manage Trust?* iTrust. Volume 3477. LNCS. Springer, 2005, pages 93–107 (cited on pages 12, 15).

[KFJ03] Lalana Kagal, Timothy W. Finin, and Anupam Joshi. *A Policy Language for a Pervasive Computing Environment.* POLICY. IEEE, 2003, page 63 (cited on page 110).

[KC88] Robert Kahn and Vinton G. Cerf. *An open architecture for a digital library system and a plan for its development.* Internet Whitepaper `http://hdl.handle.net/4263537/2091` (1988) (cited on page 94).

[KW06] Robert E. Kahn and Robert Wilensky. *A framework for distributed digital object services.* Int. J. Digit. Libr. 6.2 (2006), pages 115–123 (cited on page 94).

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition.* CRC Press, 2014 (cited on page 38).

[Kim19] Pyung Soo Kim. *Comparison and Analysis of DNS and DOA for Internet of Things Naming System.* ICAIIC. IEEE, 2019, pages 552–556 (cited on page 94).

[Koa+] Chong-Gee Koa, Swee-Huay Heng, Syh-Yuan Tan, and Ji-Jian Chin. *Review of Blockchain-Based Public Key Infrastructure.* Cryptology and Information Security Conference 2020, page 20 (cited on page 34).

[Koc+22] Karl Koch, Stephan Krenn, Tilen Marc, Stefan More, and Sebastian Ramacher. *KRAKEN: a privacy-preserving data market for authentic data.* DE@CoNEXT. ACM, 2022, pages 15–20 (cited on pages 8, 135–136).

[Koc+20] Karl Koch, Stephan Krenn, Donato Pellegrino, and Sebastian Ramacher. *Privacy-Preserving Analytics for Data Markets Using MPC.* Privacy and Identity Management. Volume 619. IFIP Advances in Information and Communication Technology. Springer, 2020, pages 226–246 (cited on pages 136, 138, 144).

[KPS18] Ahmed E. Kosba, Charalampos Papamanthou, and Elaine Shi. *xJsnark: A Framework for Efficient Verifiable Computation.* IEEE S&P. IEEE, 2018, pages 944–961 (cited on page 40).

[Kou+20] Vlasis Koutsos, Dimitrios Papadopoulos, Dimitris Chatzopoulos, Sasu Tarkoma, and Pan Hui. *Agora: A Privacy-aware Data Marketplace*. ICDCS. IEEE, 2020, pages 1211–1212 (cited on pages 136, 146).

[KRA22] KRAKEN Consortium. *KRAKEN D4.3 Prototype implementation of cryptographic libraries*. https://krakenh2020.eu/node/154. not online; results provided to us in private by the authors. 2022 (cited on page 144).

[Kyi+23] Lin Kyi, Sushil Ammanaghatta Shivakumar, Cristiana Teixeira Santos, Franziska Roesner, Frederike Zufall, and Asia J. Biega. *Investigating Deceptive Design in GDPR's Legitimate Interest*. CHI. ACM, 2023, 583:1–583:16 (cited on page 135).

[Lan+17] David Lane, Christos Vontas, Thomas Rückstieß, and David Poggi. *jsonpath-object-transform*. https://github.com/dvdln/jsonpath-object-transform. online, accessed on 22 January 2021. 2017 (cited on pages 171–172).

[Lau+08] Ben Laurie, Geoffrey Sisson, Roy Arends, and David Blacka. *DNS Security (DNSSEC) Hashed Authenticated Denial of Existence*. RFC 5155 (2008), pages 1–52 (cited on pages 37, 78).

[Lee11] Adam J. Lee. *Credential-Based Access Control*. In: *Encyclopedia of Cryptography and Security*. Edited by Henk C. A. van Tilborg and Sushil Jajodia. Boston, MA: Springer US, 2011, pages 271–272. ISBN 978-1-4419-5906-5. doi:10.1007/978-1-4419-5906-5_898. https://doi.org/10.1007/978-1-4419-5906-5_898 (cited on page 183).

[Li+21] Kai Li, Jiaqi Chen, Xianghong Liu, Yuzhe Richard Tang, XiaoFeng Wang, and Xiapu Luo. *As Strong As Its Weakest Link: How to Break Blockchain DApps at RPC Service*. NDSS. The Internet Society, 2021 (cited on pages 213, 227).

[LM03] Ninghui Li and John C. Mitchell. *A Role-based Trust-management Framework*. DISCEX (1). IEEE, 2003, page 201 (cited on page 110).

[LWM01] Ninghui Li, William H. Winsborough, and John C. Mitchell. *Distributed credential chain discovery in trust management: extended abstract*. CCS. ACM, 2001, pages 156–165 (cited on page 133).

[Lor+03] Markus Lorch, Seth Proctor, Rebekah Lepro, Dennis G. Kafura, and Sumit Shah. *First experiences using XACML for*

access control in distributed systems. XML Security. ACM, 2003, pages 25–37 (cited on page 110).

