Stefan J. More, BSc

TIGHT^{est}

Towards automating global Academic Mobility

Master's Thesis

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Ass.Prof. Dipl.-Ing. Dr.techn. Peter Lipp

Institute of Applied Information Processing and Communications

Faculty of Computer Science and Biomedical Engineering

Graz, April 2018

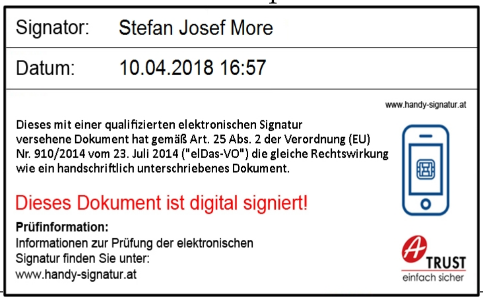
This thesis is set in Palatino,
compiled with `latexrun`, pdfLaTeX2e, and `Biber`.

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

PlantUML was used to generate the diagrams used in this thesis.

Affidavit

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



Signator: Stefan Josef More

Datum: 10.04.2018 16:57

www.handy-signatur.at

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

Dieses Dokument ist digital signiert!

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

TRUST
einfach sicher

Signature

Abstract

Academic mobility is key in a global world. It is possible to graduate from one university in one country and continue studying at a different institution abroad. Doing so follows a procedure established by laws, treaties, and other rules. It often involves a lot of paperwork, translations of documents, certifications, and various authorities. To automate this process, we introduce TIGHT^{est}.

The *Tiny Infrastructure for Global Heterogeneous Trust management in support of an open Ecosystem of Stakeholders and Trust schemes* (TIGHT^{est})[1] defines a trust infrastructure to automate trust decisions arising in global academic mobility. TIGHT^{est} defines a set of software components, formats and protocols.

We show that the trust infrastructure of the Domain Name System (DNS) and its security extensions (DNSSEC) can be used to establish a global trust root for automated trust decisions. We demonstrate that using this trust root and defined protocols, it is possible to locate and query the authorities responsible for authenticating documents. In addition, we show how this can be used to automate the verification of student applications.

Keywords: Trust Management, Academic Mobility, DNSSEC

[1] TIGHT^{est} is based on the LIGHT^{est} project, which is funded by the European Commission as an Innovation Act as part of the Horizon2020 program under grant agreement number 700321.

Kurzfassung

Akademische Mobilität ist ein Schlüsselkonzept in einer globalisierten Welt. Es ist möglich, ein Studium auf einer Universität abzuschließen und anschließend an einer anderen Institution in einem anderen Land weiter zu studieren. Möglich ist dies dank etablierter Abkommen, Gesetze und anderer Regeln. Außerdem sind üblicherweise viele Formulare, Übersetzungen von Dokumenten, Bescheinigungen, Beurkundungen, Zertifizierungen und Institutionen beteiligt. Um diesen Prozess zu automatisieren präsentieren wir TIGHT^{est}.

Das *Tiny Infrastructure for Global Heterogeneous Trust management in support of an open Ecosystem of Stakeholders and Trust schemes* (TIGHT^{est})[2] Projekt stellt eine Infrastruktur bereit, um Vertrauensfragen zu automatisieren, die bei akademischer Mobilität auftreten. TIGHT^{est} definiert dazu Software Komponenten, Formate und Protokolle.

Wir zeigen, dass die Infrastruktur des Domain Name System (DNS) und seiner Sicherheitserweiterungen (DNSSEC) dazu verwendet werden kann, einen globalen Vertrauensanker für automatische Vertrauensfragen zu etablieren. Des weiteren demonstrieren wir, wie es mit Hilfe dieses Vertrauensankers und definierter Protokolle möglich ist, die zuständigen Institutionen zu finden und abzufragen. Darüber hinaus zeigen wir, wie das Verarbeiten von Bewerbungen der Studierenden automatisiert werden kann.

Keywords: Trust Management, Akademische Mobilität, DNSSEC

[2] TIGHT^{est} basiert auf das LIGHT^{est} Projekt, welches von der Europäischen Kommission als Innovation Act im Horizon2020 Programm unter der Grant Agreement Nummer 700321 gefördert wird.

Contents

List of Figures

List of Tables

Listings

List of Acronyms

API	Application Programming Interface
DANE	DNS-based Authentication of Named Entities
DNS	Domain Name System
DNSSEC	Domain Name System Security Extensions
DSA	Digital Signature Algorithm
ECDSA	Elliptic Curve Digital Signature Algorithm
eIDAS	electronic IDentification, Authentication and trust Services
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IANA	Internet Assigned Numbers Authority
ICANN	Internet Corporation for Assigned Names and Numbers
NAPTR	Name Authority Pointer
NSD	Name Server Daemon
PKI	Public Key Infrastructure
RRSIG	Resource Record digital Signature
RSA	the Rivest–Shamir–Adleman public-key cryptosystem
SaaS	Software as a Service
SHA	Secure Hash Algorithm
TLS	Transport Layer Security
TLSA	Transport Layer Security Authentication
ATV	Automated Trust Verifier
ET	Electronic Transaction
TL	Trust List
TP	Trust Policy
TSMC	Trust Scheme Membership Claim
TSP	Trust Scheme Publisher
TSPA	Trust Scheme Publication Authority

1. Introduction

Academic mobility is the process of members of the academic community moving between universities to study, teach, or do research.

In this thesis we focus on students who graduate from a university and continue their study at TIGHT^{est} one abroad. To do so, they need to register or apply at a different university. This involves proofing existing degrees. For example, students provide diplomas stating their former university, graduation level and study program. The new university then checks if the diploma is valid, the issuing university is legit, and the conditions required for applying are fulfilled – a complex and time-consuming process.

This thesis is concerned with the automatization of this process using electronic tools. This means a student submits their documents in electronic form. The documents are digitally signed. The system proposed in this thesis can be used to discover and query the authorities, which can authenticate the signer of those signatures. Those authorities are responsible for a certain domain and area. This allows automated checking of the authenticity of documents issued abroad.

1.1. Legalization of documents

A document which has been issued in one country (Country A) and is legal there is not necessarily automatically legal in TIGHT^{est} country (Country B). For the document to be legal in the other country, it is required that it's *legalized* first (Council of the Notariats of the European Union, 2008).

There are countries who are party to the *Hague Convention Abolishing the Requirement for Legalisation for Foreign Public Documents* (*Apostille Convention*) (Hague Conference on Private International Law, 1961). Those countries use a simpler form of legalization. Nevertheless, our system can also be used in transactions involving those countries.

In general, *legalization* of a document involves multiple entities.

- An entity in Country A which can issue relevant documents. For example, a university which issues graduation certificates (diplomas) to students.
- An entity which is a trusted authority in Country A, for example, the foreign ministry. The foreign ministry knows (and trusts) all legitimate entities in its country.
- An entity of Country B, for example, an embassy. An embassy knows (and trusts) the authority of the foreign ministry of Country A. In addition, the embassy is trusted by Country B (through its foreign ministry).

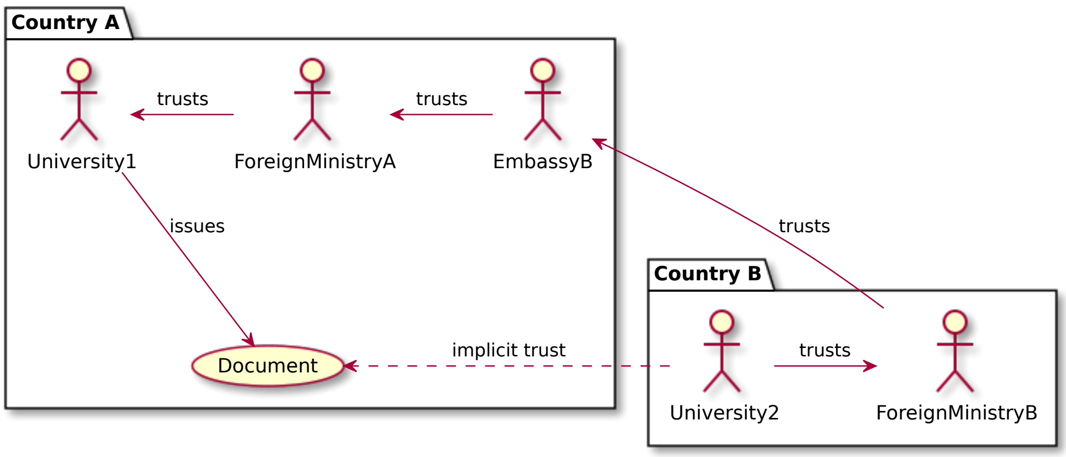


Figure 1.1.: Trust path involved in legalization of a document

This chain of trust establishes the trust path shown in Figure 1.1. Since every entity trusts the next one in the hierarchy, documents issued by the university in Country A can be authenticated by all entities in Country B. For example, by trusting its foreign ministry, a university in Country B can authenticate a student application from Country A. This creates an implicit (unidirectional) trust between the two universities.

The process of legalization has multiple steps, shown in figure 1.2.

- The process starts when a student is graduating from a university.
- First, a document (diploma) is issued by the university in Country A.
- Next, the foreign ministry of Country A comes into play. It needs to certify the document. By doing so, it certifies that the issuing university is actually a legitimate entity in Country A.

- After that, the embassy of Country B is needed to certify the document again. This time it certifies that the certification of the foreign ministry is valid. This can also involve language translators. Since the embassy is trusted in Country B (through its foreign ministry), this legalizes the document.

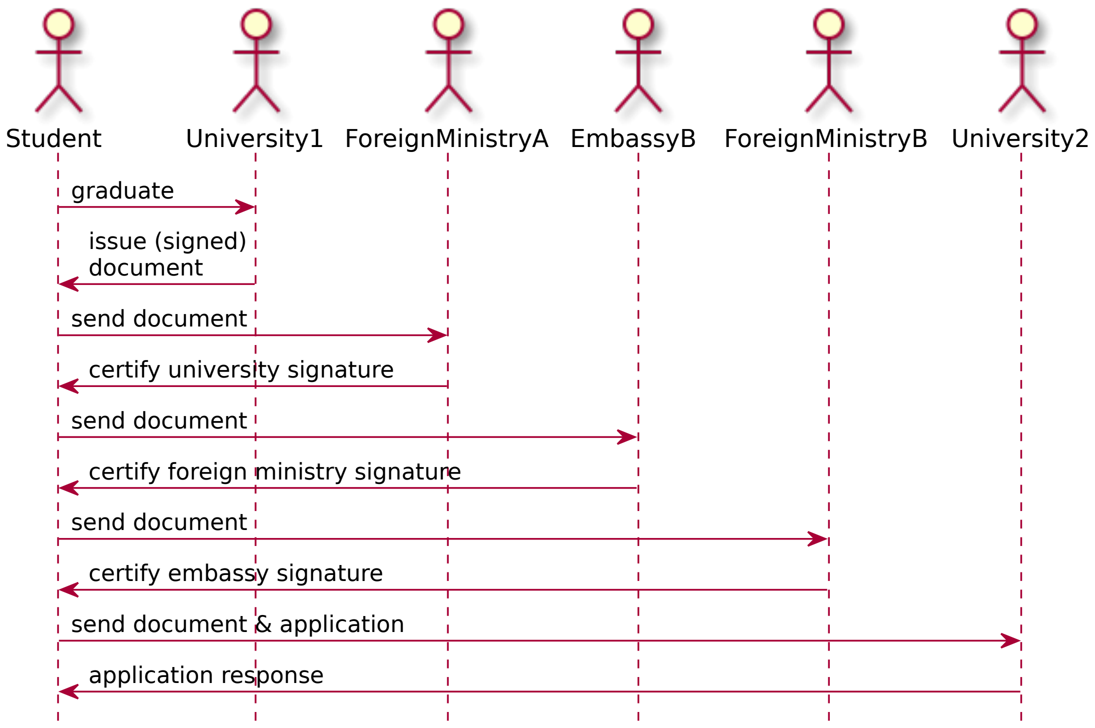


Figure 1.2.: Legalization of a document

Doing so, the chain of authorities forms a *chain of trust*. Each authority is trusted by the one next in line, as shown in Figure 1.1. The last one (in the pictured example the foreign ministry) serves as a *trusted root* for other entities in Country B. This trusted root is used by all entities in Country B to verify documents.

In the domain of academic mobility, this process can involve TIGHT^{est} step: The foreign ministry of a country might not know all legitimate universities in its territory. So it needs the certification of an institution which does, for example, the education ministry. This adds one step to the chain of trust but does not change the overall idea.

In this thesis, we look into a simplified version of this process. We assume the existence of a global entity, which is trusted by all entities in the process. This is done to simplify the demonstration of the technical feasibility. Chapter 6 explains how this limitation can be removed in a follow-up project.

1.2. Introducing TIGHT$^{\text{est}}$

In this thesis we introduce TIGHT$^{\text{est}}$. TIGHT$^{\text{est}}$ is a step towards automating the process of global legalization of electronic documents involved in academic mobility, as described in Section 1.1. It does so by defining a system which uses the Domain Name System Security Extensions (DNSSEC) root zone as global trust root. TIGHT$^{\text{est}}$ can be generalized for the legalization of all kinds of documents.

We define an architecture in the form of protocols and formats. This architecture is not specific to a programming language or operating system. This allows implementations on various platforms. Furthermore, it enables integration with existing workflows. We also provide a prototype implementation of this architecture, the TIGHT$^{\text{est}}$ reference implementation.

The TIGHT$^{\text{est}}$ infrastructure can be used by universities to automatically verify the authenticity of student applications. Furthermore, it provides means of verifying the legitimacy of the issuing university. In addition, it allows automated checking of the application using policies defined by the university.

A **TIGHT$^{\text{est}}$ process** starts with a university receiving electronic applications from a student who did their previous studies at TIGHT$^{\text{est}}$ university. An application consists of multiple electronic documents the university requests from its applicants. For the sake of simplicity, we focus on applications consisting of one document, but the process works the same for multiple documents per application.

The university can check the content of the document. This check might be trivial or non-trivial, depending on the content of the document. In addition, the university needs to check the document's authenticity. The document might contain a seal in form of a digital signature of the issuing university, which can be used to check the authenticity. To do so, the university needs to verify two things:

- Q1: Is the issuing university a recognized institution?
- Q2: Is the digital signature valid and was it issued by this university?

Those are the **trust questions** TIGHT^{est} aims to answer in an automated way.

To answer the first question, the university needs to build a chain of trust from a trusted entity to the electronic certificate which was used to sign the document. To answer the second question, the university needs to check if that certificate was actually used to sign the document.

TIGHT^{est} answers both of the questions automatically as follows:

The client software starts the process by extracting the certificate and performing a signature check. The first step is making sure that the issuing university actually signed the document. This is done by performing a simple signature check using the cryptographic keys contained in the certificate. If successful, the document is analyzed. It contains a claim stating the authority which is responsible for the issuing university. The claim comes in the form of a domain name identifying this authority. If the claimed authority is part of the university list of trusted authorities, TIGHT^{est} can use this information to verify that the issuing university is legitimate. This requires a verification that the issuing university is actually known and trusted by the authority it claims to be. The software verifies this by first discovering an Application Programming Interface (API) of this authority using the domain name. This is done in a secure way using the Domain Name System (DNS). By contacting the API, the software then verifies that the university is legitimate. The transmission of the query result is secured by well established cryptographic protocols. TIGHT^{est} concludes the check by verifying the contents of the document using a specified set of conditions an applicant has to fulfill. This process is shown in Figure 1.3.

