D6.2

Requirements and Design of a Conceptual Framework for Trust Policies (2)

Document Identification	
Date	28.08.2018
Status	Final
Version	Version 0.3

Related WP	WP2, WP 3, WP 4, WP 5	**Related Deliverable(s)**	D2.1, D2.3, D6.1, D5.3, D5.4,D4.3, D4.4, D3.3, D3.4
Lead Authors	TUG	**Dissemination Level**	PU
Lead Participants	TUG	**Contributors**	USTUTT, FHG, ATOS, DTU, TUBITAK, OIX, GS, IBM
Reviewers	OIX, NLNET		

This document is issued within the frame and for the purpose of the LIGHT*est* project. LIGHT*est* has received funding from the European Union's Horizon 2020 research and innovation programme under G.A. No 700321.

1. Executive Summary

Trust policies are important for access control and verification services. They present some conditions that must be fulfilled before access or authorization is granted for a resource. In the context of LIGHTest, the trust policy presents a pre-defined set of conditions to verify electronic transactions. This document describes the basis of the trust policies in Lightest and how the trust policies are formulated for different scenarios. Since the Trust Policies need the Automatic Trust Verifier (ATV), a short description and the interaction of the ATV and other components are given.

In this deliverable, we discuss electronic transactions and the choice of our representation. We provide an insight into this representation and its operations.

Trust Policies are usually technical, and therefore the design and creation of a user-friendly tool are proposed to reduce the complexity of defining trust policies.

Chapter 5 introduces electronic transactions and its standardized representation. It explains the operations allowed on the representation.

Chapter 6 introduces trust policies and the logic behind its representation i.e., TPL. It also provides some examples of policies that different users in different sectors would write

Chapter 7 of this deliverable also explains interaction between the Automatic Trust Verifier (ATV), Trust Scheme Publication Authority, Trust Translation Authority and finally Delegation Publisher. This document describes the interaction clearly in order to provide a good basis for the development of the ATV.

2. Document Information

2.1 Contributors

Name	Partner
Omolola Olamide	TUG
Georg Wagner	TUG
Sven Wagner	FHG
Miguel Angel Rosa	ATOS
Martin Hoffman	NLNET
Miryam Villegas	ATOS
Javier Presa	ATOS
Miguel A. Mateo	ATOS
Muhammet Yildiz	TUBITAK
Edona Fasllija	TUBITAK
Elif Ustundag	TUBITAK
Sebastian Mödersheim	DTU
Rasmus Birkedal	DTU
Sue Dawes	OIX
Michelle Parks	OIX

2.2 History

Version	Date	Author	Changes
0.1	29/05/2018	Omolola Olamide	Document created
0.2	20/07/2018	Omolola Olamide, Georg Wagner, Sven Wagner, Miguel Mateo, Martin Hoffmann	Chapter 7 changes
0.3	28/08/2018	Omolola Olamide	Final changes

3. Table of Contents

1. Executive Summary 2

2. Document Information 3
2.1 Contributors 3
2.2 History 3

3. Table of Contents 4
3.1 Table of Figures 6
3.2 Table of Tables 6

4. Scope of deliverable 7

5. Electronic Transaction 8
5.1 Introduction to Electronic Transaction 8
5.2 Components of Electronic transactions 8
5.2.1 Electronic Signatures 9
5.2.2 Electronic Transaction Data 10
5.3 Requirements for electronic transaction trustworthiness 10
5.4 Trust Assurance for Electronic transactions 12
5.4.1 Types of ASiC Containers 13
5.4.2 Container Validation Process 14
5.4.3 Signature Validation Process 14