[LR09] Wenjing Lou and Kui Ren. *Security, privacy, and accountability in wireless access networks*. IEEE Wirel. Commun. 16.4 (2009), pages 80–87 (cited on page 29).

[Lys02] Anna Lysyanskaya. *Signature schemes and applications to cryptographic protocol design*. PhD Thesis. Massachusetts Institute of Technology, Cambridge, MA, USA, 2002 (cited on page 38).

[Mai+18] David Maier, K. Tuncay Tekle, Michael Kifer, and David Scott Warren. *Datalog: concepts, history, and outlook*. In: *Declarative Logic Programming*. ACM / Morgan & Claypool, 2018, pages 3–100 (cited on page 110).

[MC13] Viktor Mayer-Schönberger and Kenneth Cukier. *Big data: A revolution that will transform how we live, work, and think*. Houghton Mifflin Harcourt, 2013 (cited on page 27).

[MM02] Petar Maymounkov and David Mazières. *Kademlia: A Peer-to-Peer Information System Based on the XOR Metric*. IPTPS. Volume 2429. LNCS. Springer, 2002, pages 53–65 (cited on page 229).

[Mea02] Michael Mealling. *Dynamic Delegation Discovery System (DDDS) Part One: The Comprehensive DDDS*. RFC 3401 (2002), pages 1–6 (cited on page 91).

[MD00] Michael Mealling and Ron Daniel. *The Naming Authority Pointer (NAPTR) DNS Resource Record*. RFC 2915 (2000), pages 1–18 (cited on page 91).

[Mera] Merriam-Webster. *Identity*. In: *Merriam-Webster.com dictionary*. https://www.merriam-webster.com/dictionary/identity (cited on page 13).

[Merb] Merriam-Webster. *Security*. In: *Merriam-Webster.com dictionary*. https://www.merriam-webster.com/dictionary/security (cited on page 14).

[Mik+15] Gert Læssøe Mikkelsen, Kasper Damgård, Hans Guldager, Jonas Lindstrøm Jensen, Jesus Luna Garcia, Janus Dam Nielsen, Pascal Paillier, Giancarlo Pellegrino, Michael Bladt Stausholm, Neeraj Suri, and Heng Zhang. *Technical Implementation and Feasibility*. In: *Attribute-based Credentials for Trust*. Springer, 2015, pages 255–317 (cited on page 211).

[Moc87a] Paul V. Mockapetris. *Domain names - concepts and facilities*. RFC 1034 (1987), pages 1–55 (cited on page 36).

[Moc87b] Paul V. Mockapetris. *Domain names - implementation and specification*. RFC 1035 (1987), pages 1–55 (cited on pages 36, 67).

[MK14] Sebastian Mödersheim and Georgios Katsoris. *A Sound Abstraction of the Parsing Problem*. CSF. IEEE, 2014, pages 259–273 (cited on page 118).

[MN19] Sebastian Mödersheim and Bihang Ni. *GTPL: A Graphical Trust Policy Language*. Open Identity Summit. Volume P-293. LNI. GI, 2019, pages 107–118 (cited on page 134).

[Möd+19] Sebastian Mödersheim, Anders Schlichtkrull, Georg Wagner, Stefan More, and Lukas Alber. *TPL: A Trust Policy Language*. IFIPTM. Volume 563. IFIP Advances in Information and Communication Technology. Springer, 2019, pages 209–223 (cited on pages 7, 94, 103, 121, 147, 262).

[Mor15] Stefan More. *Java Privacy Guard - The OpenPGP Message Format and an Implementation in Java*. 2015. doi:10.5281/zenodo.31419. http://dx.doi.org/10.5281/zenodo.31419 (cited on page 154).

[Mor23] Stefan More. *Trust Scheme Interoperability: Connecting Heterogeneous Trust Schemes*. ARES. ACM, 2023, 124:1–124:9 (cited on pages 7, 57, 102, 261).

[MA22] Stefan More and Lukas Alber. *YOU SHALL NOT COMPUTE on my Data: Access Policies for Privacy-Preserving Data Marketplaces and an Implementation for a Distributed Market using MPC*. ARES. ACM, 2022, 137:1–137:8 (cited on pages 7, 135, 147, 204, 263).

[Mor+21] Stefan More, Peter Grassberger, Felix Hörandner, Andreas Abraham, and Lukas Daniel Klausner. *Trust Me If You Can: Trusted Transformation Between (JSON) Schemas to Support Global Authentication of Education Credentials*. SEC. Volume 625. IFIP Advances in Information and Communication Technology. Springer, 2021, pages 19–35 (cited on pages 7, 95, 102, 149, 170, 177, 262).

[MHW22] Stefan More, Jakob Heher, and Clemens Walluschek. *Offline-verifiable Data from Distributed Ledger-based Registries*. SECRYPT. SCITEPRESS, 2022, pages 687–693 (cited on pages 7, 213, 230, 263).