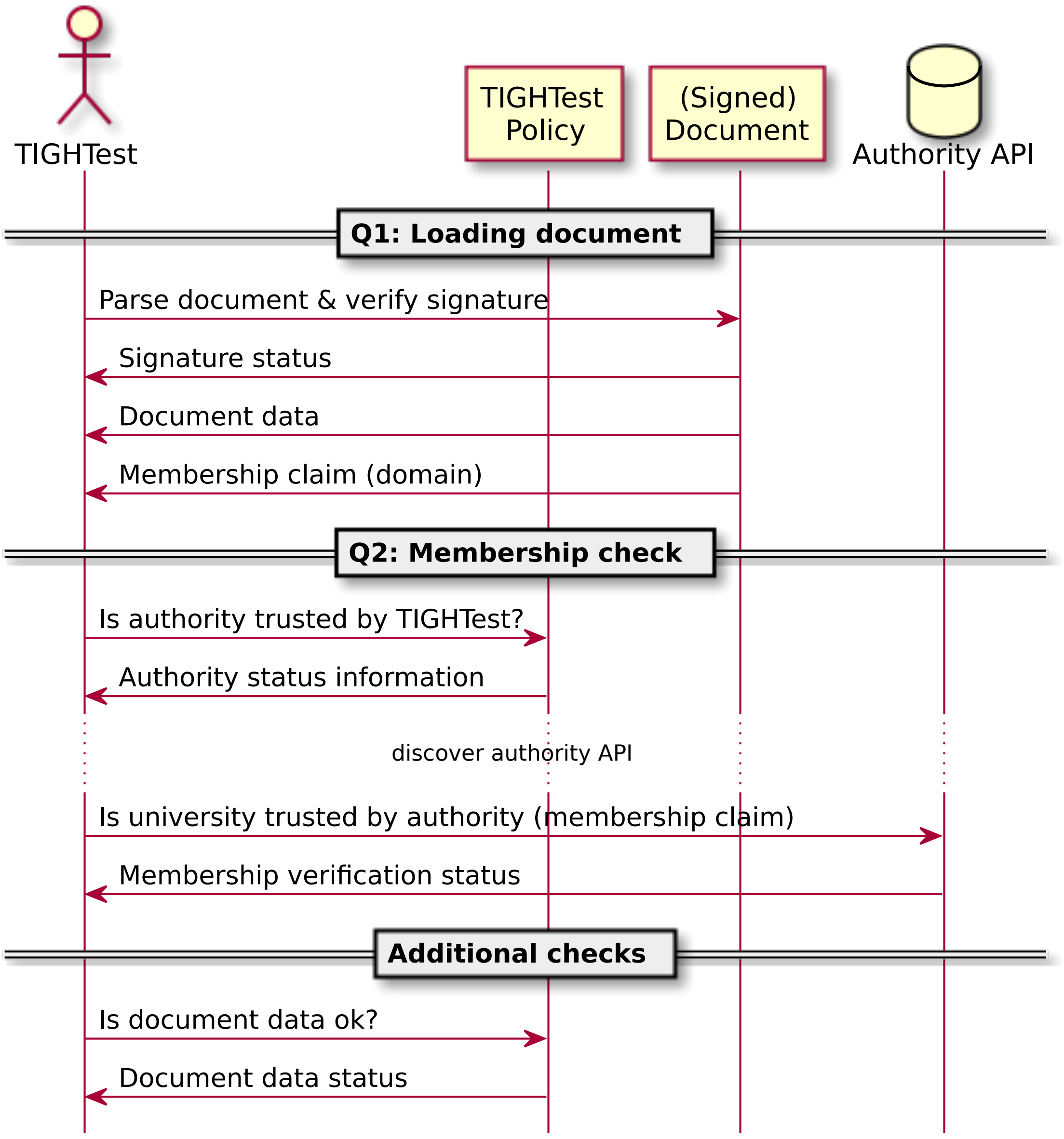

TIGHTest
TIGHTest Policy
(Signed) Document
Authority API
Q1: Loading document
Parse document & verify signature
Signature status
Document data
Membership claim (domain)
Q2: Membership check
Is authority trusted by TIGHTest?
Authority status information
discover authority API
Is university trusted by authority (membership claim)
Membership verification status
Additional checks
Is document data ok?
Document data status

Figure 1.3.: TIGHT^{est} verification process (simplified)

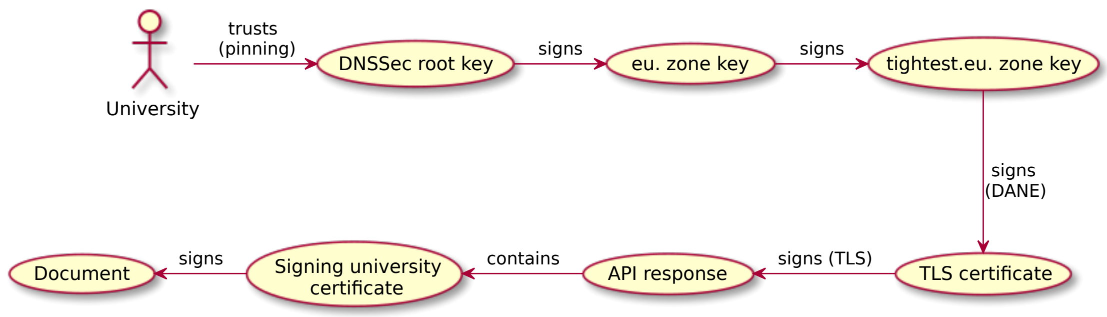


Figure 1.4.: TIGHT^{est} DNSSEC trust path (simplified)

A **chain of trust** to the document is built using the mentioned authority and the DNSSEC system. TIGHT^{est} makes sure the trust from the (trusted) global root is transferred step by step to the document. The trust path is visualized in Figure 1.4. The university needs to trust the global DNSSEC root. This is achieved by pre-configuring the keys used to sign the DNS root zone (pinning) in the tool used to carry out the checks. Since the root zone is global, these are the only keys the university needs to trust. The tool then traverses from the root zone to the domain name of the responsible authority. In DNSSEC trust is transferred to the next zone by signing its keys. The process continues until the trust path reaches the claimed domain name. This transfers trust to the authority. After doing so, the tool uses the domain name of the authority to discover a Hypertext Transfer Protocol (HTTP) API. The communication with the API is secured by Transport Layer Security (TLS). Trust in the API is established by using the (now trusted) DNSSEC keys of the authority to sign the TLS certificate. This is done using DNS-based Authentication of Named Entities (DANE). This authenticates the response of the API. By querying the API with the university certificate, trust is finally transferred to the certificate itself. The certificate of the issuing university is then linked to the document by means of a digital signature. DANE is explained in more detail in Section 2.5.3.

1.3. Research Questions

TIGHT^{est} is a research prototype. It is crafted to explore technical questions, and not to provide a ready-to-use solution to the stated problem.

The main focus of the TIGHT^{est} project is concerned with the process of establishing trust. This involves building a trust path from the university to the authority. In addition, it covers the discovery of the authority API, which can be used to verify the legitimacy of a foreign university. Furthermore, it focuses on the transfer of trust from the DNS to the API.

In more general terms, this thesis tries to answer the following questions: How can the DNS be used to discover trust services in a secure way? Are the technical aspects proposed in the LIGHT^{est} project feasible?

1.4. Assumptions

In this thesis we assume a world which is globally digitalized. We assume that universities issue digital graduation diplomas to their students when they graduate. Those diplomas are cryptographically signed. Furthermore, we assume all universities use a standardized format for issuing those diplomas.

In addition, for this thesis we assume there is a globally recognized entity which coordinates institutions of higher education. This is used to simplify the scope of the thesis. We discuss this limitation in Section 5.5.

1.5. Thesis outline

The rest of this thesis is structured as follows: Chapter 2 introduces some of the technologies used in this thesis. It focuses on an introduction into the DNS and DNSSEC. Chapter 3 briefly discusses the project this thesis is based on, LIGHT^{est}. Chapter 4 introduces the architecture of TIGHT^{est} and its components, while Chapter 5 goes more into detail of our reference implementation. Chapter 6 looks into the future and discusses limitations of TIGHT^{est} and how one could remove them. Chapter 7 concludes the thesis.

1.6. Acknowledgements

The concepts used in this thesis are based on the LIGHT^{est} project. The LIGHT^{est} project is introduced in Chapter 3. Chapter 6 discusses the relation between the two projects in more detail.

The use-case of academic mobility is based on a contribution to the LIGHT^{est} project by Sebastian Mödersheim and Rasmus Birkedal (Technical University of Denmark) (LIGHTest Consortium, 2017).

2. Preliminaries

In this chapter we introduce several concepts, terms, and systems in the domain of our thesis.

2.1. Trust Management

The field of trust management deals with questions related to trust decisions. It is concerned with the issue of how and in what way information provided by a party can be trusted (Blaze, Ioannidis, and Keromytis, 2003). Furthermore, it is concerned with the identity of the involved parties itself.

Common applications of trust management are authentication and authorization. Concepts include access policies, passwords and other authentication factors.

2.1.1. Trust Infrastructure

A trust infrastructure helps dealing with trust decisions in an automated way. It is often backed by some formal rules, like a law, a treaty between countries, company policies, or a contract between companies.

An example of a trust infrastructure relevant for TIGHT$^{\text{est}}$ are the trust services envisaged by Europe's electronic IDentification, Authentication and trust Services (eIDAS) regulation, like electronic signatures, qualified digital certificates, electronic seals and timestamps.

2.2. Public-key Cryptography

Public-key cryptography references a class of cryptographic algorithms which require two different keys (Menezes, Van Oorschot, and Vanstone, 1997). In the context of digital signatures, the so called private-key is used to sign a message. On the other hand, a public-key is needed to verify the signature. This allows anyone in the possession of the public-key (which is by definition public) to verify the signature. In addition, it makes sure that only the person in the possession of the private-key can create a valid signature. Public-key cryptography is also called *asymmetric cryptography*. The process principle is illustrated in Equations 2.1 and 2.2.

$$\text{signature} = \text{signer}(\text{message}, \text{privateKey}) \tag{2.1}$$

$$\text{signStatus} = \text{verifier}(\text{signature}, \text{message}, \text{publicKey}) \tag{2.2}$$

2.3. X.509

X.509 is a format for public-key certificates (Cooper et al., 2008). A certificate is used to link an identity to a public-key. It is a data structure containing the public-key, the identity information, and some metadata. The certificate is then either self-signed or signed by an authority.

The authority which issues certificates is called *Certificate Authority* (CA). In the same way, the identity of a CA can be certified by TIGHT$^{\text{est}}$ CA. This creates a hierarchical trust tree, called Public Key Infrastructure (PKI). An example PKI structure is shown in Figure 2.1. If a CA is at the top of this tree, its certificate has to be self-signed. Such a CA acts as a root of trust. It is therefore called *Root Certificate Authority* (Root CA). A PKI trust system has many roots of trust. If a CA is not at the top of the tree, and therefore signed by TIGHT$^{\text{est}}$ CA, it is called *Intermediate CA*.

X.509 certificates are represented using *Abstract Syntax Notation.One* (ASN.1).

X.509 certificates can be used to authenticate arbitrary data. For example, it is used in Transport Layer Security (TLS) to authenticate a communication participant in a secure connection. In addition, it is possible to use the public-key of a certificate to authenticate an electronic document.

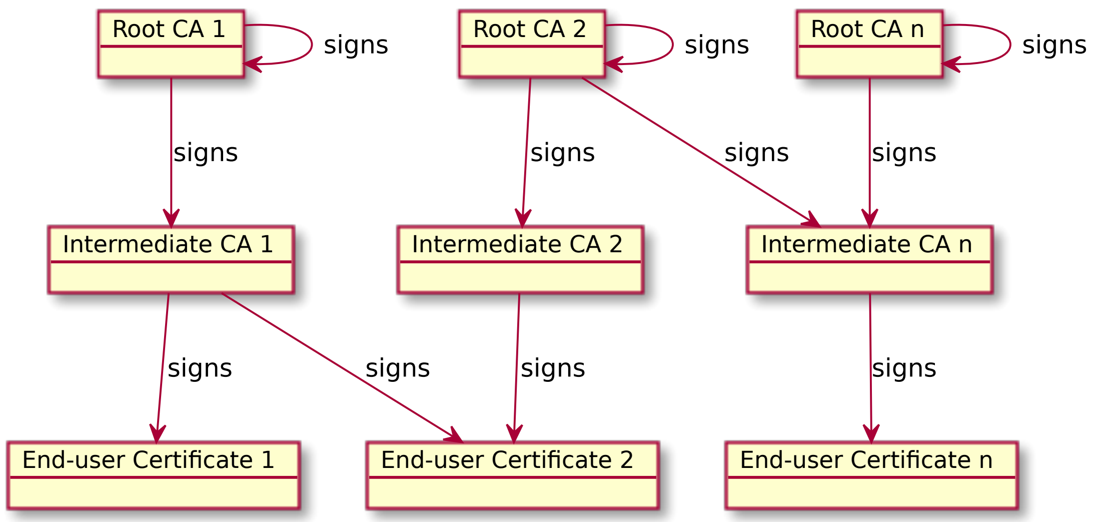


Figure 2.1.: Example (simplified) PKI hierarchy

2.4. TLS / HTTPS

TLS is a protocol used to protect network connections. "The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery" (Dierks and Rescorla, 2008).

TLS provides confidentiality by using symmetric cryptography to encrypt the transmitted data. In addition, it authenticates the transmitted data by using public-key cryptography. This is done by using the public-key from an authenticated X.509 certificate to sign agreement data during the connection handshake.

TLS is used in the Hypertext Transfer Protocol Secure (HTTPS) protocol to protect HTTP communication.

2.5. Domain Name System (DNS)

The Domain Name System (DNS) is the address book of the Internet. In its core it is a distributed key-value database. It consists of a set of protocols, and software components implementing those protocols to talk to each other. Its main usage is the translation of addresses from a human-readable representation (*domain name*) into a representation used by computers to communicate on the Internet (*IP address*).

DNS is specified in multiple RFCs. It was first defined by Mockapetris, (1987a) and Mockapetris, (1987b), later Postel, (1994).

A domain name has a hierarchical structure, each level separated by a period. Every level of the hierarchy is called a *zone*. For example, the domain name *tightest.eu.* (with a period on the right end) represents the zone *tightest*, which is part of the *eu* zone (top-level domain), which by itself is part of the . zone (*root zone*). This hierarchy is shown in Figure 2.2.