6. Trust Policy 16
6.1 Introduction to Trust Policy 16
6.2 Trust Policy Languages 16
6.2.1 Existing Trust Policy Languages 17
6.2.2 Proposed Trust Policy Language for LIGHTest 17
6.2.2.1 Policy Vocabulary Model 19
6.3 Data Formats for Trust Policy 21
6.4 Existing Trust Policy Authoring Tools 21
6.5 Proposed LIGHTest Trust Policy Authoring Tool 23
6.6 Trust Policy Templates 24
6.6.1 Business 24
6.6.1.1 Use Case (Purchasing Order 1): 24
6.6.1.2 Use Case (Purchasing Order 2): 24
6.6.1.3 Use Case (Correos MyMailbox): 25
6.6.1.4 Use Case (Correos MyVerifiedCommunications): 27
6.6.1.5 Use Case (**Credit Card**) 28
6.6.1.6 Use Case (Trusted Billing) 29
6.6.1.7 Use Case (**Notary**): 29

6.6.1.8 Use Case (**Cross Borders Banking**):.....30
6.6.2 Health 31
6.6.2.1 Use Case (Cross-border Health services) 31
6.6.3 Academics 32
6.6.3.1 Use Case 32

7. Automatic Trust Verifier 34
7.1 Introduction to Automatic Trust Verifier 34
7.2 Related Components 36
7.2.1 Trust Scheme Publication Authority 36
7.2.2 Trust Translation Authority 40
7.2.3 Delegation Publisher 42
7.2.3.1 Communication between the Delegation Provider and the ATV.....45

8. References 46

9. Project Description 49

3.1 Table of Figures

Figure 1 Electronic Transaction 8
Figure 2: Basic Validation Scheme, ETSI TS 319 102-1 v1.1.1 (2016-05) 15
Figure 3: Trust Policy 16
Figure 4 Structure of the Trust Policy Authoring Tool 23
Figure 5 Automatic Trust Verifier 34
Figure 6 ATV process flow 35
Figure 7: Sequence Diagram for Trust Publication of a Qualified Signature (Boolean) [2] 37
Figure 8: DNS queries in TTA (from D4.4) 41
Figure 9: steps 1 and 2 of Figure 7 42
Figure 10: steps 3 and 4 of Figure 7 42
Figure 12: Delegation Provider 43
Figure 14: Sequence Diagram for Trust Delegation Scenario (from [26]) 45

3.2 Table of Tables

Table 1: General Overview of Existing Policy Authoring Tools 21

4. Scope of deliverable

This deliverable defines concepts of electronic transactions and policies that describe the trustworthiness of the electronic transactions within the scope of LIGHTest. It provides the basis and the representation of electronic transaction. This deliverable also gives a concise description of the communication between the Automatic Trust Verifier and Trust Scheme Publisher, Trust Translation Authority, Delegation Provider.

5. Electronic Transaction

5.1 Introduction to Electronic Transaction

The following definition was taken from early versions of D2.2 [1]and D2.14 [2].