[Mor+22] Stefan More, Sebastian Ramacher, Lukas Alber, and Marco Herzl. *Extending Expressive Access Policies with Privacy*

Features. TrustCom. IEEE, 2022, pages 574–581 (cited on pages 7, 183, 211, 263).

[Müh+18] Alexander Mühle, Andreas Grüner, Tatiana Gayvoronskaya, and Christoph Meinel. *A survey on essential components of a self-sovereign identity*. Comput. Sci. Rev. 30 (2018), pages 80–86 (cited on pages 33, 184).

[Mun13] Randall Munroe. *Privacy Opinions*. xkcd 1269. enhanced using www.upscale.media. 2013 (cited on page 28).

[Nak08] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. Technical report. 2008 (cited on page 35).

[NOW04] Wolfgang Nejdl, Daniel Olmedilla, and Marianne Winslett. *PeerTrust: Automated Trust Negotiation for Peers on the Semantic Web*. Secure Data Management. Volume 3178. LNCS. Springer, 2004, pages 118–132 (cited on page 110).

[Nie97] Jakob Nielsen. *Usability Engineering*. In: *The Computer Science and Engineering Handbook*. CRC Press, 1997, pages 1440–1460 (cited on pages 188, 208, 225).

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*. Volume 2283. LNCS. Springer, 2002 (cited on page 134).

[Omo+19] Olamide Omolola, Stefan More, Edona Fasllija, Georg Wagner, and Lukas Alber. *Policy-based Access Control for the IoT and Smart Cities*. Open Identity Summit. Volume P-293. LNI. GI, 2019, pages 157–163 (cited on pages 8, 85).

[OBK23] Alejandra Gómez Ortega, Jacky Bourgeois, and Gerd Kortuem. *What is Sensitive About (Sensitive) Data? Characterizing Sensitivity and Intimacy with Google Assistant Users*. CHI. ACM, 2023, 586:1–586:16 (cited on page 24).

[OM23] Sarah Otto and Michael Meisel. *X out of N Credential Requests using Presentation Exchange*. Open Identity Summit. Volume P-335. LNI. Gesellschaft für Informatik e.V., 2023 (cited on page 192).

[Oxf] Oxford University Press. *Infosec*. In: *Exford Learner's Dictionaries*. https://www.oxfordlearnersdictionaries.com/definition/english/infosec (cited on page 15).

[Ped91] Torben P. Pedersen. *Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing*. CRYPTO. Volume 576. LNCS. Springer, 1991, pages 129–140 (cited on page 207).

[Per99] Radia J. Perlman. *An overview of PKI trust models*. IEEE Netw. 13.6 (1999), pages 38–43 (cited on pages 13, 18–19, 37, 72, 88).

[PH10] Andreas Pfitzmann and Marit Hansen. *A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management*. v0.34. 2010 (cited on pages 13, 28–29).

[PAZ22] Blaz Podgorelec, Lukas Alber, and Thomas Zefferer. *What is a (Digital) Identity Wallet? A Systematic Literature Review*. COMPSAC. IEEE, 2022, pages 809–818 (cited on page 191).

[PS16] David Pointcheval and Olivier Sanders. *Short Randomizable Signatures*. CT-RSA. Volume 9610. LNCS. Springer, 2016, pages 111–126 (cited on page 207).

[Pro21] Protocol Labs. *IPFS Documentation*. `https://docs.ipfs.io`. online, accessed on 22 January 2021. 2021 (cited on page 36).

[Prü16] Bernd Prünster. *Gríð – Secure Information Exchange Based on an Inverted Trust Model*. English. Masters thesis. 2016 (cited on page 19).

[RS13] Stefan Rass and Daniel Slamanig. Artech, 2013. ISBN 9781608075768 (cited on page 26).

[Ree+21] Drummond Reed, Manu Sporny, Dave Longley, Christopher Allen, Ryan Grant, and Markus Sabadello. *Decentralized Identifiers (DIDs) v1.0*. W3C Working Draft. W3C, 20 Jan 2021. `https://www.w3.org/TR/2021/WD-did-core-20210128/` (cited on pages 33, 142).

[Res+00] Paul Resnick, Ko Kuwabara, Richard Zeckhauser, and Eric Friedman. *Reputation systems*. Commun. ACM 43.12 (2000), pages 45–48 (cited on page 15).

[Rod+19] Michael Rodler, Wenting Li, Ghassan O. Karame, and Lucas Davi. *Sereum: Protecting Existing Smart Contracts Against Re-Entrancy Attacks*. NDSS. The Internet Society, 2019 (cited on page 101).

[Rod+21] Michael Rodler, Wenting Li, Ghassan O. Karame, and Lucas Davi. *EVMPatch: Timely and Automated Patching of Ethereum Smart Contracts*. USENIX. USENIX Association, 2021, pages 1289–1306 (cited on page 101).

[Roß17] Heiko Roßnagel. *A Mechanism for Discovery and Verification of Trust Scheme Memberships: The Lightest Reference Architecture*. Open Identity Summit. Volume P-277. LNI.