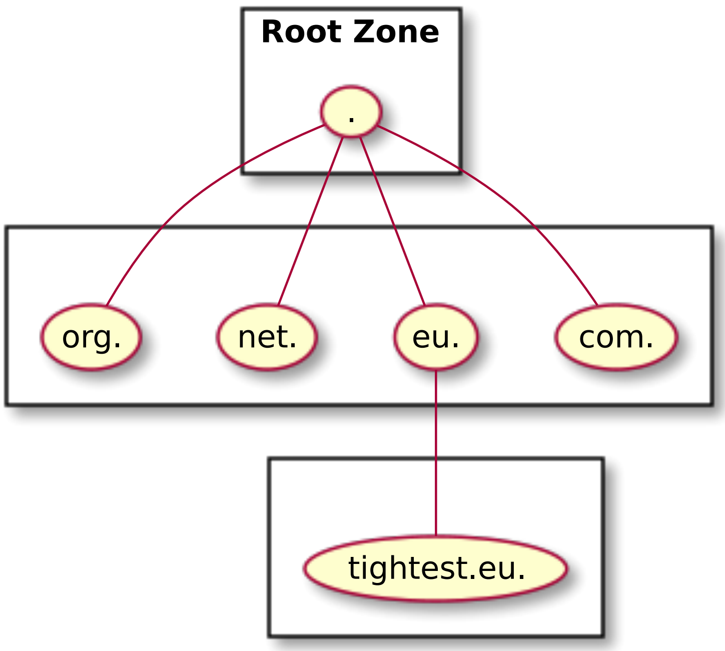


Figure 2.2.: Example DNS hierarchy for *tightest.eu.*

The process of translating a domain name into an IP address is called *resolving*. A client computer uses a *DNS resolver* to resolve a domain name into an IP address. Resolving a domain name works in a hierarchical way, following the structure of the domain name. Usually, the local DNS resolver (*stub resolver*) talks to a *recursive* resolver on the Internet. A recursive resolver is commonly hosted by an infrastructure provider (ISP, ...) and has a cache for domains to be resolved. The resolver starts resolving the domain name from the right-hand side, the root zone. The servers which answer DNS queries for a zone are called *nameserver*. The root zone nameserver tells the resolver the nameserver of the desired zone. After receiving the information via the DNS protocol, the recursive resolver sends the next query to the zones nameserver. The recursive resolver repeats this process until it reaches the left-most part of the domain name. The nameserver responsible for the left-most part is called authoritative nameserver. After traversing the zone hierarchy, the stub resolver ends up with the IP of the server identified by the domain name.

For example, while resolving *tightest.eu.*, the resolver first queries the root zone (.) nameserver for information about the *eu* zone. Next, it queries the *eu* zone nameserver for information about *tightest.eu*. Using this nameserver, it retrieves the desired IP address and finally establishes a connection to the server. This process is shown in Figure 2.3.

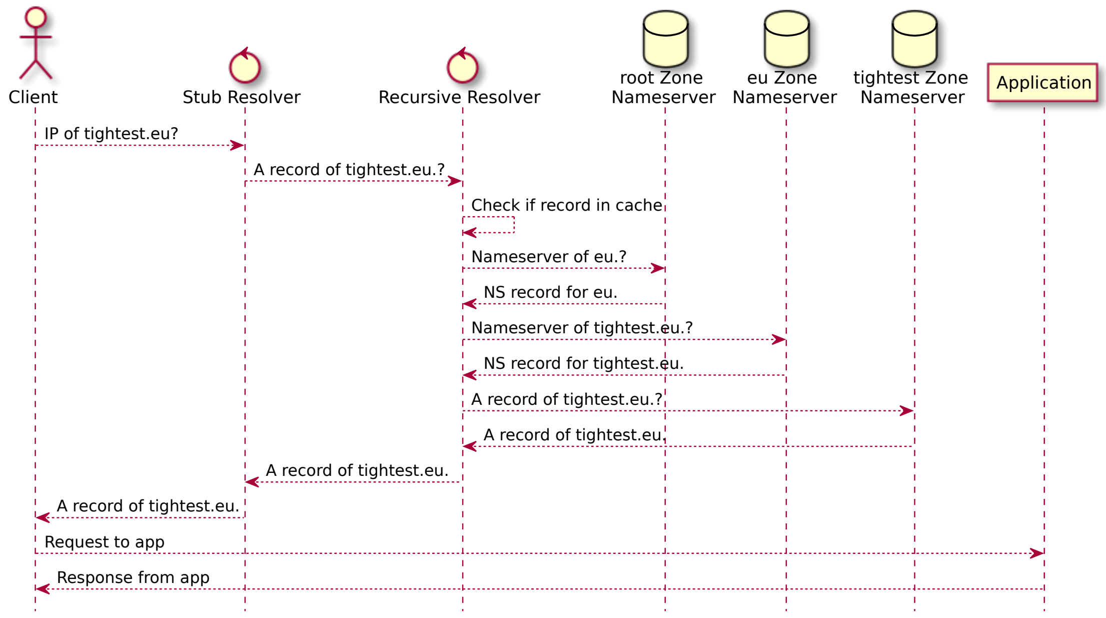


Figure 2.3.: Example DNS query for *tightest.eu*.

A DNS resolver queries a DNS server for a specific *record*. In addition to a (domain) name, such a record is identified by its *record type*. Depending on the queried record type, the DNS server responds with a different record. By doing so, the DNS does support many different forms of queries. For example, the answer to a query for the IP address of a domain name is called *A record* (for an IPv4 address).

An example query and response for an A record is shown in Listing 2.1. In addition to the desired A record, the server returns a list of the authoritative nameservers for this zone.

```
;; QUESTION SECTION:
tightest.eu.    IN      A

;; ANSWER SECTION:
tightest.eu.    5       IN      A       185.194.143.204

;; AUTHORITY SECTION:
tightest.eu.    10      IN      NS      ns1.failing.systems.
tightest.eu.    10      IN      NS      ns2.failing.systems.
```

Listing 2.1: Example A record for tightest.eu.

Operation of the DNS involves many parties, following the hierarchy shown in Figure 2.2. For example, the root zone nameserver (. zone) is operated by the Internet Corporation for Assigned Names and Numbers (ICANN) and its subsidiary Internet Assigned Numbers Authority (IANA)[1], while the top level domain nameservers (e.g. *.eu* zone) are run by different *registries*. In addition, the authoritative nameservers are operated by cloud infrastructure providers, web hosters, or the service providers itself.

The configuration file used to describe a zone is called *zone file*. A zone file is loaded by the nameserver software and contains all records of a zone, or pointers to other nameservers. Listing 2.2 shows the content of an example zone file. The main content is the A record also used in the query shown in Listing 2.1. In addition, it lists the zone name itself, and Start of Authority (SOA) and Name Server (NS) records needed to operate the zone.

[1] https://www.iana.org/domains/root

```
$ORIGIN tightest.eu.
$TTL 10

@    IN       SOA    ns1.failing.systems.       s.failing.systems. (
201712002              ; serial number
3600                   ; refresh
900                    ; retry
1209600                ; expire
10                     ; ttl
)

; Name servers
     IN       NS     ns1.failing.systems.
     IN       NS     ns2.failing.systems.

; A records of .tightest.eu
@    IN       A       185.194.143.204
```

Listing 2.2: Example zone file for tightest.eu. zone

2.5.1. NAPTR Records

Name Authority Pointer (NAPTR) records are a type of records returned by a DNS server (Mealling and Daniel, 2000). Its main usage is in Internet telephony to translate phone numbers into Internet addresses. In a more generic way, they allow translation of domain names to uniform resource identifiers (URIs). In contrast to other record types, the response to a NAPTR query does not contain the final response. Instead the nameserver returns a response containing a regular expression. The application then uses this regular expression and other query data to construct the desired URI. An example NAPTR query and response is shown in Listing 2.3.

```
;; QUESTION SECTION:
scheme.tightest.eu.      IN        NAPTR

;; ANSWER SECTION:
scheme.tightest.eu.      10        IN        NAPTR    100 10 "U" "tightest" "!^(.*)
    $!https://tightest.eu/query/\\1.json!" .
```

Listing 2.3: Example NAPTR records

2.5.2. Domain Name System Security Extensions (DNSSEC)

The DNS does not provide any form of protection of the transmitted or stored data. It is therefore possible to eavesdrop on the resolved domains or returned server address. It is furthermore possible to replace the returned address, by doing so redirecting the traffic to a malicious host. To counteract the dangers of the latter, a set of extensions to the DNS, called Domain Name System Security Extensions (DNSSEC) (Eastlake, 1999) has been defined.

The main feature of DNSSEC is to digitally sign the zone content and therefore provide a chain of trust from the root zone to the leaf zone. The root zone's signing key acts as a root of trust. This structure authenticates the response to DNS queries and enables to resolve and verify the integrity of every (signed) domain in the world by trusting one global root. Since it is possible to store arbitrary records in the DNS, DNSSEC effectively provides a global integrity protected database. This is what TIGHT^{est} uses to build its trust path, as described in Chapters 4 and 5.

To build this trust path, DNSSEC follows the DNS hierarchy introduced in Section 2.5. This is done using public-key cryptography. Every (signed) zone has two key pairs, called *zone-signing key* (ZSK) and *key-signing key* (KSK). The reason for two separate key pairs are operational requirements. For example, the ZSK is loaded on the server operating the zone, while the KSK can be kept offline. In case of a security breach, it is then possible to replace the ZSK and re-sign the zone during operation without effecting other zones.

The trust path starts at the root zone. Every DNSSEC-enabled resolver needs to know the key-signing key of the root zone. This is done by pinning the key, for example by hard-coding it in the program code, or shipping it together with operating system updates. The root zone key-signing key is used by the operator of the root zone to sign the zone-signing key. The signature of the zone-signing key is put next to it in the root zone file.

A resolver verifies the authenticity of a record in the root zone by first verifying the signature on the root-signing key using its pinned public-key part of the key-signing key. It then uses the zone-signing key to verify the signatures on the zone records. A simplified version of this trust path is shown in Figure 2.4.

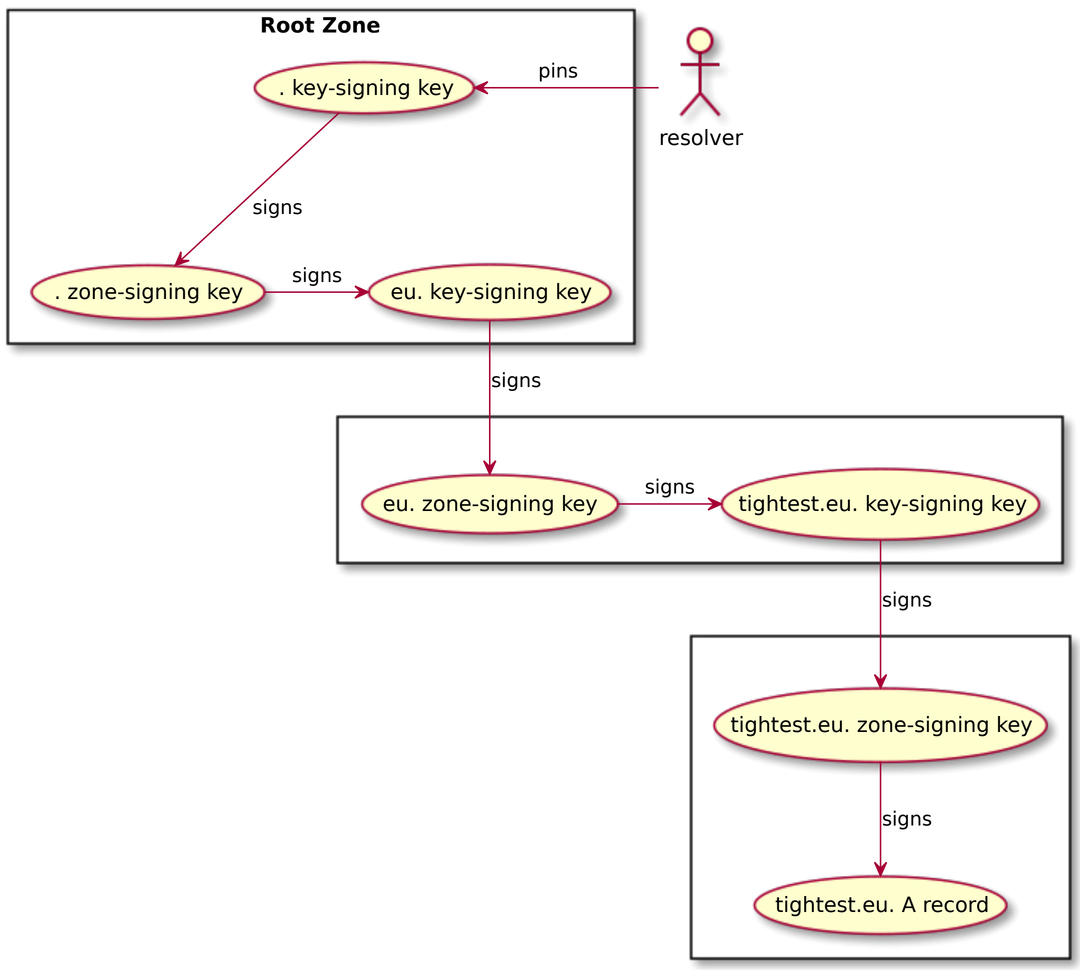


Figure 2.4.: Example DNSSEC hierarchy for *tightest.eu*.

To transfer trust from a zone to the next one in the DNS hierarchy, DNSSEC pins the zone-signing key of the child zone in the parent zone. This is done by adding the hash digest of the KSK public key to the zone file of the parent zone. In addition, the parent zones ZSK is used to sign this record. Figure 2.5 shows in more detail how the transfer of trust from a parent zone to a child zone works. An even more detailed view is shown in the Appendix by Figure A.1.

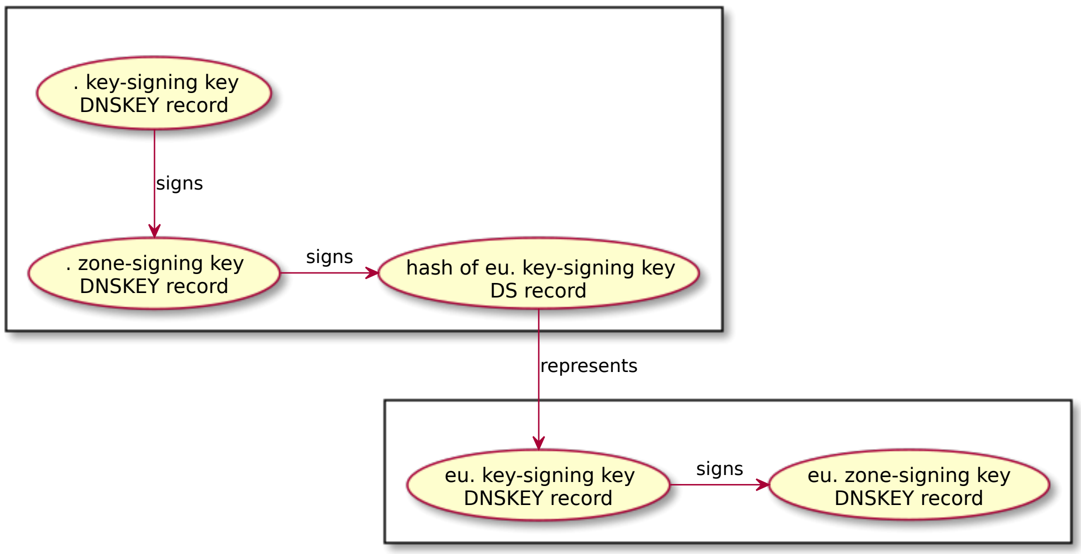

. key-signing key DNSKEY record
signs
. zone-signing key DNSKEY record
signs
hash of eu. key-signing key DS record
represents
eu. key-signing key DNSKEY record
signs
eu. zone-signing key DNSKEY record

Figure 2.5.: Example DNSSEC zone transfer

A stub resolver requests a DNSSEC enabled resolving by setting the *DNSSEC OK* (DO) bit. The recursive resolver then uses the data provided by DNSSEC to authenticate the response. If successful, the recursive resolver sets the *Authenticated Data* (AD) bit when sending the query result back to the stub resolver. The process of DNSSEC-enabled resolving is shown in Figure 2.6.