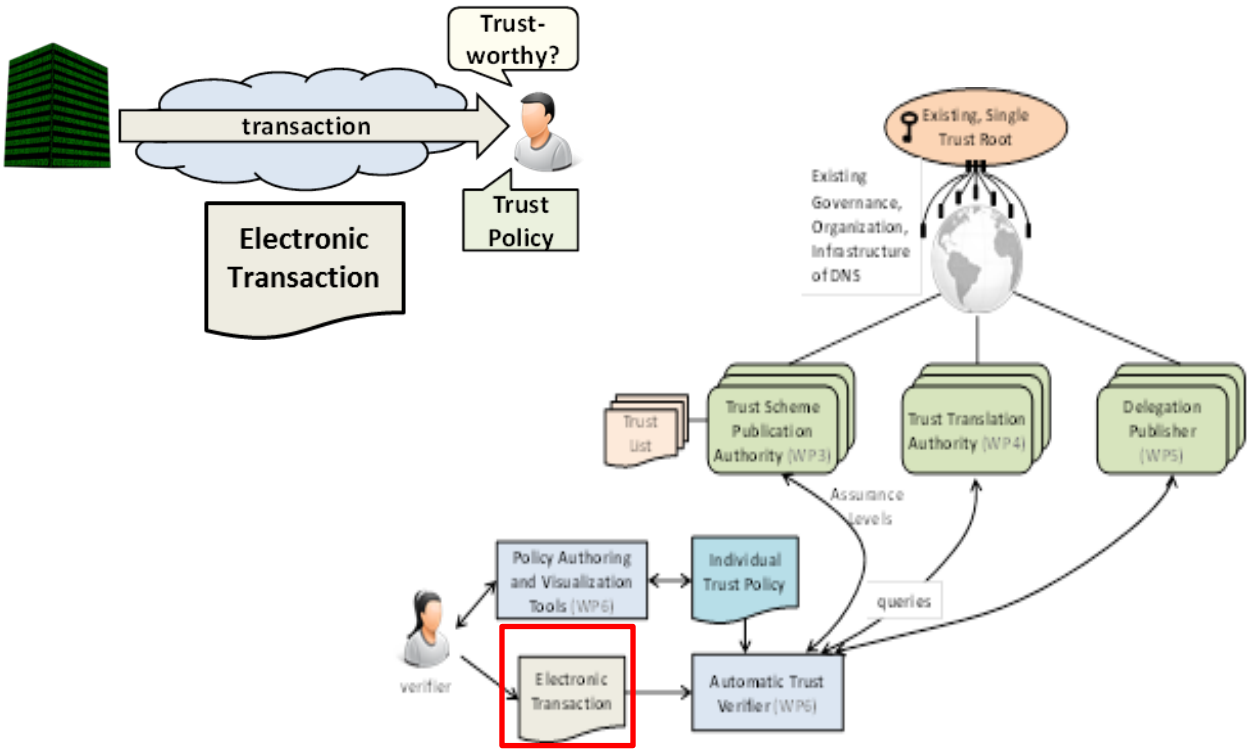

Figure 1 Electronic Transaction

An Electronic Transaction (see

Figure 1 Electronic Transaction) is something that we receive from a remote digital agent: it might be a digitally signed document, a multi-part Purchase Order, etc. Work Packages 6 and 9 will present more information and a selection of concrete Electronic Transactions that can be used to demonstrate LIGHTest.

5.2 Components of Electronic transactions

The most straightforward possible Electronic Transaction is a single document that is cryptographically associated with an electronic identity, e.g., through the mechanism of electronic signature.

In the more general case, an Electronic Transaction is a container (of a given format) that contains several documents or sub-containers. Optionally, documents and containers are associated with an electronic identity, e.g., via electronic signature.

An Electronic Transaction or the digital identities (e.g., certificates) associated with its parts can also contain "Discovery Data" such as membership claims that point to the Trust Schema that certifies its trustworthiness [2].

According to the OECD, an electronic transaction is:

"the sale or purchase of goods or services, whether between businesses, households, individuals, governments, and other public or private organisations, conducted over computer-mediated networks. The goods and services are ordered over those networks, but the payment and the ultimate delivery of the good or service may be conducted on or off-line." [3]

There are some ways that an electronic transaction can be carried out. Some of the ways are: between business entities, a business entity to a consumer, between people or between agencies. Electronic transactions do not, however, involve only sales of goods and services but could also refer to a process of authentication or authorisation between two or more entities. According to the Regulation of the Government of the Republic of Indonesia Concerning Electronic System and Transaction Operation, all electronic transactions should be in good faith, apply the prudence principle, have transparency, accountability and reasonableness. An electronic contract should contain identity data of the parties, objects and specifications, requirements of the electronic transactions, prices and costs, procedures in case of cancellation by the parties and electronic transaction completion legal options. [4]

On a fundamental level, there are some pre-requisites for a successful electronic transaction. These will include internet access, IT literate company employees and consumers, for e-commerce a banking institution with transaction clearing services. Electronic transactions also require a legal framework to back them up, whether they involve an e-commerce transaction or identity verification services. An authentication authority that serves as a trusted third party

ensures the integrity and security of transactions. These regulations are vital to protect consumers and businesses from fraud and ensure standards of privacy [5]. The main components of electronic transactions are discussed below

5.2.1 Electronic Signatures

One of the critical elements of a trustworthy electronic transaction is an electronic signature. There has been a lot of work done in this field and a strong focus on legislation globally over the last ten years.