Gesellschaft für Informatik, Bonn, 2017, pages 81–92 (cited on pages 43, 94, 261–262).

[SKR12] Ahmad Sabouri, Ioannis Krontiris, and Kai Rannenberg. *Attribute-Based Credentials for Trust (ABC4Trust)*. TrustBus. Volume 7449. LNCS. Springer, 2012, pages 218–219 (cited on page 188).

[SBT19] Anders Schlichtkrull, Jasmin Christian Blanchette, and Dmitriy Traytel. *A verified prover based on ordered resolution*. CPP. ACM, 2019, pages 152–165 (cited on page 134).

[Sch+18] Anders Schlichtkrull, Jasmin Christian Blanchette, Dmitriy Traytel, and Uwe Waldmann. *Formalization of Bachmair and Ganzinger's Ordered Resolution Prover*. Arch. Formal Proofs 2018 (2018) (cited on page 134).

[SM20] Anders Schlichtkrull and Sebastian Mödersheim. *Accountable Trust Decisions: A Semantic Approach*. Open Identity Summit. Volume P-305. LNI. Gesellschaft für Informatik e.V., 2020, pages 71–82 (cited on page 134).

[Sea+02] Kent E. Seamons, Marianne Winslett, Ting Yu, Bryan Smith, Evan Child, Jared Jacobson, Hyrum Mills, and Lina Yu. *Requirements for Policy Languages for Trust Negotiation*. POLICY. IEEE, 2002, pages 68–79 (cited on pages 105, 107, 109, 130–131, 186).

[Sel+19] Rachelle Sellung, Fabina Dietrich, Heiko Roßnagel, Bud Bruegger, Sue Dawes, Charo Encinas Bayán, Stefan More, Georg Wagner, Peter Lipp, Alberto Crespo Garcia, Jan Camienisch, and Sebastian Alexander Mödersheim. *LIGHTest D2.2: Inventories*. `https://www.lightest.eu/static/deliverables/D2.2.pdf`. online, accessed on 27 March 2023. LIGHTest Consortium, 2019 (cited on pages 93, 160).

[ST05] Vitaly Shmatikov and Carolyn L. Talcott. *Reputation-based trust management*. J. Comput. Secur. 13.1 (2005), pages 167–190 (cited on page 15).

[SM95] Gustavus J. Simmons and Catherine A. Meadows. *The Role of Trust in Information Integrity Protocols*. J. Comput. Secur. 3.1 (1995), pages 71–84 (cited on page 15).

[SWS07] Anoop Singhal, Theodore Winograd, and Karen A. Scarfone. *SP 800-95. Guide to Secure Web Services*. Technical report. Gaithersburg, MD, USA, 2007 (cited on page 183).

[Sol11] D.J. Solove. *Nothing to Hide: The False Tradeoff Between Privacy and Security*. Yale University Press, 2011. ISBN 9780300172331 (cited on page 26).

[SLC22] Manu Sporny, Dave Longley, and David Chadwick. *Verifiable Credentials Data Model 1.1*. W3C Recommendation. W3C, 03 Mar 2022. https://www.w3.org/TR/2022/REC-vc-data-model-20220303/ (cited on pages 20, 33–34, 140, 158, 164, 184, 188–189, 269–270).

[SSP23] Manu Sporny, Orie Steele, and Michael Prorock. *DID Specification Registries*. W3C Note. W3C, Mar 2023. https://www.w3.org/TR/2023/NOTE-did-spec-registries-20230310/ (cited on pages 33, 128).

[SLL13] Klaus Stranacher, Thomas Lenz, and Konrad Lanz. *Trust-Service Status List Based Signature Verification - Opportunities, Implementation and Survey*. EGOVIS/EDEM. Volume 8061. LNCS. Springer, 2013, pages 29–42 (cited on pages 48, 92).

[Tho+17] Kurt Thomas, Frank Li, Ali Zand, Jacob Barrett, Juri Ranieri, Luca Invernizzi, Yarik Markov, Oxana Comanescu, Vijay Eranti, Angelika Moscicki, Daniel Margolis, Vern Paxson, and Elie Bursztein. *Data Breaches, Phishing, or Malware?: Understanding the Risks of Stolen Credentials*. CCS. ACM, 2017, pages 1421–1434 (cited on page 33).

[TLS14] TLSNotary. *TLSnotary - a mechanism for independently audited https sessions*. https://tlsnotary.org/TLSNotary.pdf. 2014 (cited on page 226).

[Tor+20] Christof Ferreira Torres, Mathis Baden, Robert Norvill, Beltran Borja Fiz Pontiveros, Hugo Jonker, and Sjouke Mauw. *ÆGIS: Shielding Vulnerable Smart Contracts Against Attacks*. AsiaCCS. ACM, 2020, pages 584–597 (cited on page 101).

[Tru21] Trust over IP Foundation. *The Good Health Pass Interoperability Blueprint*. Aug 2021. https://trustoverip.org/wp-content/uploads/The-Good-Health-Pass-Interoperability-Blueprint-V1.0-2021-08-01.pdf (cited on page 158).