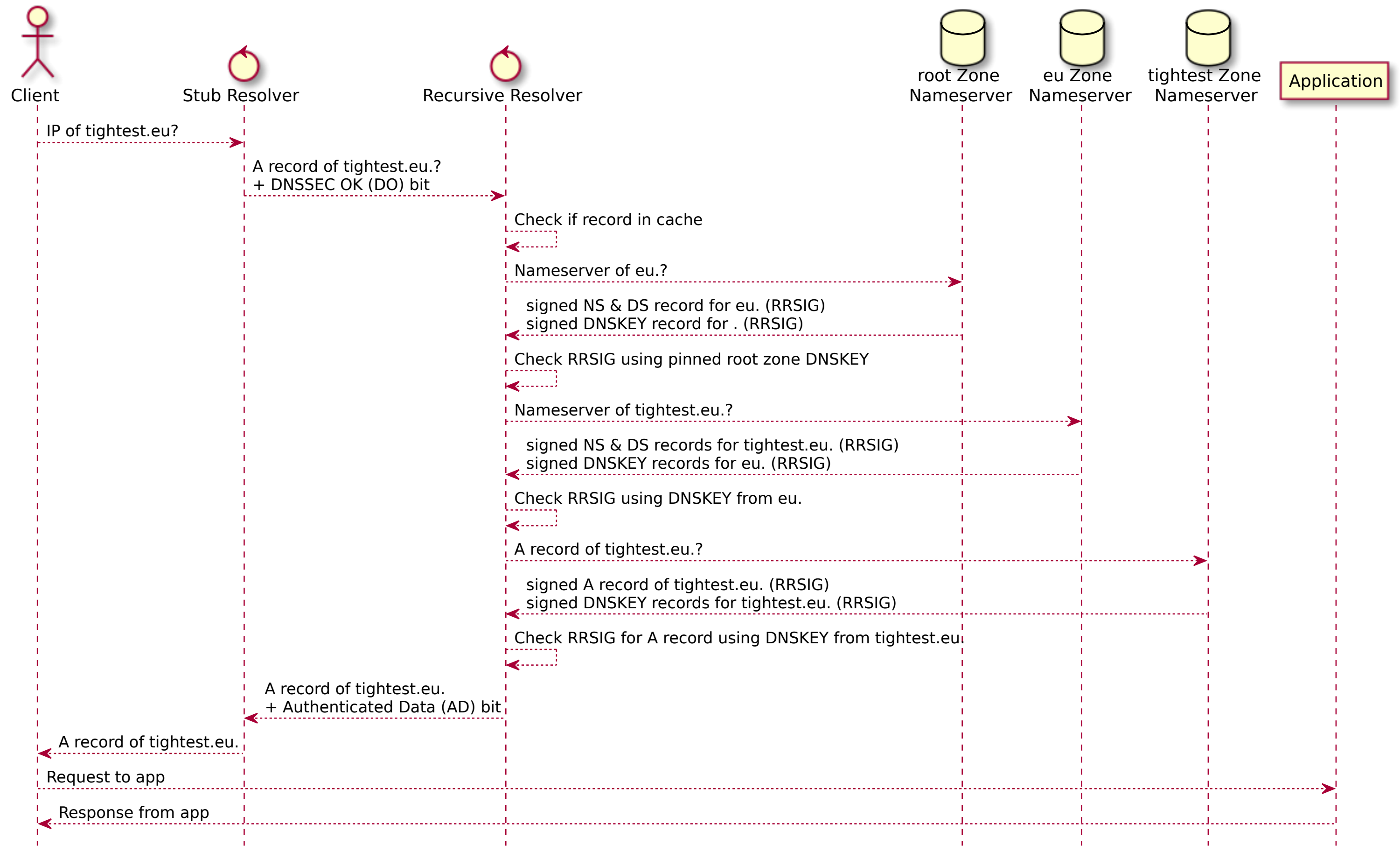

Client
Stub Resolver
Recursive Resolver
root Zone Nameserver
eu Zone Nameserver
tightest Zone Nameserver
Application
IP of tightest.eu?
A record of tightest.eu.?
+ DNSSEC OK (DO) bit
Check if record in cache
Nameserver of eu.?
signed NS & DS record for eu. (RRSIG)
signed DNSKEY record for . (RRSIG)
Check RRSIG using pinned root zone DNSKEY
Nameserver of tightest.eu.?
signed NS & DS records for tightest.eu. (RRSIG)
signed DNSKEY records for eu. (RRSIG)
Check RRSIG using DNSKEY from eu.
A record of tightest.eu.?
signed A record of tightest.eu. (RRSIG)
signed DNSKEY records for tightest.eu. (RRSIG)
Check RRSIG for A record using DNSKEY from tightest.eu.
A record of tightest.eu.
+ Authenticated Data (AD) bit
A record of tightest.eu.
Request to app
Response from app

Figure 2.6.: Example DNSSEC query for *tightest.eu*.

DNSSEC Record Types

DNSSEC signs records in a zone file to provide proof of authenticity of records. The signature of a record is added next to the record in the same zone file. In addition, it adds the hash digests of the keys used to sign a zone in the parent zone's zone file. Furthermore, DNSSEC is able to show the non-existence of a record name. To do al of this, it introduces multiple new record types:

- **RRSIG:** signature of a record, put right next to the record
- **DNSKEY:** public key used to sign the zone, used to verify the signatures in *RRSIG* records
- **DS:** hash digest of the key, put in the parent zone file
- **NSEC/NSEC3:** link to the next record name in the zone, used to show non-existence of a record name

Listing 2.4 shows an example signed zone file. This zone file contains one SOA record, one A record, two NS records and two DNSKEY records. In addition, it contains the signatures of all those records. The example leaves out the NSEC3 records, since they are not relevant for this thesis.

```
tightest.eu.    10      IN      SOA     cloud.failing.systems. s.failing.
    systems. 201706002 3600 900 1209600 10
tightest.eu.    10      IN      RRSIG   SOA 13 2 10 20171223022555
    20171125022555 65265 tightest.eu.   <signature>

tightest.eu.    10      IN      A       185.194.143.204
tightest.eu.    10      IN      RRSIG   A 13 2 10 20171223022555
    20171125022555 65265 tightest.eu. <signature>

tightest.eu.    10      IN      NS      ns1.failing.systems.
tightest.eu.    10      IN      NS      ns2.failing.systems.
tightest.eu.    10      IN      RRSIG   NS 13 2 10 20171223022555
    20171125022555 65265 tightest.eu. <signature>

tightest.eu.    10      IN      DNSKEY  256 3 13 <key_data> ;{id = 65265 (zsk
    ), size = 256b}
tightest.eu.    10      IN      DNSKEY  257 3 13 <key_data> ;{id = 22696 (ksk
    ), size = 256b}
tightest.eu.    10      IN      RRSIG   DNSKEY 13 2 10 20171223022555
    20171125022555 22696 tightest.eu. <signature>
```

Listing 2.4: Example signed DNS zone (without NSEC3 records)

It is important to note that DNSSEC does not protect the confidentiality of DNS queries and results, since it is not encrypting data.

2.5.3. DNS-based Authentication of Named Entities (DANE)

To ensure safe transmission of Hypertext Transfer Protocol (HTTP) resources, the HTTPS protocol is used. To authenticate an HTTPS connection, X.509 certificates and PKI are used. In contrast to DNSSEC, in PKI there are many roots of trust, called Root Certificate Authorities.

DNS-based Authentication of Named Entities (DANE) has been proposed as an alternative trust model to PKI (Barnes, 2011; Hoffman and Schlyter, 2012). It uses the global single root of trust of DNSSEC to establish a chain of trust to an X.509 certificate. This is done by storing the certificate's fingerprint in a DNS record and signing the record. By doing so, trust is transferred from the (authenticated) DNS zone to the X.509 certificate, and therefore to the TLS connection. Using this approach, trust in the root-zone key is enough to authenticate any HTTP response, e.g., content served by an Application Programming Interface (API). Figure 2.7 shows this trust path.

The DNS record type used to store the X.509 certificate fingerprint in DNS is called Transport Layer Security Authentication (TLSA). Listing 2.5 shows an example zone file with a TLSA record. The TLSA record is followed by its signature in an RRSIG record.

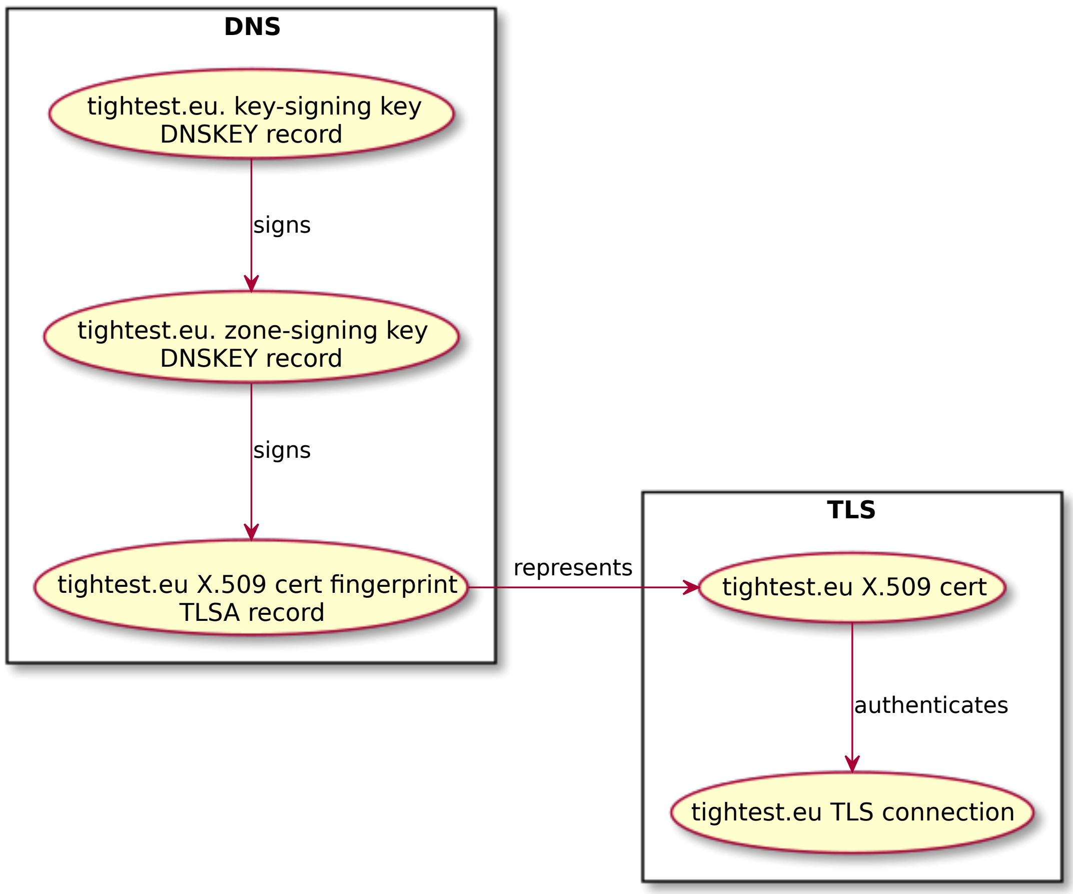

DNS
tightest.eu. key-signing key DNSKEY record
signs
tightest.eu. zone-signing key DNSKEY record
signs
tightest.eu X.509 cert fingerprint TLSA record
represents
TLS
tightest.eu X.509 cert
authenticates
tightest.eu TLS connection

Figure 2.7.: Example DANE trust transfer

```
_443._tcp.tightest.eu.   10    IN    TLSA     3 1 2 99d6d...9c8079e
_443._tcp.tightest.eu.   10    IN    RRSIG    TLSA 13 4 10 20180204232452
    20180107232452 65265 tightest.eu. <signature>
```

Listing 2.5: Example TLSA record in a DNSSEC signed zone file

2.6. XML Signatures

XML Signature is an XML-based format for digital signatures (Solo et al., 2008). It can be used to sign arbitrary data, for example an image, an HTML file, or an XML document.

An XML Signature contains references to the signed data and used keys. In addition, it contains information about the canonicalization of the data, what algorithm was used to form the message digest, and how the signing was done. Furthermore, it contains the actual signature. Example signature data is shown in Listing 2.6.

```
<Signature Id="MyFirstSignature" xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
  <CanonicalizationMethod Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
  <SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-
      sha256"/>
  <Reference URI="http://www.w3.org/TR/2000/REC-xhtml1-20000126/">
    <Transforms>
      <Transform Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
    </Transforms>
    <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
    <DigestValue>dGhpcyBpcyBub3QgYSBzaWduYXR1cmUK...</DigestValue>
  </Reference>
</SignedInfo>
  <SignatureValue>...</SignatureValue>
  <KeyInfo>
   <KeyValue>
     <DSAKeyValue>
       <P>...</P><Q>...</Q><G>...</G><Y>...</Y>
     </DSAKeyValue>
   </KeyValue>
  </KeyInfo>
</Signature>
```

Listing 2.6: Example XML Signature[2]

[2]https://www.w3.org/TR/xmldsig-core/#sec-o-Simple, accessed 2018-01-12

3. Related Work

In this chapter we introduce LIGHT^{est}, the project on which TIGHT^{est} is based.

To the best of our knowledge there are no other systems that try to build a global trust management infrastructure for academic mobility.

Lightweight Infrastructure for Global Heterogeneous Trust management in support of an open Ecosystem of Stakeholders and Trust schemes (LIGHT^{est}) is an EU-funded project working towards establishing a global trust management infrastructure (Bruegger and Lipp, 2016). It is concerned with similar questions as the TIGHT^{est} project but develops a more abstract and broader system. This chapter gives a high-level overview of the system.