According to the American Bar Association:

"Electronic signature" is a legal concept rather than a technology concept. An electronic signature is the electronic equivalent of a traditional manual signature placed on a piece of paper. In general, a signature creates binding rights and obligations when three things come together: a mark or symbol chosen to represent the signer (such as an autograph) is placed on a document to manifest the signer's intention to be legally bound. The same general requirements apply whether the signature is made with a pen on paper or through some electronic process. Because the legal requirements for a signature are very broad, there is an almost limitless number of different ways to create a signature, including many different technological processes.

Whether or not something that happens in the material world—such as placing an autograph on a document—produces a specific legal outcome is a question that can only be definitively resolved within the legal system, it can never be definitively resolved merely by proving that something happened. The term "electronic signature" is used to describe the fact that a signature process has occurred and the legal outcome that someone is bound to comply with the terms contained in a document. As a result, the two different concepts are often confused. The confusion of those two ideas has been a source of controversy ever since the concept of "electronic signature" came into widespread use." [6]

This strong focus on electronic signature legislation led to the eIDAS regulations that came into effect in July 2014. eIDAS provides the regulation on electronic transactions in the EU, and under their regulation, an electronic signature is any data in electronic form that can serve as a method of authentication. eIDAS has made cross-border electronic transactions more secure and trustworthy. There are three different types of electronic signatures:

1. Standard electronic signature – any form of verification with evidence
2. Advanced electronic signature – the data (i.e., the private key) used to generate the digital signature is in the sole control of the signer
3. Qualified electronic signature – the certificated identifying the digital signature comes from a company on the qualified provider list [7]. This certificate must have been issued following the rules and procedures laid out in the regulation (or possibly the accompanying legislation) thereby forming the basis for elevating the signatures to be legally binding.

5.2.2 Electronic Transaction Data

Another critical component of Electronic transactions is the data sent in the process of the transaction. The data could be a description of the transaction itself, i.e., the metadata of the transaction. The data could also be other necessary information determining how the transactions are to be handled or evaluated. The data can also be evidence that an action has happened. The electronic transaction data could be in various electronic forms either document formats (.pdf, .doc, .xml) which goes on to say that the representation can take different forms i.e. human-readable and machine-readable forms.

5.3 Requirements for electronic transaction trustworthiness

Trust is defined as “Trust is the extent to which one party is willing to depend on somebody, or something, in a given situation with a feeling of relative security, even though negative consequences are possible.” [8]. Trust has been studied and found to be subjective, non-reciprocal and non-transitive. Subjective in the sense that it is “an opinion an entity holds of another entity” and non-reciprocal in the sense that “an opinion or level of trust that entity A has for entity B is not the same level that entity B has for entity A”. Transitive means that “the level of trust an entity A holds for entity B which is the same level that entity B has for entity C does not mean that entity A trusts entity C at that same level. ”

Transactions have some features that have to be fulfilled before it can be trusted. These features are as follows [8]:

1. Identification and authentication
2. Message confidentiality
3. Message integrity
4. Non-repudiation
5. Transparent transaction process
6. Traceability and accountability

Transactions done electronically have become commonplace. They are usually faster and more comfortable than the other means of transactions, but it also has to fulfil all the features listed above before it can be trusted. In contrast to traditional ways of carrying out transactions, these features are harder to achieve in electronic transactions. The parties carrying out the transaction are often unable to confirm the identity of the person (natural or legal) or device they are performing the transaction with or verify the message integrity.