[Usz+03] Andrzej Uszok, Jeffrey M. Bradshaw, Renia Jeffers, Niranjan Suri, Patrick J. Hayes, Maggie R. Breedy, Larry Bunch, Matt Johnson, Shriniwas Kulkarni, and James Lott. *KAoS Policy and Domain Services: Toward a Description-Logic Approach to Policy Representation, Deconfliction, and Enforcement*. POLICY. IEEE, 2003, page 93 (cited on page 110).

[VT17] Maarten Van Steen and Andrew S Tanenbaum. *Distributed systems*. Maarten van Steen Leiden, The Netherlands, 2017 (cited on page 34).

[Ver22] Verizon. *IP Latency Statistics*. online, accessed on 22 April 2022. 2022. https://enterprise.verizon.com/terms/latency/ (cited on page 224).

[Wac15] Raymond Wacks. *Privacy: A Very Short Introduction*. Oxford University Press, Mar 2015. ISBN 9780198725947. doi:10.1093/actrade/9780198725947.001.0001. https://doi.org/10.1093/actrade/9780198725947.001.0001 (cited on pages 24–25).

[WOM17] Georg Wagner, Olamide Omolola, and Stefan More. *Harmonizing Delegation Data Formats*. Open Identity Summit. Volume P-277. LNI. Gesellschaft für Informatik, Bonn, 2017, pages 25–34 (cited on page 8).

[Wag+19] Georg Wagner, Sven Wagner, Stefan More, and Martin Hoffmann. *DNS-based Trust Scheme Publication and Discovery*. Open Identity Summit. Volume P-293. LNI. GI, 2019, pages 49–58 (cited on pages 7, 31, 57, 94, 102, 261).

[WE18] Isabel Wagner and David Eckhoff. *Technical Privacy Metrics: A Systematic Survey*. ACM Comput. Surv. 51.3 (2018), 57:1–57:38 (cited on page 211).

[WKR19] Sven Wagner, Sebastian Kurowski, and Heiko Roßnagel. *Unified Data Model for Tuple-Based Trust Scheme Publication*. Open Identity Summit. Volume P-293. LNI. GI, 2019, pages 131–142 (cited on pages 43, 60, 76, 90, 93).

[Wal01] Priscilla Walmsley. *Definitive XML schema*. Pearson Education, 2001 (cited on page 153).

[WO19] Stephanie Weinhardt and Olamide Omolola. *Usability of Policy Authoring Tools: A Layered Approach*. ICISSP. SciTePress, 2019, pages 301–308 (cited on page 134).

[WP19] Stephanie Weinhardt and Doreen St. Pierre. *Lessons learned - Conducting a User Experience evaluation of a Trust Policy Authoring Tool*. Open Identity Summit. Volume P-293. LNI. GI, 2019, pages 185–190 (cited on page 134).

[Wes67] Alan F. Westin. *Privacy and Freedom*. Atheneum, 1967 (cited on page 24).

[Woo+22] Gavin Wood et al. *Ethereum: A secure decentralised generalised transaction ledger*. Ethereum project yellow paper, Berlin version (2022). online, accessed on 22 April 2022 (cited on page 221).

[WAH20] Austin Wright, Henry Andrews, and Ben Hutton. *JSON Schema Specification*. https://json-schema.org/specification.html. online, accessed on 22 January 2021. 2020 (cited on page 34).

[Xia+20] Yang Xiao, Ning Zhang, Wenjing Lou, and Y. Thomas Hou. *A Survey of Distributed Consensus Protocols for Blockchain Networks*. IEEE Commun. Surv. Tutorials 22.2 (2020), pages 1432–1465 (cited on page 34).

[Yai19] Reda Yaich. *Trust Management Systems: a Retrospective Study on Digital Trust*. In: *Cyber-Vigilance and Digital Trust*. John Wiley and Sons, Ltd, 2019. Chapter 2, pages 51–103. ISBN 9781119618393. doi:https://doi.org/10.1002/9781119618393.ch2 (cited on pages 109–110).

[YKS19] Shoko Yonezawa, Tetsutaro Kobayashi, and Tsunekazu Saito. *Pairing-Friendly Curves*. Internet-Draft draft-yonezawa-pairing-friendly-curves-02. IETF Secretariat, Jul 2019. http://www.ietf.org/internet-drafts/draft-yonezawa-pairing-friendly-curves-02.txt (cited on page 204).

[You23] Kaliya Young. *Wallets Can't be the Adapters Between Credential Formats*. May 2023. https://identitywoman.net/wallets-cant-be-the-adapters-between-credential-formats/ (cited on pages 158, 176).

[ZM00] Giorgos Zacharia and Pattie Maes. *Trust Management through Reputation Mechanisms*. Appl. Artif. Intell. 14.9 (2000), pages 881–907 (cited on page 15).