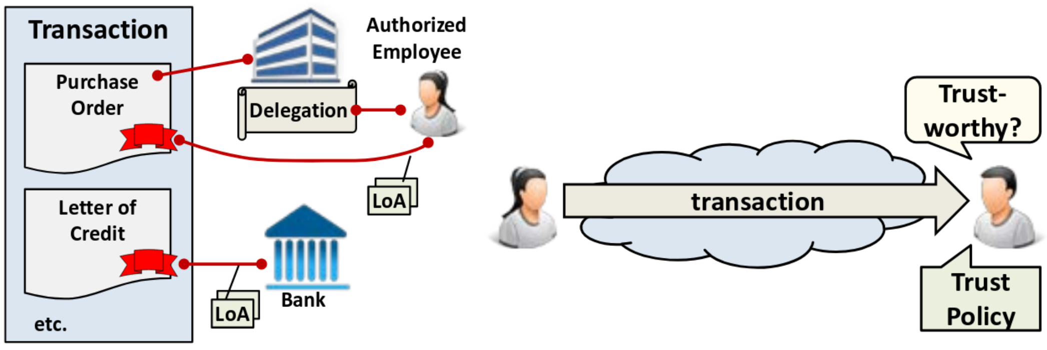


Figure 3.1.: Example LIGHT^{est} trust decision (Bruegger and Lipp, 2016)

LIGHT^{est} establishes a system which is able to **answer trust questions**. A trust question is concerned with whether the claims made in an electronic transaction can be trusted. Trusted in this context means an authority qualified for that domain has signed it. An electronic transaction is a collection of documents signed by various entities. LIGHT^{est} uses those signatures to establish trust in the documents. For example, when two companies take part in a business transaction, a purchase order is issued. In our example, a letter of credit is attached to the purchase order. The company making that purchase is signing the purchase order, while a bank may sign the letter of credit. The seller then uses this authenticated information to decide if the buyer is trustworthy. This example is shown in Figure 3.1. In addition to transactions between companies, LIGHT^{est} supports transactions between individual people and other forms of organizations, but also devices.

LIGHT^{est} proposes various software components, protocols and formats to automate those trust decisions. The system uses the **global trust root** of the Domain Name System Security Extensions (DNSSEC) to establish trust in transactions and entities. An overview of the architecture is shown in Figure 3.2.

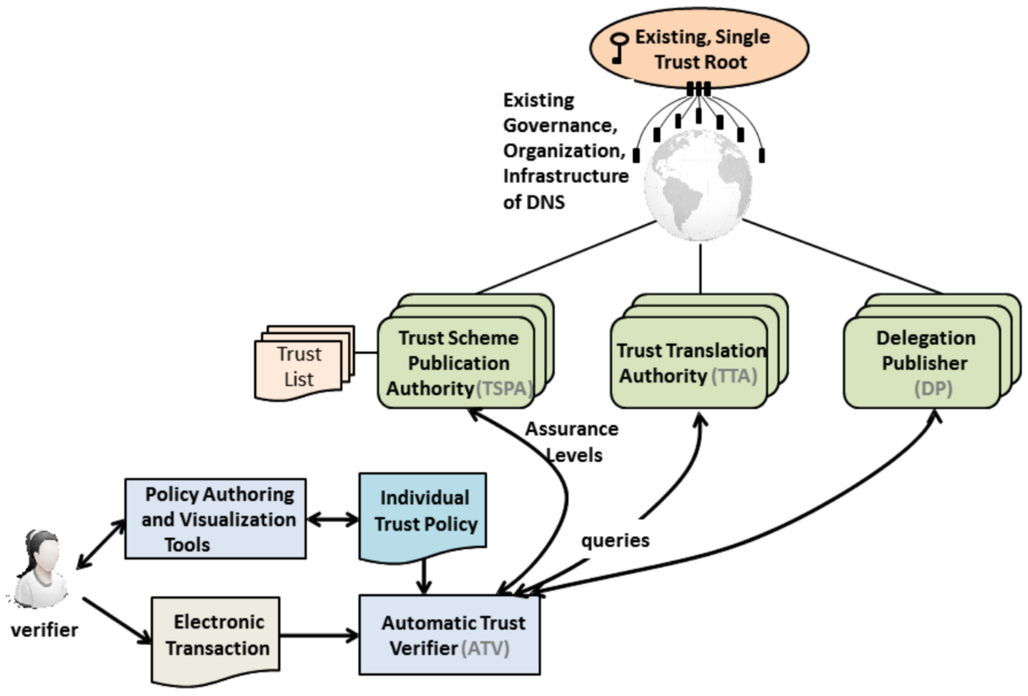

Existing, Single Trust Root
Existing Governance, Organization, Infrastructure of DNS
Trust List
Trust Scheme Publication Authority (TSPA)
Trust Translation Authority (TTA)
Delegation Publisher (DP)
Assurance Levels
queries
Policy Authoring and Visualization Tools
Individual Trust Policy
verifier
Electronic Transaction
Automatic Trust Verifier (ATV)

Figure 3.2.: The LIGHT^{est} Reference Architecture (Bruegger and Lipp, 2016)

LIGHT^{est} uses a software tool to carry out those trust decisions. It is possible to customize the behavior of this tool using **trust policies**. Such policies enable using different trust schemes by defining them as trust anchor in the trust policy. In addition, other **transaction specific policies** can be defined. For the above example, this could be a maximum transaction volume, or a list of trusted banks.

LIGHT^{est} is a system built with **existing trust infrastructures** kept in mind. It was inspired by the European trust management framework eIDAS. This framework already establishes trust infrastructures in the European Union. An example trust service is the use of qualified signatures in European member states, which is a valid signature in the whole European Union. To establish a list of trusted authorities, the European Commission publishes a trust list. The authorities on this list act as trusted authorities who sign certificates. Signatures issued using one of those certificates are qualified signatures in the European Union trust scheme. In addition, a trust list contains information about expired or revoked authorities. LIGHT^{est} creates a global standard way of publishing trust lists using the Domain Name System (DNS).

In addition, LIGHT^{est} provides means to **translate trust** from one trust scheme to TIGHT^{est}. This allows delegating the negotiation with other trust schemes to a higher authority. For example, the European Commission negotiated various treaties with other countries. In LIGHT^{est}, it could publish the rules contained in those treaties in an electronic way using the DNS. In addition, the European Commission publishes information on how to translate different levels of assurance between the trust schemes. This helps the software at a company carrying out a trust decision involving a transaction issued by an entity that is part of a different trust scheme. The company does not need to trust the other trust scheme. Instead, the company trusts the decision of the trust scheme authority of its trust scheme and thus the published translation of trust to the other trust scheme. For example, a company part of the Swiss trust scheme would like to order something from a company part of the European Commission trust scheme. To establish trust in the order document, the seller needs to establish trust in the buyer's trust scheme. It does so by querying the trust schemes authority for a translation and sets up the trust path. This example is illustrated in Figure 3.3.

Delegations are TIGHT^{est} concept supported by LIGHT^{est}. Transactions are often not signed by an organization certificate directly. Instead, individual employees sign the transaction with their personal certificate. To transfer trust from the company to the employee, the company publishes a delegation mandate. This is done using DNS similarly than with trust translations.

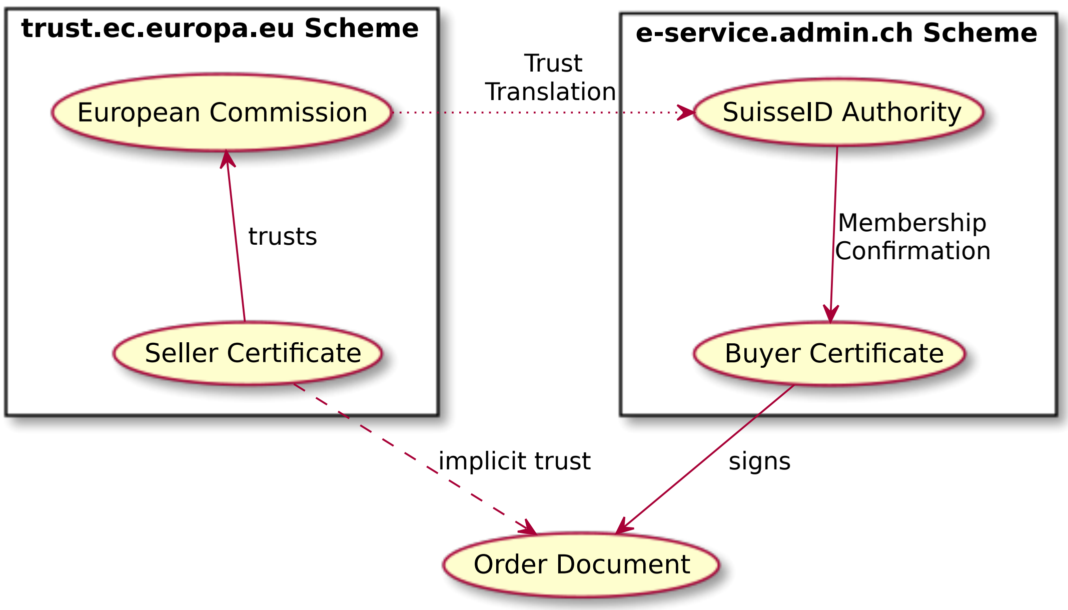


Figure 3.3.: Example LIGHT^{est} trust translation

The LIGHT^{est} project develops a generic framework. In contrast, the main goal of TIGHT^{est} was to strip away some concepts and features to focus on questions related to the technical feasibility of some design ideas. Furthermore, while LIGHT^{est} is a very abstract system, TIGHT^{est} focuses on the problems and processes of academic mobility. Thus, it implements one of many use-cases of LIGHT^{est}. In principle, TIGHT^{est} is a slim LIGHT^{est} prototype, thus its name *tiny*. Furthermore, LIGHT^{est} extends the concepts also present in TIGHT^{est}. This is described in more detail in Chapter 6.

4. TIGHT^{est} Architecture

This chapter introduces the architecture of TIGHT^{est}. It starts with a brief overview of the components of TIGHT^{est} and how they interact. It continues by explaining the components in more detail. The chapter concludes with a discussion of the involved actors.

TIGHT^{est} itself is not a software, but a suite of protocols and formats. This allows any component to exist in various or multiple programming languages. Furthermore, it allows integration in existing software and customization to business processes. This chapter discusses the protocols and formats of TIGHT^{est}. The details of the reference implementation are described in Chapter 5.

4.1. Component Overview

-	Local university	Entity
-	Issuing university	Entity
ET	Electronic Transaction	Document
TP	Trust Policy	Document
TL	Trust List	Document
TSPA	Trust Scheme Publication Authority	Entity
TSP	Trust Scheme Publisher	Software
ATV	Automated Trust Verifier	Software
-	TSP Discovery	Protocol
-	TS Membership Verifying	Protocol

Table 4.1.: Overview of the TIGHT^{est} components.

To perform a check as described in Section 1.2, TIGHT^{est} requires several components. Some of those components are software, others are documents or entities. In this section we discuss how they are connected. An overview of the involved components is given in Table 4.1. Figure 4.1 shows how the components are connected. The following sections describe them in more detail.

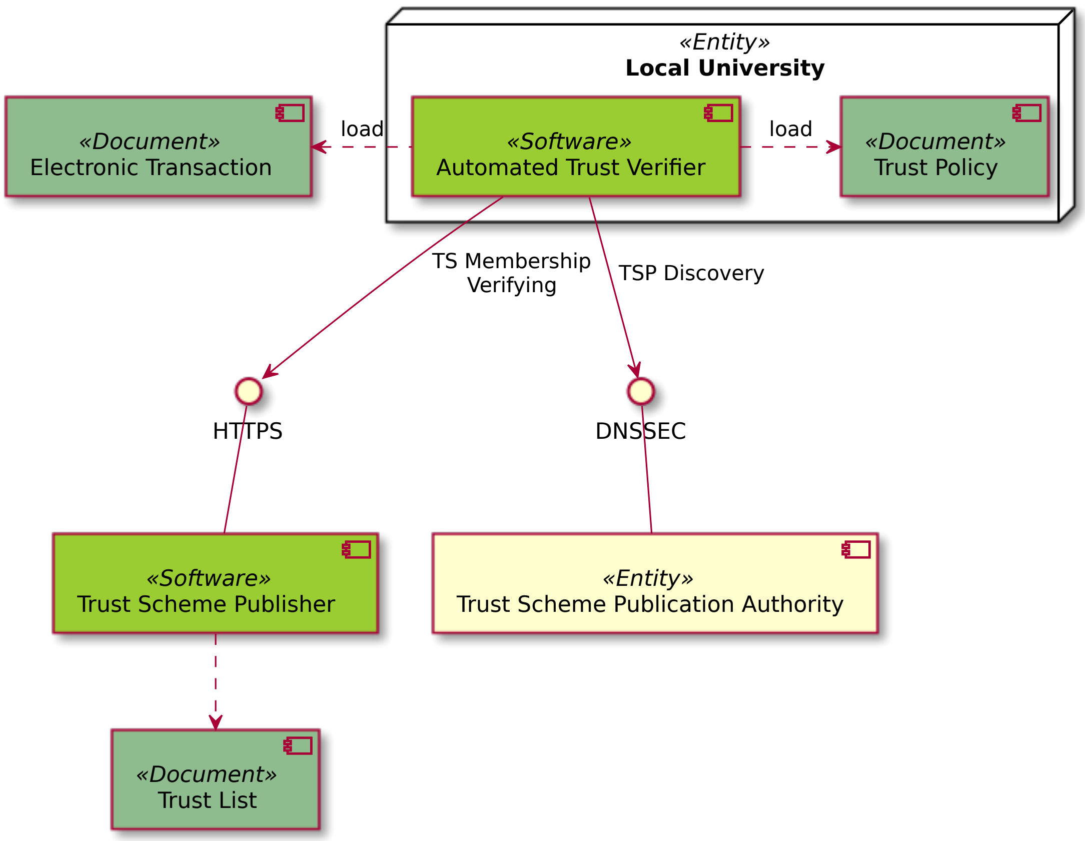


Figure 4.1.: Overview of the TIGHT^{est} architecture.

Summary: A student wants to register for a study program at our local university. To do so, they submit all requested documents in form of *Electronic Transactions*. Those electronic transactions are digitally signed by the university the student graduated from. To verify the authenticity of the documents, the local university uses a tool, the *Automated Trust Verifier*. This tool checks the X.509 certificate and signatures on the electronic transaction. An electronic transaction contains a *Trust Scheme Membership Claim*. The tool checks if the claimed trust scheme is part of the universities *Trust Policy*. To verify if the signing university is actually trustworthy/legitimate, the local university then verifies this trust scheme membership claim. The *Trust Scheme Publication Authority* provides an interface to verify this claim, the *Trust Scheme Publisher*. This interface is discovered by querying the Domain Name System (DNS) of the authority. In addition, the trust policy of the local university contains other rules the electronic transaction has to fulfill.

TIGHT^{est} aims to automate the answering of the following trust questions:

- *Did the issuing university really sign the transaction?*
  $\rightarrow$ Is the signature on the electronic transaction valid?
- *Is the issuing university a legitimate university?*
  $\rightarrow$ Is the issuer part of a trusted trust scheme?
- *Is the applicant eligible to apply?*
  $\rightarrow$ Does the electronic transaction fulfill the rules in the trust policy?

4.1.1. Electronic Transaction (ET)

TIGHT^{est} is verifying the authenticity of documents. A document is usually some sort of diploma. It contains the information which is validated against a Trust Policy (TP). It is digitally signed and accompanied by the certificate used to sign it.