A lot of research has gone into ensuring these features are fulfilled and increase trust in transactions carried out electronically. Some of the means of confirming the trustworthiness of electronic transactions presently are as follows:

- Electronic Identifications (eID) – This form of identification allows for the identification of a person behind a transaction or a request for a service. This identification is especially necessary to prevent online fraud and ensure that the person benefits from data legislation laws. eIDs were limited in usage to the borders of the issuing country. This need encouraged the European Union to create the eIDAS regulation, which created

interoperability of eIDs among European Union member states. The eIDAS regulation also introduced some other trust services, which validate electronic transactions. They include:

- Electronic Signature – This is the electronic form of a written signature (not in representation but concept)
- Time stamping – This is the date and time on a document which proves it existed at a point in time and the document has not changed since then
- Electronic Seal – Similar to physical seals. Seals show the integrity and origin of the document as long as the seal is not broken
- Electronic registered delivery – Similar to the registered post
- Website Authentication [9]

- Public key infrastructure (PKI): PKIs provide a system for achieving trust in transactions over the internet [8]. These systems use digital certificates to distribute Public Keys and other pertinent data about the owner of the certificate. Digital Certificates are Public Key Certificates issued to the owner of a particular public key. A certification authority (CA) issues the certificates, and it includes information about the owner and the public key. The certificate is often based on the X.509 format. The Public key found is used to encrypt confidential report by the entity sending the information. The receiver/owner of the certificate decrypts the information with his private key.
- Authorization schemes such as ABAC (Attribute Based Access Control) [10], RBAC (Role Based Access Control) [11] are also in use to ensure the right person has access to the right resource and no more.
- Electronic Transaction protocols: Many Electronic transactions have been developed, and a good example is Secure Electronic Transactions (SET). SET is an open protocol proposed by Visa and MasterCard. It uses both symmetric (DES) and asymmetric cryptography (RSA) to secure transactions and prove its reliability [12]. Extensions have been proposed to SET [13] [14] [15], and this protocol is often used in financial transactions.

Many participants in electronic transactions require one or more of these means to validate the integrity of the electronic transaction. Any electronic transaction that fulfils the above requirements is deemed trustworthy.

5.4 Trust Assurance for Electronic transactions

An *electronic transaction* in the general sense is considered to be a **container** that incorporates multiple parts. These containers associate parts/files containing data objects with cryptographic elements such as digital signatures/time assertions. Containers allow for more natural interchange and help guarantee that the correct signature, in addition to its metadata, is used during the validation process. The same concept also applies to associate time assertions, such as time-stamp tokens or evidence records to their data objects (parts). Electronic transactions can range from simple to complex.

A simple electronic transaction can be thought to be a container that associates a single document to an electronic identity, through a digital signature.

A complex electronic transaction, in contrast, can be considered to be a container or a given format that contains several documents or sub-containers, whose trustworthiness evaluation possibly requires multiple trust transactions (trust validation processes).

To facilitate interoperability and provide trust and confidence in electronic transactions, electronic transactions need a standard (standardised) way for performing the association between the data object and digital signature parts.

ASiC (Associated Signature Container) [16] provides a standard for container types for packaging and associating the data and cryptographic parts of an electronic transaction.

An ASiC container is a data container that incorporates a group of file objects and their associated digital signatures/and or time assertions using the ZIP format.

The internal structure of an ASiC container includes the following:

- A **root** folder for all the container content, including folders that reflect the structure of the content
- A "**META-INF**" folder inside the above-mentioned root folder that holds files that contain metadata about the content, including the associated signature/and or time assertion files

The associated signature/and or time assertion files that are contained in the "META-INF" folder are placed in such a way that they can be verified against their associated file objects. There is a standard container validation process, which will be discussed later in the chapter.

The signature file will hold either a single Cryptographic Message Syntax Advanced Electronic Signature (CAdES) or one or more XML Advanced Electronic Signature (XAdES). Quoting from [17]

"CAdES is built around the Cryptographic Message Syntax (CMS), a basic building block for digital signatures based on standard public key infrastructure (PKI) principles. CAdES adds to CMS an infrastructure for a set of increasingly ambitious standards for digital signatures that can be applied to any kind of digital data".