[Zam+21] Alexei Zamyatin, Mustafa Al-Bassam, Dionysis Zindros, Eleftherios Kokoris-Kogias, Pedro Moreno-Sanchez, Aggelos Kiayias, and William J. Knottenbelt. *SoK: Communication Across Distributed Ledgers*. Financial Cryptography (2). Volume 12675. LNCS. Springer, 2021, pages 3–36 (cited on page 226).

[Zha+20] Fan Zhang, Deepak Maram, Harjasleen Malvai, Steven Goldfeder, and Ari Juels. *DECO: Liberating Web Data Using Decentralized Oracles for TLS*. CCS. ACM, 2020, pages 1919–1938 (cited on page 226).

[Zhe+18] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. *Blockchain challenges and opportunities: a survey*. Int. J. Web Grid Serv. 14.4 (2018), pages 352–375 (cited on page 34).

[Zim95] Philip R Zimmermann. *The official PGP user's guide*. MIT press, 1995 (cited on page 18).

[ZS18] Yixin Zou and Florian Schaub. *Concern But No Action: Consumers' Reactions to the Equifax Data Breach*. CHI Extended Abstracts. ACM, 2018 (cited on page 33).

Appendix

A

Contribution Statements

The research on the topics of this thesis resulted in the publication of eight peer-reviewed papers accepted at international conferences. The author of this thesis is the lead or one of the main authors of all these publications. Additionally, some of the results are also implemented and evaluated in the course of three Horizon 2020 research and innovation projects (LIGHTest, KRAKEN, Eratosthenes).

Wagner G, Wagner S, More S & Hoffmann M, **DNS-based trust scheme publication and discovery**. in Open Identity Summit 2019 [Wag+19].

- Research Area: Trust
- Contribution Statement: This publication builds on the ideas of the LIGHTest project [BL16], in which the author actively participated. The described component is based on mechanisms introduced in the LIGHTest reference architecture [Roß17], which presents an earlier version of the approach. The author is a co-designer of the architecture and the lead developer of the described automated trust verification system.
- CRediT:[1] Conceptualization, Software, Writing

More S, **Trust Scheme Interoperability: Connecting Heterogeneous Trust Schemes**. in ARES SECPID 2023 [Mor23].

- Research Area: Trust
- Contribution Statement: This publication builds on ideas of the LIGHTest project [BL16], in which the author actively participated.

[1] https://www.elsevier.com/authors/policies-and-guidelines/credit-author-statement

The described component is based on mechanisms introduced in the LIGHTest reference architecture [Roß17], which presents an earlier version of the approach. The author is a co-designer of the architecture and the lead developer of the described automated trust translation system.

- CRediT: Conceptualization, Software, Investigation, Writing, Visualization

---

Mödersheim S, Schlichtkrull A, Wagner G, More S & Alber L, **TPL: A Trust Policy Language**. in IFIPTM 2019 [Möd+19].

- Research Area: Trust
- Contribution Statement: This publication builds on ideas of the LIGHTest project [BL16], in which the author actively participated. Specifically, the author is a co-designer of the described policy system. The author is also the advisor of the student who developed the implementation.
- CRediT: Software, Investigation, Writing

Alber L, More S, Mödersheim S & Schlichtkrull A, **Adapting the TPL Trust Policy Language for a Self-Sovereign Identity World**. in Open Identity Summit 2021 [Alb+21].

- Research Area: Trust
- Contribution Statement: This publication is based on ideas and research of the author. The author is the lead of the project and the lead author of the publication. The author is further a co-designer of the described system and one of the supervisors of the student who helped with the implementation.
- CRediT: Conceptualization, Software, Investigation, Writing, Visualization, Supervision

---

More S, Grassberger P, Hörandner F, Abraham A & Klausner LD, **Trust Me If You Can: Trusted Transformation Between (JSON) Schemas to Support Global Authentication of Education Credentials**. in IFIP SEC 2021 [Mor+21].

- Research Area: Trust
- Contribution Statement: This publication is based on ideas and research of the author. The author is the lead of the project and the lead author of the publication. The author is further the designer of the described system and the supervisor of the students who helped with the implementation.
- CRediT: Conceptualization, Software, Investigation, Writing, Visualization, Supervision

---

More S & Alber L, **YOU SHALL NOT COMPUTE on my Data: Access Policies for Privacy-Preserving Data Marketplaces and an Implementation for a Distributed Market using MPC**. in ARES SECPID 2022 [MA22].

- Research Area: Trust (Use Case)
- Contribution Statement: This publication is based on ideas and research of the author, integrated into ideas of the KRAKEN project. The author is the lead author of the publication.
- CRediT: Conceptualization, Software, Investigation, Writing, Visualization

---

More S, Ramacher S, Alber L & Herzl M, **Extending Expressive Access Policies with Privacy Features**. in TrustCom 2022 [Mor+22].