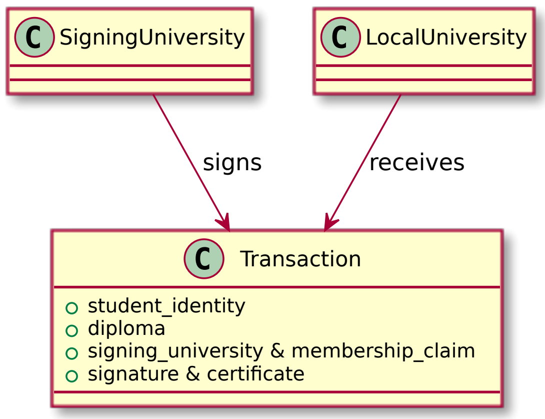


Figure 4.2.: Structure of an Electronic Transaction

An Electronic Transaction (ET) is the signed diploma together with its digital certificate. Furthermore, some metadata is attached. This metadata contains a Trust Scheme Membership Claim (TSMC). In addition, the digital identity of the student is stored in the transaction. This allows additional checks. For example, to ensure the diploma belongs to the applying student, the university could require that the student signs their application. This process is not covered by TIGHT^{est}. Figure 4.2 gives an overview of the structure of an ET.

An ET is a set of signed documents in *XML* format. Since an ET is exchanged between entities (universities), the format is not implementation specific. Listing 4.1 gives an overview of the format of an ET.

To obtain a message digest of a document, all members of the Secure Hash Algorithm (SHA) 2 family are supported. For signing the Rivest–Shamir–Adleman public-key cryptosystem (RSA), Digital Signature Algorithm (DSA) or Elliptic Curve Digital Signature Algorithm (ECDSA) can be used. Signatures are encoded in the W3C *XML Signature* format. Certificates used to sign transaction documents are represented in *X.509* format. Listing 4.2 gives an overview of the format of this signature data.

```
<diploma creator="tightest">

<student>
    <name>Alice Musterfrau</name>
    <country>Austria</country>
     <X509Certificate> ... </X509Certificate>
</student>

<university>
    <name>Graz University of Technology</name>
    <country>AT</country>
    <scheme>trust.academia.ec.eu</scheme>
</university>

<certification>
 <level>MSC</level>
 <studyProgram>ComputerScience</studyProgram>
</certification>

<!-- Signature Block -->
</diploma>
```

Listing 4.1: A sample electronic transaction: Data & Metadata

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
 <SignedInfo>
  <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n
      -20010315#WithComments"/>
  <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
  <Reference URI="">
   <Transforms>
    <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
        signature"/>
   </Transforms>
   <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
   <DigestValue> ... </DigestValue>
  </Reference>

 </SignedInfo>
 <SignatureValue> ... </SignatureValue>

 <KeyInfo Id="KeyInfo">
  <X509Data>
   <X509Certificate> ... </X509Certificate>
  </X509Data>
 </KeyInfo>
</Signature>
```

Listing 4.2: A sample electronic transaction signature & certificate

4.1.2. Trust Policy (TP)

An Automated Trust Verifier (ATV) follows the rules defined in a *Trust Policy (TP)*. Every university can define their own TP. It can define multiple TPs, for example for different study programs.

The main feature of a TP is to list the trust schemes relevant for the university. A trust scheme is identified by a human-readable domain name, which is part of the TP.

Furthermore, the TP can contain other rules which need to be fulfilled for a transaction to pass. The university can restrict the applicable study program the student has to have graduated in. It can also restrict the level of graduation (e.g. bachelor or master level).

A Trust Policy is represented by a simple, TIGHT^{est} specific scheme in *XML* format, called TIGHT^{est} *tinypolicy*. TIGHT^{est} tinypolicy is a simple policy format created for

TIGHT^{est} to demonstrate the verification process. It is not a generic policy format and tied to the use case of academic mobility. The tinypolicy format provides means to specify rules defining the required *level* of graduation, and *study program*. In addition, it allows to chain multiple rules, connecting them with logical *and* and *or* operators. Figure 4.3 gives an overview of the structure of a tinypolicy.

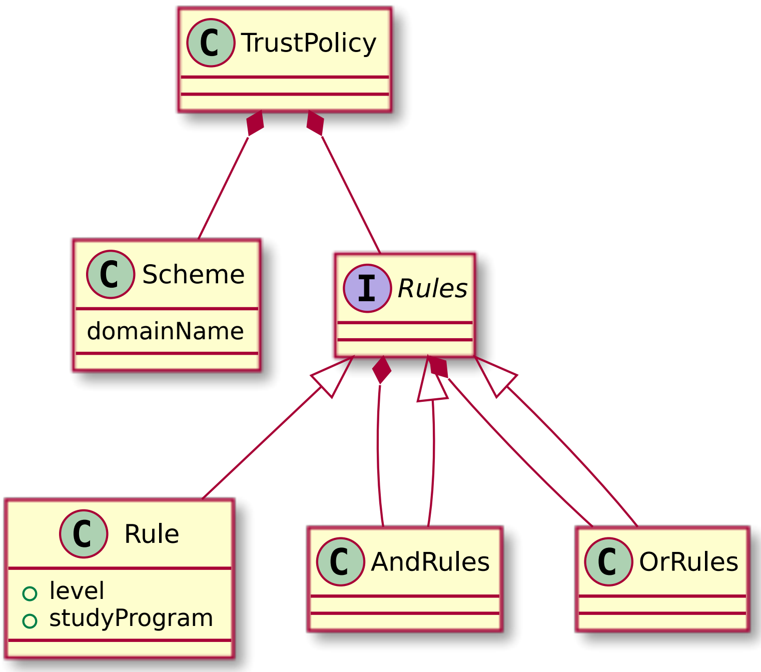


Figure 4.3.: Structure of a *tinypolicy* Trust Policy

Example policy: To apply for a PhD program in computer science, a student has to provide a graduation certificate (diploma) from a computer science or ICE masters program of a recognized university. In this context *recognized university* is defined by the membership in a trusted trust scheme. A trust scheme becomes trusted by being listed in the TP. This example is represented by the TP given in Listing 4.3.

```
<tinyPolicy creator="tightest" name="ComputerScience-PhD">

<trustedSchemes>
    <trustedScheme>trust.academia.ec.eu</trustedScheme>
    <trustedScheme>scheme.tightest.eu</trustedScheme>
</trustedSchemes>

<rules>
    <or>
        <rule>
            <level>MSC</level>
            <studyProgram>ComputerScience</studyProgram>
        </rule>
        <rule>
            <level>MSC</level>
            <studyProgram>ICE</studyProgram>
        </rule>
    </or>
</rules>