"XAdES is built around the XML Digital Signatures (XML-DSIG) standard for digital signatures represented in XML. (Note that these signatures can be applied to any kind of data, not just XML data.) Like CAdES, XAdES-BES defines Basic Electronic Signatures and XAdES-EPES defines Explicit Policy Electronic Signatures. XAdES also adds the same LTVchecking to XML-DSIG as CAdES does to CMS".

The time assertion file will hold either a one time-stamp token that conforms to IETF RFC 3161 [18] or a single evidence record that conforms to IETF RFC 4998 [19]or IETF RFC 6283 [20].

The concepts of simple and complex-type electronic definitions can easily be mapped to two types of ASiC Containers, namely the ASiC-S and ASiC-E containers, respectively. The details of these ASiC containers types are given in the following section.

5.4.1 Types of ASiC Containers

The *ASiC Simple (ASiC-S)* container associates one file object with either a signature file or a time assertion file. A file named "mime type," which specifies the media type may also be included in this container. This container type allows additional signatures to be added at a later date to sign the stored file object and add ASiC ArchiveManifest files that will protect long-term time-stamp tokens.

The *ASiC Extended (ASiC-E*) container associates one or more signature or time assertion files that apply to their own sets of file objects. Each file object can have additional information or metadata associated to it that can be protected by the signature. This type of container can be designed to restrict this modification or to allow their inclusion without damaging the previous signatures.

As it can be seen from the definitions given above, the ASiC-S container is a good fit for the concept of a simple electronic transaction, and the ASiC-E container for the complex-type electronic transaction.

5.4.2 Container Validation Process

The steps of validating and verifying an ASiC container can be summarized as follows:

1. The Container format is checked for compliance with ISO/IEC 21320-1 standard.
2. The Container type identification takes place (deciding whether a container is of ASiC-S or ASiC-E type)
3. The media-type that the Container holds is identified (for extracting the container file extension)
4. The Container's mime-type file existence is checked (may be present in the container)
5. Container data file existence check (at least one data object shall be present in the container)
6. The Container META-INF folder existence is checked (shall be present)
7. Container signed data object existence is checked (at least one signed data object shall be present in the META-INF folder)

After these validation steps concerning mostly the structure of the container, its elements are extracted to initiate the signature validation process.

5.4.3 Signature Validation Process

The Signature Validation Process comprises a signature validation application (SVA) and Signature Validation Inputs such as the signature validation policy and the signature itself. The SVA validates the signature against a signature validation policy and outputs a status indication and validation report.

SVA includes the following building blocks to facilitate the validation process:

Format Checking: This building blocks checks that the signature is conformant to the applicable base format such that it can be processed by the cryptographic verification building block. The necessary input for this building block is Signed Data Object.

Identification of the signing certificate: This building block identifies the signing certificate that will be used to validate the signature. The signature element is mandatory, and signing certificate itself is optional.

Validation context initialisation: This building block initialises the validation constraints (X.509 validation constraints, cryptographic constraints, signature elements constraints) and parameters (X.509 validation parameters including trust anchors and certificate validation data). The building block needs necessary Signature element and optional Signature Validation Policies, Trust anchor list (e.g. TSL), and local configuration elements.

Revocation freshness checker: This building block checks that a given revocation status information is "fresh" at a given validation time. The mandatory elements for this building block are Revocation status information, the certificate for which the revocation is being checked, Validation time and X.509 validation constraints.

X.509 certificate validation: This building block validates the signing certificate at validation time or at the current time depending on the validation time provision. Signing certificate, X.509 Validation Constraints, Trust Anchors and revocation status from the revocation freshness checker are mandatory while Validation time, Certificate Validation Data, X.509 Validation Parameters and Cryptographic Constraints are optional for this building block.

Cryptographic verification: The actual integrity of the signed data is checked in here by performing the cryptographic verifications. Signature, the signed document and Signing Certificate are mandatory to facilitate cryptographic checks while validated certificate chain is optional.