- Research Area: Privacy
- Contribution Statement: This publication is based on ideas and research of the author. The author is the lead of the project and the lead author of the publication. The author is further the designer of the described system and one of the supervisors of the student who helped with the implementation.
- CRediT: Conceptualization, Investigation, Writing, Visualization, Supervision

---

More S, Heher J & Walluschek C, **Offline-verifiable Data from Distributed Ledger-based Registries**. in SECRYPT 2022 [MHW22].

- Research Area: Privacy
- Contribution Statement: This publication is based on ideas and research of the author. The author is the lead of the project and the lead author of the publication. The author is further the designer of the described system and the supervisor of the student who helped with the implementation.
- CRediT: Conceptualization, Software, Investigation, Writing, Visualization, Supervision

B

TPL EBNF Grammar

In Chapter 7 we introduce our trust policy system TPL. Using W3C's extended Backus–Naur form (EBNF),[1] the syntax of TPL can be described as shown in Listing B.1 and as visualized in Figure B.1.

```
TPLPolicy   ::=   Clause*

Query       ::=   (Predication ',')* Predication '.'

Clause      ::=  Predication '.'
               |  Predication ':-'
                  (Predication ',')* Predication '.'

Predication ::= 'PredicateSymbol'
               | 'PredicateSymbol' '(' (Term ',')* Term ')'


Term        ::= 'VariableSymbol'
               | 'ConstantSymbol'
               | 'FunctionSymbol' '(' (Term ',')* Term ')'
```

Listing B.1.: TPL's grammar in EBNF notation.

[1] https://www.w3.org/TR/REC-xml/#sec-notation

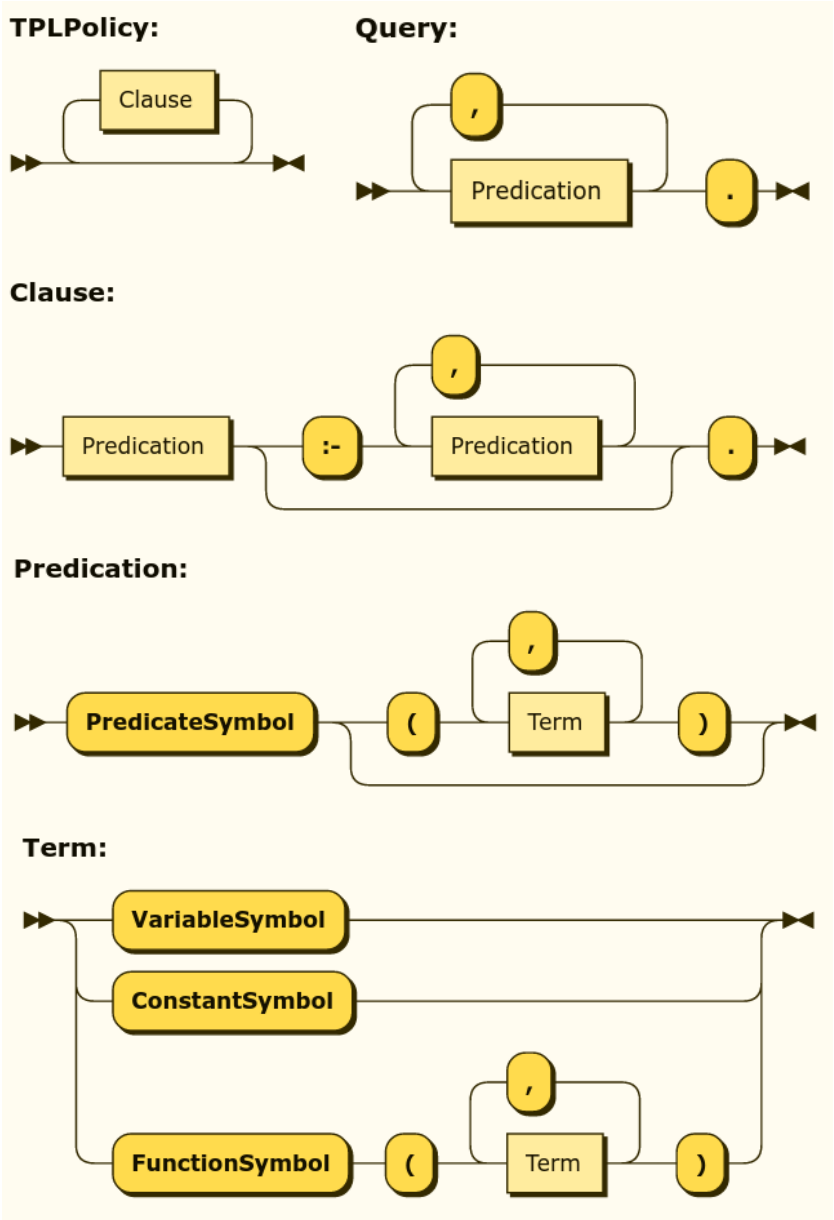

Figure B.1.: Railroad diagram of TPL's EBNF grammar from Listing B.1. Diagram generated using the Railroad Diagram Generator at `https://bottlecaps.de/rr/ui`