</tinyPolicy>
```

Listing 4.3: Example Trust Policy in *tinypolicy* format

4.1.3. Trust List (TL)

A Trust List (TL) is a list of trusted services in the scope of a specific trust scheme. It contains a list of X.509 certificates of those services. In addition, it may contain meta information about those services. For example, it can define the scope in which a trust service is valid. Another example is the validity period of a X.509 certificate, or revocations.

To validate an Trust Scheme Membership Claim, an ATV queries a Trust Scheme Publisher (TSP). As a result, the TSP returns the relevant segment of its TL.

A TL is represented in XML format. An example XML scheme is the *European Telecommunications Standards Institute (ETSI) TS 119 612* standard for *Trusted Lists* [1]. This allows the ATV to verify that the membership claim is valid at the specific time.

[1] http://www.etsi.org/deliver/etsi_ts/119600_119699/119612/

4.1.4. Automated Trust Verifier (ATV)

The software component executing all the checks and queries is called ATV. The reference implementation ATV is described in Section 5.1.

The ATV is a software operated at the university processing the student application. The ATV queries other components to verifying the authenticity of an ET.

The university configures its ATV by providing a TP. Different policies can be used for each check. This allows specific rules, depending on the nature of the check.

An ATV verifies the authenticity of an ET by checking the signature and the certificate. Furthermore, it verifies the TSMC by querying the Trust Scheme Publication Authority (TSPA) configured in its TP. If the TSPA confirms that the certificate used to sign the transaction is a member of its scheme, the membership claim is valid. In addition, the ATV performs other checks described in the TP.

4.1.5. Trust Scheme Publication Authority (TSPA)

To check if the university which signed the submitted ET is legitimate, the ATV needs to verify the *Trust Scheme Membership Claim* of the ET.

All entities are members of a trust scheme. The transactions issued by entities contain a claim which informs other parties about this membership.

To verify a TSMC, an ATV queries a TSPA. A TSPA is an institution recognized by all entities taking part in the process. It acts as a root of trust for a trust scheme.

A TSPA is identified by a human-readable domain name. To recognize the TSPA, a university operating an ATV needs to know the domain name of the TSPA it trusts. This is a human-readable identifier of the relevant trust scheme. The domain name of the trusted TSPA is then configured in the universities TP. If the domain name in the TP matched the one in the transaction, the claimed scheme is considered trusted.

Using the domain name in the transaction, an ATV queries the TSPA to discover the TSP. This is shown in Figure 4.4.

A TIGHT^{est} TSPA operates a DNS server to support discovery of its TSP. An ATV uses the domain name to request the pointer to the TSP from the DNS server. The DNS

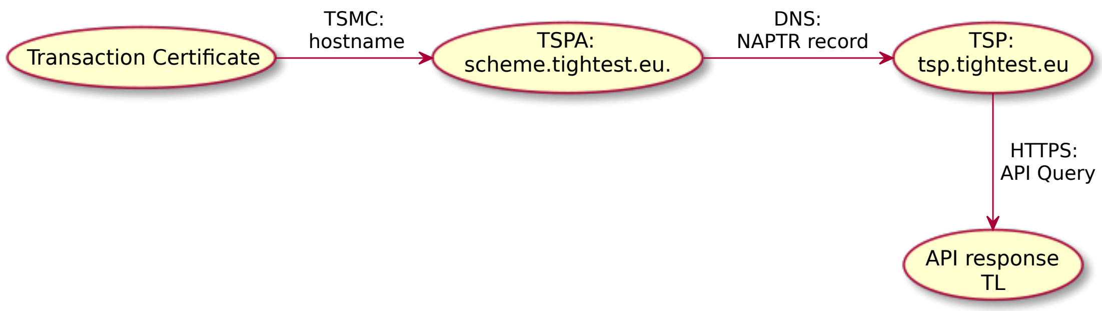


Figure 4.4.: Discovery of a TSP using a TSPA

server returns the requested pointer in form of a *NAPTR* record. The ATV resoles this *NAPTR* record to retrieve the TP location in form of a new new domain name. After doing so, it queries this new domain name to retrieve the location of the TSP HTTP API. The discovery is complete.

To build a chain of trust, the DNS server is *DNSSec* enabled. All responses are signed, and it is therefore possible to verify its authenticity using the configured root-zone keys. In addition, for the pointers to the Hypertext Transfer Protocol (HTTP) Application Programming Interface (API), the DNS server returns records authenticated by DNS-based Authentication of Named Entities (DANE). This extends the chain of trust from the DNS to the TSP HTTP API. The resulting trust path is shown in Figure 4.5.

4.1.6. Trust Scheme Publisher (TSP)

A TSP is a software operated by a TSPA. A TSP uses a *Trust List*. This is a list of the X.509 certificates of all legitimate universities in the trust scheme. This trust list is issued by the authority of the trust scheme. The reference implementation TSP is described in Section 5.3.

The TSP provides an API to query this list. A TSPA provides means to discover this API from its domain name. This API is then used by an ATV to verify a TSMC.

The ATV queries the API using the X.509 certificates fingerprint. The TSP returns the trust list section relevant for the requested X.509 certificate, if it is part of its trust scheme.

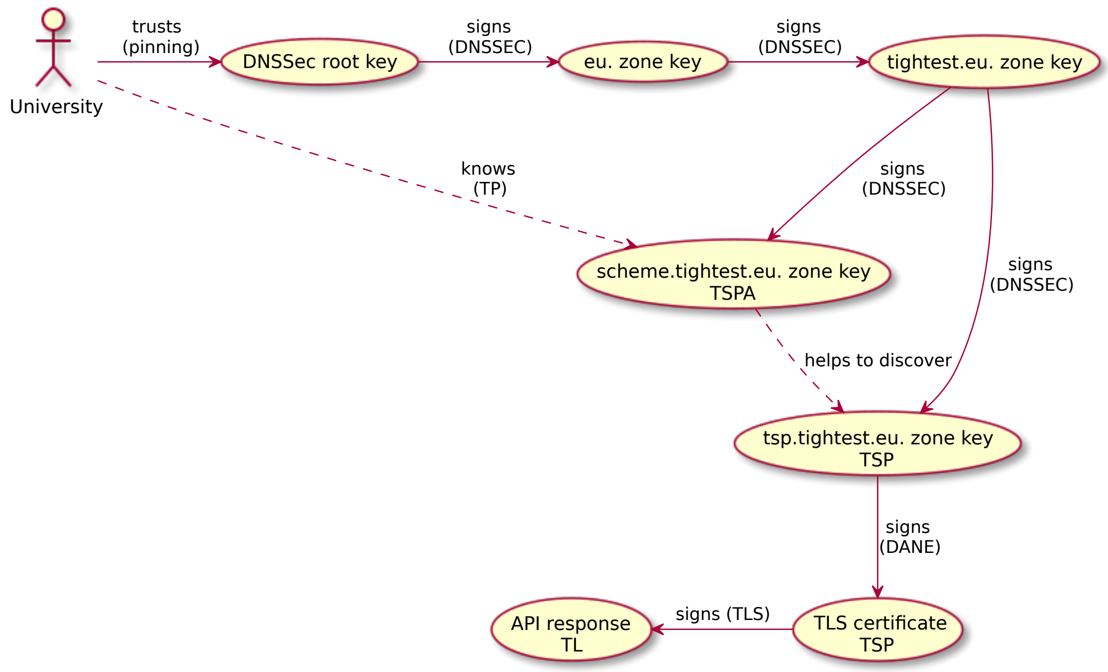


Figure 4.5.: TIGHT^{est} trust path from University to TSP API reponse

All transmissions are secured using the Transport Layer Security (TLS) protocol. The TLS connection is authenticated using a X.509 certificate. The ATV retrieves the fingerprint of the TLS certificate via DNS from the TSPA during discovery of the TSP (using DANE). It thereby extends the chain of trust from the TSPA to the data returned.

4.2. Actors

An **issuing university** issues and signs a diploma for a student. A **student** sends an application containing a diploma to a second university. An ATV is operated at the second university. The **person responsible for processing student applications** starts the verification process by loading a student application into the ATV. These actors are shown in Figure 4.6.

In addition to operating an ATV, the local university also needs to create and maintain a TP. This can be done by hand, or with any form of TP editor. It is possible to write a

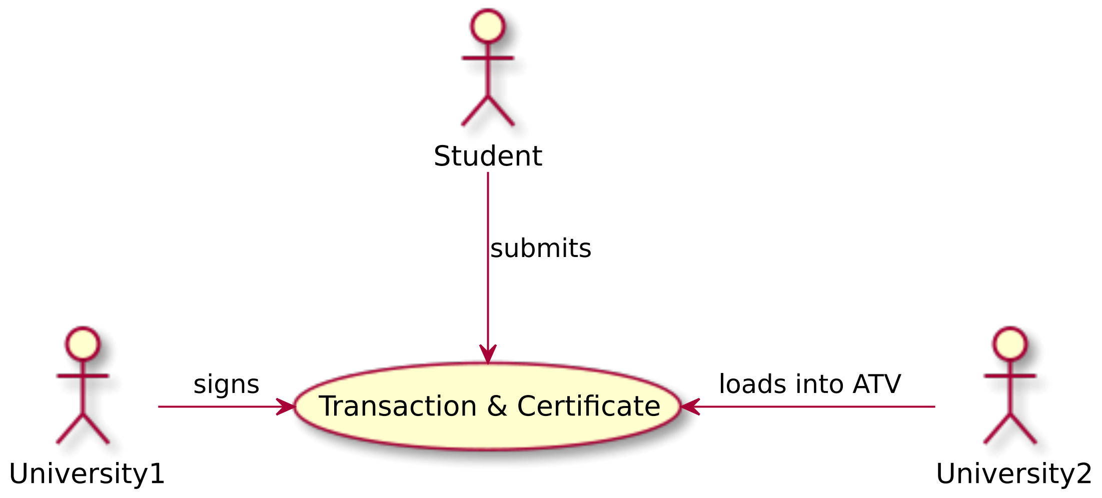


Figure 4.6.: Actors involved in a student application

TP in human-readable form and translate it to the *tinypolicy* XML format. This is not in the scope of this thesis. As part of creating an trust policy, the university also defines the trusted trust schemes by listing its TSPA domain names. This is shown in Figure 4.7.

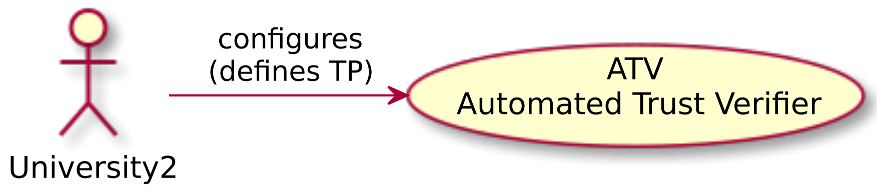


Figure 4.7.: Actor involved in configuration of an ATV

Since the TSP is a fully automated software operated by a TSPA, it does not require a human during operation. Setup of the TSP API and discovery mechanism only needs to be done once.

5. Reference Implementation

This chapter is concerned with the technical implementation of the TIGHT^{est} reference architecture. In addition, it gives an overview of the structure of a TIGHT^{est} transaction.

5.1. Automated Trust Verifier (ATV)

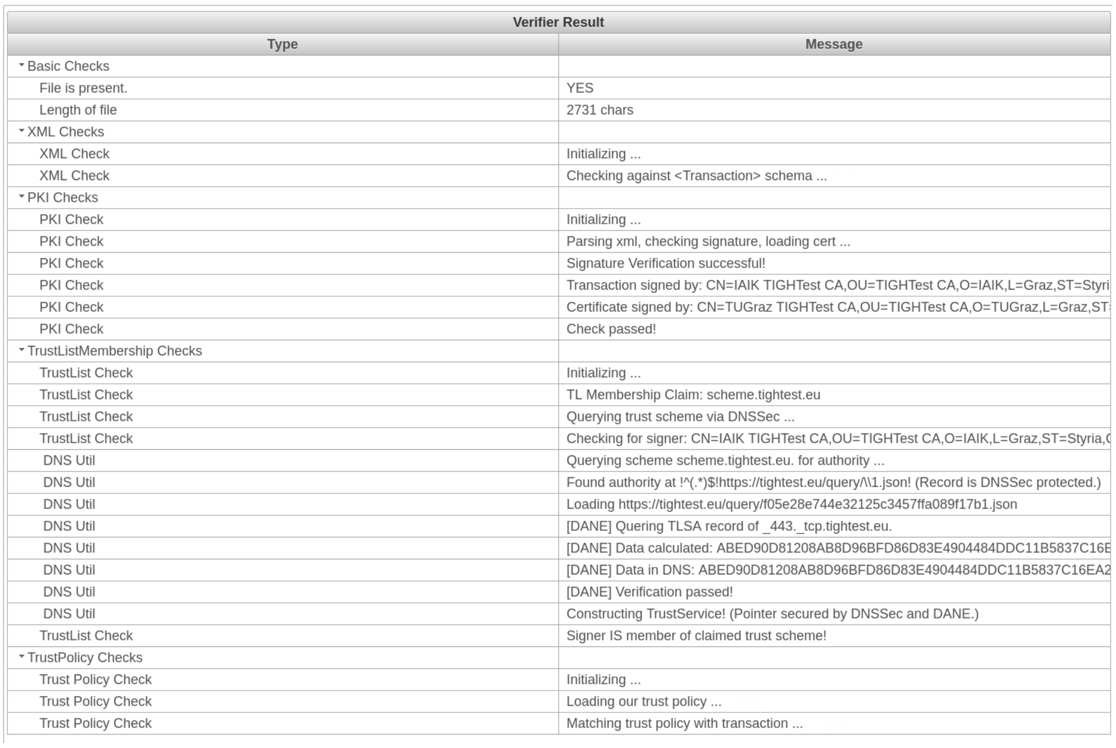
A TIGHT^{est} Automated Trust Verifier (ATV) is a web application developed using the *Java* programing language.

It consists of modules to:

- interact with users
- read Electronic Transactions (ETs)
- validate XML signatures
- read and understand Trust Policys (TPs)
- discover and query Trust Scheme Publishers (TSPs)
- execute and validate DNSSEC queries
- retrieve and understand Trust List (TL) segments

The ATV is hosted at a university webserver. As an alternative, TIGHT^{est} possible deployment options could to rent an ATV instance as a Software as a Service (SaaS).

Figure 5.1 shows the prototype implementation of an ATV graphical user interface. It displays the verification steps in the left column. In addition, the results of the verification of a sample transaction are displayed in the right column. Section 5.4 describes the steps in more detail.



Verifier Result

Type	Message
▾ Basic Checks	
File is present.	YES
Length of file	2731 chars
▾ XML Checks	
XML Check	Initializing ...
XML Check	Checking against <Transaction> schema ...
▾ PKI Checks	
PKI Check	Initializing ...
PKI Check	Parsing xml, checking signature, loading cert ...
PKI Check	Signature Verification successful!
PKI Check	Transaction signed by: CN=IAIK TIGHTest CA,OU=TIGHTest CA,O=IAIK,L=Graz,ST=Styri
PKI Check	Certificate signed by: CN=TUGraz TIGHTest CA,OU=TIGHTest CA,O=TUGraz,L=Graz,ST
PKI Check	Check passed!
▾ TrustListMembership Checks	
TrustList Check	Initializing ...
TrustList Check	TL Membership Claim: scheme.tightest.eu
TrustList Check	Querying trust scheme via DNSSec ...
TrustList Check	Checking for signer: CN=IAIK TIGHTest CA,OU=TIGHTest CA,O=IAIK,L=Graz,ST=Styria,
DNS Util	Querying scheme scheme.tightest.eu. for authority ...
DNS Util	Found authority at !^(.*)$!https://tightest.eu/query/\\1.json! (Record is DNSSec protected.)
DNS Util	Loading https://tightest.eu/query/f05e28e744e32125c3457ffa089f17b1.json
DNS Util	[DANE] Quering TLSA record of _443._tcp.tightest.eu.
DNS Util	[DANE] Data calculated: ABED90D81208AB8D96BFD86D83E4904484DDC11B5837C16E
DNS Util	[DANE] Data in DNS: ABED90D81208AB8D96BFD86D83E4904484DDC11B5837C16EA2
DNS Util	[DANE] Verification passed!
DNS Util	Constructing TrustService! (Pointer secured by DNSSec and DANE.)
TrustList Check	Signer IS member of claimed trust scheme!
▾ TrustPolicy Checks	
Trust Policy Check	Initializing ...
Trust Policy Check	Loading our trust policy ...
Trust Policy Check	Matching trust policy with transaction ...

Figure 5.1.: Screenshot of an Automated Trust Verifier prototype

5.2. Trust Scheme Publication Authority (TSPA)

The Trust Scheme Publication Authority (TSPA) is used by the ATV to discover the TSP Application Programming Interface (API) using the Domain Name System (DNS). In our reference implementation, we used the Name Server Daemon (NSD)[1] software to respond to DNS queries. In addition to responding to DNS queries by the ATV, NSD also serves signatures of the requested records.

To discover the TSP API, the ATV queries the TSPA for a Name Authority Pointer (NAPTR) record on the trust scheme domain configured in the transaction (and TP). The NAPTR record belonging to the TSPA zone contains a regular expression, as illustrated in Listing 5.1. This regular expression is used by the ATV to discover the TSP API. TSP discovery is explained in more detail in Section 4.1.5.

[1] https://www.nlnetlabs.nl/projects/nsd/

To build trust in the response to the ATV query, we use the Domain Name System Security Extensions (DNSSEC), as described in Section 2.5.2. The Resource Record digital Signature (RRSIG) record contains the signature of the NAPTR record, as illustrated in Listing 5.2. The record is therefore signed by the zones parent zone key. This authenticates the TSPAs response, as shown in Figure 4.5.

```
scheme.tightest.eu.      10      IN      NAPTR    100 10 "U" "tightest+
    trustscheme" "!^(.*)$!https://tightest.eu/query/\\1.json!" .
```

Listing 5.1: Example NAPTR record

```
scheme.tightest.eu.      10      IN      RRSIG    NAPTR 13 3 10 20171223022555
    20171125022555 65265 tightest.eu. 3
    QTvWK9wEWWtDE6BshLPscwGoCjQoaedcqULzl9DtAUCR ... nz94A==
```

Listing 5.2: Example RRSIG record for a NAPTR record

5.3. Trust Scheme Publisher (TSP)

A TIGHT^{est} TSP is a lightweight web API. Since it communicates with the ATV over a specified protocol, it is not necessary to develop the components in the same programming language. In our reference implementation we developed it using the *Python* programing language[2] to demonstrate this property.

We use the *Django* web framework[3] to implement the TSP. For storing trust list data we use *SQLite*[4]. Figure 5.2 shows the structure of the database. Our database schema supports the operation of multiple trust schemes with one TSP. In addition, it allows publishing of multiple trust services per trust scheme. Furthermore, a trust list section may contain multiple signing keys, for example for different departments.

Editing of endpoints and trust list data is done using the web interface generated by Django. To provide the API, we create a REST API using *Django REST framework*[5]. The Django application is hosted on a *nginx* webserver[6]. To secure the connection we

[2]https://www.python.org
[3]https://www.djangoproject.com
[4]https://www.sqlite.org
[5]http://www.django-rest-framework.org
[6]https://www.nginx.com

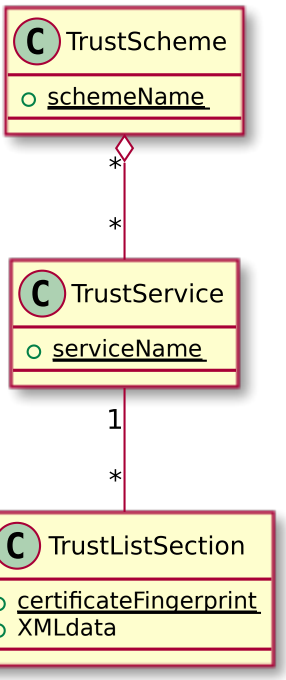


Figure 5.2.: Structure of the database used by the TSP reference implementation

use a *Let's Encrypt*[7] Transport Layer Security (TLS) certificate. For deployment and management of TLS certificates we use *Certbot*.

To transfer trust from the TSPA to the TSP API, we pin the TLS certificate in DNS using DNS-based Authentication of Named Entities (DANE), as illustrated in Listing 5.3. The Transport Layer Security Authentication (TLSA) record belonging to the TSP zone contains the TLS certificates fingerprint. The RRSIG record contains the signature of the TLSA record, signed by the parent zone key. Since the parent zone is authenticated by DNSSEC, this results in the trust path shown in Figure 4.5.

```
_443._tcp.tightest.eu.   10      IN      TLSA     3 1 2 99
    d6d27832c13622e50137fd42ca05b82a014f31f5f...
_443._tcp.tightest.eu.   10      IN      RRSIG    TLSA 13 4 10 20171223022555
    20171125022555 65265 tightest.eu.
    TG9zQ1RGe3RoaXNfaXNfYWxtb3NoX3N1cmVseV9hX2ZsYWd9Cg==
```

Listing 5.3: Example TLSA and RRSIG records

[7] https://letsencrypt.org

An ATV queries the TSP with a X.509 certificate fingerprint, as shown in Listing 5.4. This request is handled by Django and the REST framework. The TSP looks for the X.509 certificate matching the fingerprint in its database. If found, it returns an HTTP status code 200 (*OK*) and the corresponding trust list section. The format of a trust list is described in Section 4.1.3. If there is no entry for the fingerprint, an error is returned in the form of a HTTP status code. By returning an error, the TSP signals to the ATV that the requested X.509 certificate is not part of the trust scheme.