C

Example Credential Schema Transformation

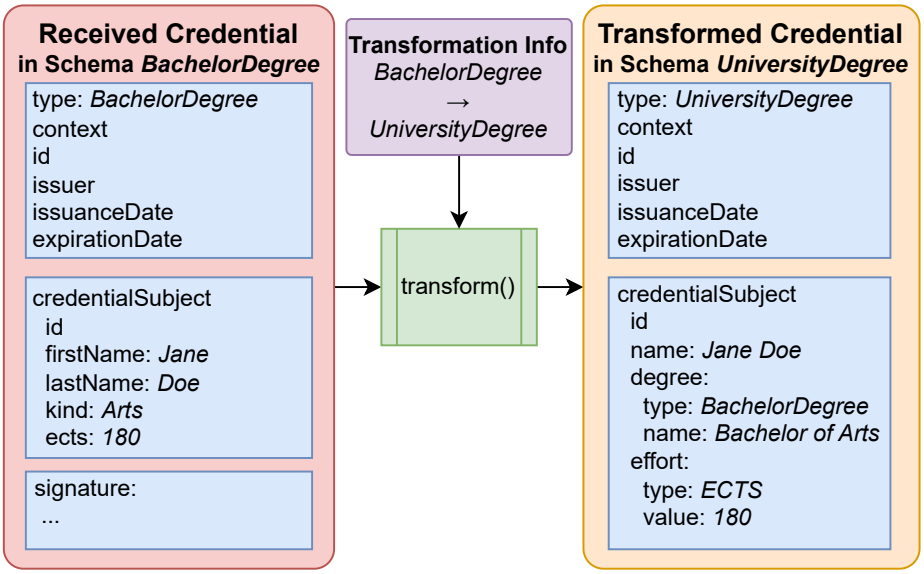

Figure C.1.: Example credential transformation process.

D

Example Credential Schema Transformation Template

Listing D.1 shows an example credential transformation process, as introduced in Chapter 8. In the used `credential`, the JSON-LD `@context` is used to contextualize the credential's attributes. Most importantly, the `type` attribute is mapped from a shortcut to a schema identifier URI (see Section 8.4.1). E.g., the string `VerifiableCredential` is mapped to the `https://www.w3.org/2018/credentials#VerifiableCredential` identifier using the first context. Additionally, the string `AlumniCredential` (representing the schema of `credentialSubject`) is mapped to the identifier `https://example.org/examples#AlumniCredential` using the `examples` context. As an alternative, W3C Verifiable Credentials (VCs) provide the `credentialSchema` property to specific a credential schema and enforce a specific structure on a credential [SLC22, Sections 5.4 and B.2].

```
var jsonpathObjectTransform = require("jsonpath-object-transform")

var template = { "TransformationInformation": { "GraduationDiploma": {
  "PersonDID": "$.credentialSubject.id", "UniversityDID":
  "$.credentialSubject.alumniOf.id", "UniversityName":
  "$.credentialSubject.alumniOf.name[?(@.lang == 'en')].value" } },
  "TransformationSignature": { "type": "RsaSignature2018", "created":
  "2019-01-01T01:00:00Z", "proofPurpose": "assertionMethod",
  "verificationMethod": "did:example:ababb1f712ebc6f1c276e12ec21",
  "jws": "TVkIEq_PbChOMqsLfRoPsnsgw5WEuts01mq...." } };

var credential = { "@context": [
  "https://www.w3.org/2018/credentials/v1",
  "https://www.w3.org/2018/credentials/examples/v1" ], "id":
  "http://example.edu/credentials/1872", "type":
  ["VerifiableCredential", "AlumniCredential"], "issuer":
  "https://example.edu/issuers/565049", "issuanceDate":
  "2020-06-18T19:73:24Z",

  "credentialSubject": { "id":
    "did:example:ebfeb1f712ebc6f1c276e12ec21", "alumniOf": { "id":
    "did:example:c276e12ec21ebfeb1f712ebc6f1", "name": [{ "value":
    "Example University", "lang": "en" }, { "value": "Exemple
    d'Université", "lang": "fr" }] } }, "proof": { "type":
    "RsaSignature2018", "created": "2020-06-18T21:19:10Z",
    "proofPurpose": "assertionMethod", "verificationMethod":
    "https://example.edu/issuers/keys/1", "jws":
    "eyJhbGciOiJSUzI1NiIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0Il19...." }
    };

var result = jsonpathObjectTransform(credential,
template.TransformationInformation);

// result:
 "GraduationDiploma": {
  "PersonDID":"did:example:ebfeb1f712ebc6f1c276e12ec21",
  "UniversityDID":"did:example:c276e12ec21ebfeb1f712ebc6f1",
  "UniversityName":"Example University"
 }
```

Listing D.1.: Example TI, encoded for the *jsonpath-object-transform* transformation engine. This TI transforms a VC into a simple diploma credential. Credential adapted from [SLC22].

That's all folks.

Version dc4a7bd74495f89c117bc699d98118752a74ca46
2023-10-31