```
tspurl      = 'https://tightest.eu/query/'
fingerprint = 'afeee0d41695b98818df721d8e0fa47f47f8f9125'

queryurl    = tspurl + fingerprint
trustlist   = requests.get(queryurl)

if trustlist.http_status_code == 200: print 'trusted!'
```

Listing 5.4: Example (unauthenticated) query to the TSP API

5.4. The TIGHT^{est} Process

The components described in the earlier sections work together to answer a trust question. The ATV loads and parses a transaction and a trust policy. It then queries the other involved components for the required information. In the end the ATV verifies if the transaction is valid with regard to the TP. This is called *the TIGHT^{est} process*. Figure 1.3 gives an overview of the process.

This section describes the TIGHT^{est} process of our reference implementation step by step.

5.4.1. Initializing ATV

The ATV needs to be configured and started on its server. This needs to be done by an administrator of the university. Since the ATV operator accesses the ATV with a web browser, there is no need for them to install anything on their local office computer.

5.4.2. Load Trust Policy

If the ATV is initialized successfully, the operator uses a web browser to load a TP. The ATV then uses the TIGHT^{est} *tinypolicy* component to parse the policy and prepare a validator.

Another important step is the identification of the relevant trust scheme. The human-readable trust scheme identifier is part of the TP.

5.4.3. Load Transaction

The operator then loads a ET document into the ATV. The ATV uses the TIGHT^{est} *transactionutils* component to parse the transaction. During this step, the ATV verifies if the provided transaction document complies with the TIGHT^{est} ET XML scheme.

Furthermore, the Trust Scheme Membership Claim (TSMC) is retrieved from the ET. The claim is then compared with the trust scheme configured in the TP. If the trust schemes don't match, the process exits with an error. If the do match, the ATV continues with the process.

5.4.4. Signature Validation

As second check, the ATV retrieves the X.509 certificate used to sign the transaction. It then uses the contained key material to verify the XML signature.

This is done by the ATV using the *IAIK JCE* and *IAIK X-SECT* libraries for Java[8].

5.4.5. Validate Trust Membership Claim

In addition, the ATV checks whether the X.509 certificate used to sign the transaction is in the claimed trust scheme. It does so by validating the TSMC using the trust path shown in Figure 1.4. Doing this requires three steps.

- First, the ATV needs to identify the TSPA for the claimed trust scheme.

[8] https://jce.iaik.tugraz.at

- Next, the TSPA is used to discover the TSP API using TIGHT$^{\text{est}}$ *dnsutils* component.
- As last step, the ATV contacts the TSP API to retrieve a trust list segment.

This steps will be described in the following section:

The first step is done by loading the TSPA identifier from the trust scheme membership claim. This TSPA identifier is the same one as in the trust policy, as checked in the previous steps. The TSP identifier is a valid domain name. The ATV queries this domain name in the next step.

In the second step, the ATV uses the retrieved domain name to query the TSPA via DNS. To obtain a pointer to the TSP API, the ATV asks for a NAPTR record. The TSPA DNS server returns this NAPTR record via DNS. This NAPTR record contains a regular expression, used in the next step. To prove the authenticity of the requested data, the DNS records are protected using DNSSEC. The ATV verifies the authenticity by checking the signatures provided by DNS. This is done by traversing the DNS zones from the root zone down, as described in Chapter 2. Every zones key is signed by the key of the zone above. That way a trust chain is established from the (trusted) root zone to the TSPA zone. This extends trust to the retrieved regular expression.

In step three, the ATV evaluates the received regular expression to retrieve the TSP API HTTP URL. Afterwards, a connection to the retrieved HTTP URL is established. The connection to the TSP API is protected by TLS. In addition, the ATV queries the TSPA DNS server for TLSA records. This records are used in DANE, as described in Chapter 2. They contain information about the certificate used for the TLS connection. In addition, they are also signed using the zones DNSSEC key. Therefore the chain of trust is extended from the global DNSSEC root to the TLS connection.

If the TSMC is validated successfully, the authenticity of the transaction is shown.

5.4.6. Validate Trust Policy

The ATV concludes the process by checking if the transaction (Listing 4.1) fulfills the rules defined in the trust policy (Listing 4.3). This is done using the TIGHT$^{\text{est}}$ *tinypolicy* component.

5.4.7. Human Readable Results

There are many possible results of a TIGHT^{est} process. The process can exit with an error, if one validation step fails. It is also possible that the validation terminates without error, but the transaction does not comply with the trust policy.

In all cases the ATV provides a human-readable result in form of a receipt. Depending on which check terminated the process, different information is contained in the receipt.

The receipt is displayed to the user in compact form. Furthermore, it can for example be archived or used to communicate possible issues to the applying student.

5.5. Discussion & Limitations

The design of TIGHT^{est} and its reference implementation answer both research question stated in section 1.3.

TIGHT^{est}, therefore, shows that is it possible to automate the process of academic mobility. This is done by providing an infrastructure to validate electronic documents in an automated way. TIGHT^{est} has also shown that the Domain Name System (DNS) can be used to discover trust services. This provides an easy and human-readable identifier to identify those trust services. Using Domain Name System Security Extensions (DNSSEC) TIGHT^{est} is then able to use this identifier securely.

This results show that the technical aspects proposed in the LIGHT^{est} project are possible. For example, it is possible to use pointers in DNS for discovery. Besides, the transfer of trust from the (trusted) global DNSSEC root to an Trust Scheme Publication Authority (TSPA) API using DNS-based Authentication of Named Entities (DANE) is working.

Nevertheless, TIGHT^{est} is a research prototype. When it comes to practical application, it has its constraints. Also, the assumptions stated in Section 1.4 lead to some limitations. Many of those limitations can be worked on, as discussed in Chapter 6.

The main constraints is the requirement of a global entity trusted by all involved parties (the TSPA). This constraints leads to limitations in real-world scenarios since such an entity does not exist. The extensions proposed by the LIGHT^{est} project tackle those constraints on a technical level. They do so by proposing a mechanism of translations between trust schemes. This approach still requires a TSPA entity but removes the need for it to be globally trusted. It instead creates the need for local trust entities, which exist in many countries.

6. Future Work

TIGHT^{est} is a system to explore the processing of trust questions in the field of academic mobility in an automated way. In this thesis, we have shown that this is possible. For our research prototype, we worked under certain limitations and assumptions. In this chapter we discuss how those limitations can be tackled.

TIGHT^{est} is a research project based on the LIGHT^{est} project. The LIGHT^{est} project is introduced by Bruegger and Lipp, (2016) and in Chapter 3. The two projects share many properties and features. For example, LIGHT^{est} also uses the Domain Name System Security Extensions (DNSSEC) root as global trust root. But it extends the functionality even further.

In LIGHT^{est}, several concepts exist that keep the stated limitations in mind. One of the main goals of LIGHT^{est} is to establish a global trust system. In addition, it provides a generic system which can be used for many domains and use cases.

To explore technical aspects, TIGHT^{est} and this thesis focus on academic mobility. Thus, the underlying architecture and the reference implementation are very domain specific. In contrast, it is possible to use the applied principles for other use cases. One of the goals of LIGHT^{est} is to explore those possibilities in a more abstract way.

6.1. Trust Policy

tinypolicy is a tiny and lightweight policy format introduced for TIGHT^{est}. Since trust policies are not in the focus of the TIGHT^{est} research questions, *tinypolicy* is a very simplified format. Furthermore, it is domain specific to the academic mobility use case. In addition, it limits the set of possible rules to those required for the technical evaluation.

In contrast, there exist multiple policy languages, some of which are relevant to trust systems. Additionally, the LIGHT^{est} system is working on its own trust policy language.

Another topic of the LIGHT^{est} project is the editing of trust policies. A *tinypolicy* is represented by an XML-based text file, and can, therefore, be edited using a text editor. Since the concepts of policies used by LIGHT^{est} are more powerful than that, using a simple text editor is not very user-friendly. The project is, therefore, exploring ways of representing trust policies in a more human-readable way. An additional idea is the transformation of a trust policy between different representations. For example, it could be possible to represent a trust policy in English for editing by a human, but transfer it to a more abstract language for processing by the Automated Trust Verifier (ATV). Other ideas involve graphical representations and editing of a trust policy.

6.2. Trust Translation

The biggest issue of TIGHT^{est} is the need of a globally trusted entity, the Trust Scheme Publication Authority (TSPA). This limitation allowed an evaluation of the technical concepts, since lifting it does not change the requirements for trust service discovery. Using TIGHT^{est}, a student can only apply to a university in the same trust scheme as the university they graduated from. This results from the fact that a university cannot verify the membership of a trust scheme it has no trust in.

In LIGHT^{est}, this limitation is removed by introducing the concept of **Trust Translation**. Using this concept allows for a trust scheme to trust TIGHT^{est} trust scheme. Furthermore, it is possible to add constraints to such a translation. This enables that an entity in one trust scheme conditionally trusts an entity in a different trust scheme. This is again secured using DNSSEC.

For example the embassy of a country can be used to establish trust to TIGHT^{est} country, as shown in Figure 1.4. In addition, treaties between countries exist. It is possible to represent those treaties in an electronic way.

In simple terms, this adds a step to the trust path. The Domain Name System (DNS) related concepts explored in this thesis can, therefore, be applied for trust translations as well.

6.3. Delegation

In addition, LIGHT^{est} introduces the concept of a delegation. This allows different forms of mandates. In addition, it enables better representation of processes in the trust chain. For example, a university can delegate the permission to issue diplomas to specific employees. While this is also what happens in TIGHT^{est}, an employee would still use the university's X.509 certificate to issue signatures. In LIGHT^{est} an employee uses their personal X.509 certificate to sign the document. The university creates such a delegation by publishing the mandate in their *Delegation Publisher* (DP). Wagner, Omolola, and More, (2017) show a generic way of representing delegations. In LIGHT^{est} delegations are then published in a cryptographically secure way by using DNSSEC and DNS-based Authentication of Named Entities (DANE). Doing so creates additional steps in the chain of trust.

7. Conclusion

In this thesis we presented the TIGHT^{est} architecture, a system to automate trust decisions. We focused on applications concerning academic mobility. Also, we provided a reference implementation of the proposed architecture. By doing so we successfully demonstrated the effectiveness of the architecture. We showed that the DNS can be used to provide a chain of trust with a global trust root. This approach allows the creation of a human-readable and easy-to-understand way of building trust into an arbitrary transaction.

TIGHT^{est} uses a domain name as scheme identifier. This identifier is used to claim membership in a trust scheme. The claim is then understandable by humans and computers. TIGHT^{est} uses DNS together with its security extensions to verify this trust scheme membership claim in an automated way.

Nevertheless we recognize the challenges involved in the real-world deployment of such a system. Most of those challenges are not of technical nature and therefore out of the scope of this thesis.

We hope this thesis contributes to the field of automating trust decisions. We hope it sparks or feeds further discussions and efforts towards a (more) automated academic mobility. Besides, want to highlight the contributions of this thesis to the LIGHT^{est} project. By using and implementing many of its design ideas, we provided a proof of concept of its design. Furthermore, the knowledge and experience gathered during the development of TIGHT^{est} directly influenced the development of LIGHT^{est}.

Appendix

Bibliography

Barnes, R. (2011). *Use Cases and Requirements for DNS-Based Authentication of Named Entities (DANE)*. RFC 6394. RFC Editor (cit. on p. 22).

Blaze, Matt, John Ioannidis, and Angelos D. Keromytis (2003). "Experience with the KeyNote Trust Management System: Applications and Future Directions." In: *Trust management first international conference, iTrust 2003, Heraklion, Crete, Greece, May 28-30, 2003: proceedings*. Springer, pp. 284–300. DOI: 10.1007/3-540-44875-6_21 (cit. on p. 10).

Bruegger, Bud P. and Peter Lipp (2016). "LIGHTest –ALightweight Infrastructurefor Global Heterogeneous TrustManagement." In: *Lecture Notes in Informatics (LNI)* (cit. on pp. 25, 26, 49).

Cooper, D. et al. (2008). *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280. RFC Editor (cit. on p. 11).

Council of the Notariats of the European Union (2008). *Comparative Study on Authentic Instruments National Provisions of Private Law, Circulation, Mutual Recognition and Enforcement, Possible Legislative Initiative by the European Union*. Tech. rep. Council of the Notariats of the European Union (cit. on p. 1).

Dierks, T. and E. Rescorla (2008). *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246. RFC Editor (cit. on p. 12).

Eastlake, Donald E. (1999). *Domain Name System Security Extensions*. RFC 2535. RFC Editor (cit. on p. 17).

Hague Conference on Private International Law (1961). "12. Convention Abolishing the Requirement of Legalisation for Foreign Public Documents." In: October (cit. on p. 1).

Hoffman, P. and J. Schlyter (2012). *The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA*. RFC 6698. RFC Editor (cit. on p. 22).

LIGHTest Consortium (2017). “LIGHTest D2.3: Requirements and Use Cases.” In: (cit. on p. 9).

Mealling, M. and R. Daniel (2000). *The Naming Authority Pointer (NAPTR) DNS Resource Record*. RFC 2915. RFC Editor (cit. on p. 16).

Menezes, A. J. (Alfred J.), Paul C. Van Oorschot, and Scott A. Vanstone (1997). *Handbook of applied cryptography*. CRC Press, p. 780. ISBN: 0849385237 (cit. on p. 11).

Mockapetris, P. (1987a). *Domain names - concepts and facilities*. STD 13. RFC Editor (cit. on p. 13).

Mockapetris, P. (1987b). *Domain names - implementation and specification*. STD 13. RFC Editor (cit. on p. 13).

Postel, Jon (1994). *Domain Name System Structure and Delegation*. RFC 1591. RFC Editor (cit. on p. 13).

Solo, David et al. (2008). *XML Signature Syntax and Processing (Second Edition)*. W3C Recommendation. W3C (cit. on p. 24).

Wagner, Georg, Olamide Omolola, and Stefan More (2017). “Harmonizing Delegation Data Formats.” In: *Lecture Notes in Informatics*, pp. 25–34 (cit. on p. 51).

Appendix A.

Additional Figures

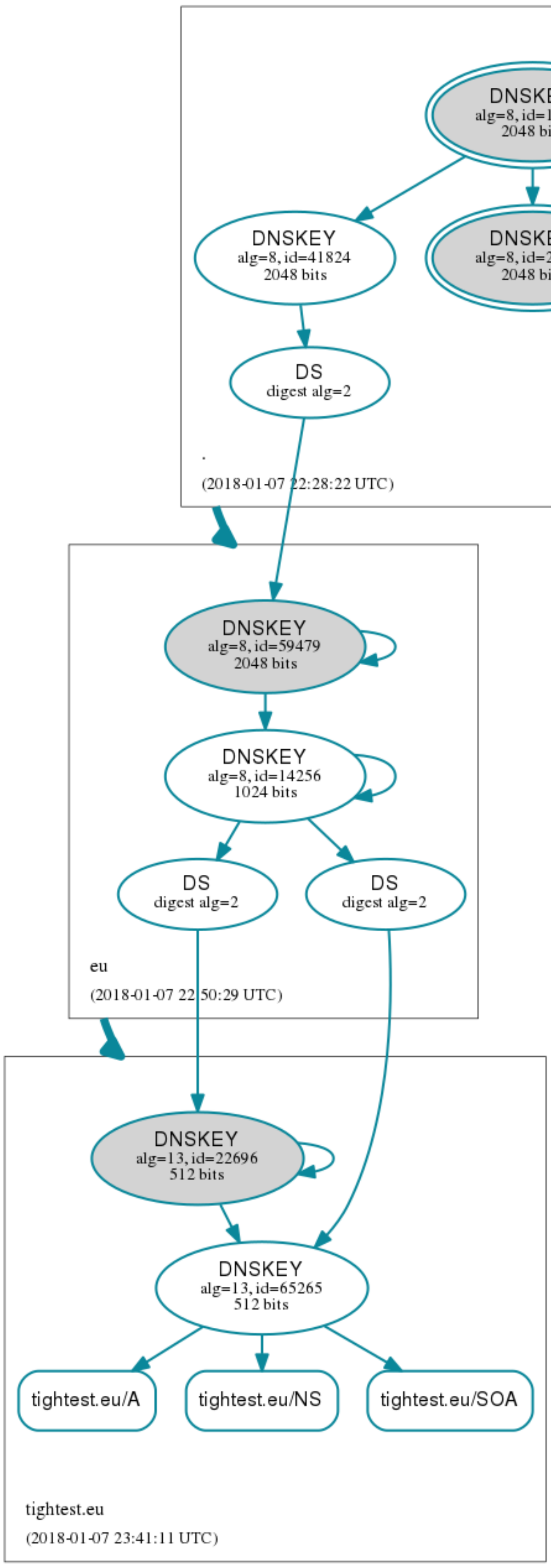


Figure A.1.: Full DNSSEC Authentication Chain for *tightest.eu.* `http://dnsviz.net/d/tightest.eu/dnssec/`, accessed 2018-01-07