The Basic Signature Validation process is depicted in Figure 2. using building blocks mentioned above.

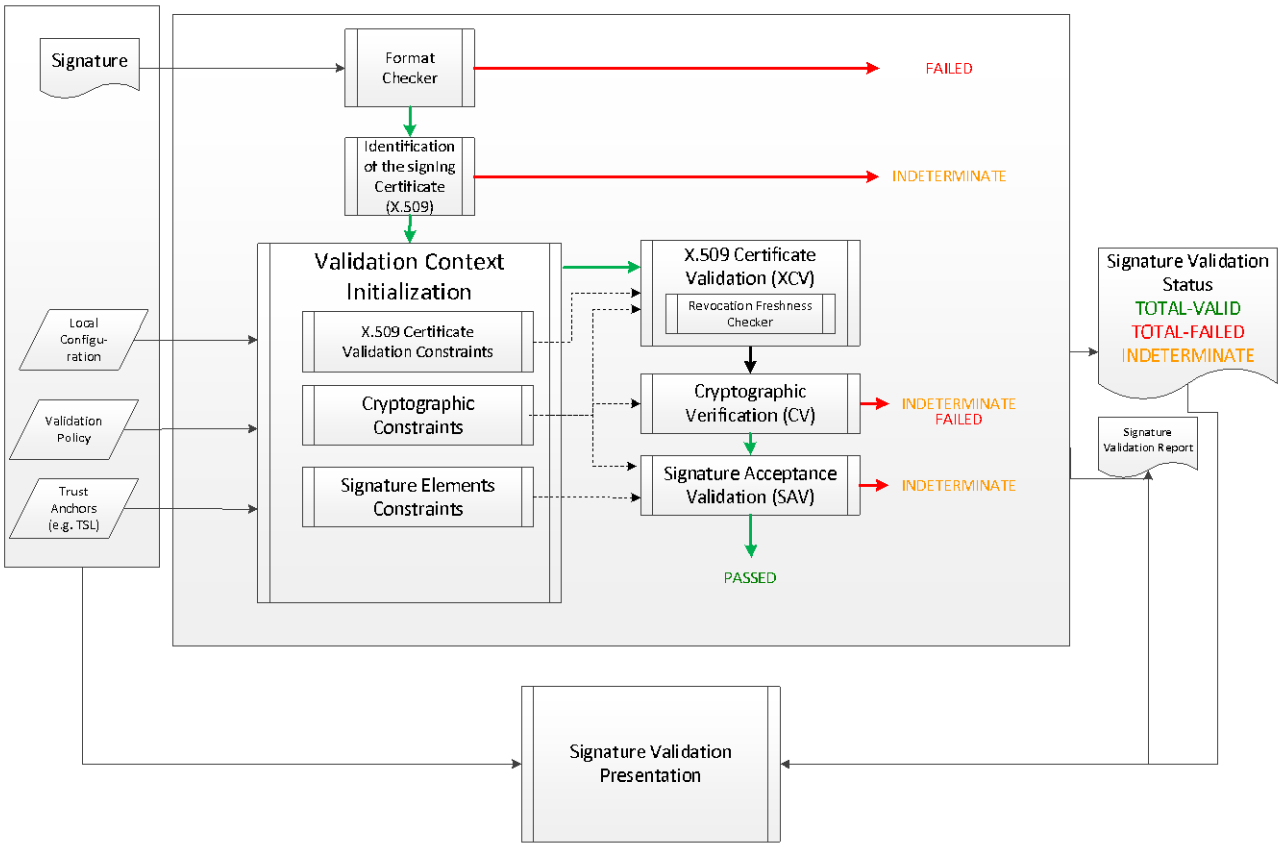

Figure 2: Basic Validation Scheme, ETSI TS 319 102-1 v1.1.1 (2016-05)

6. Trust Policy

6.1 Introduction to Trust Policy

The following definition was taken from early versions of D2.2 [1]and D2.14 [2].

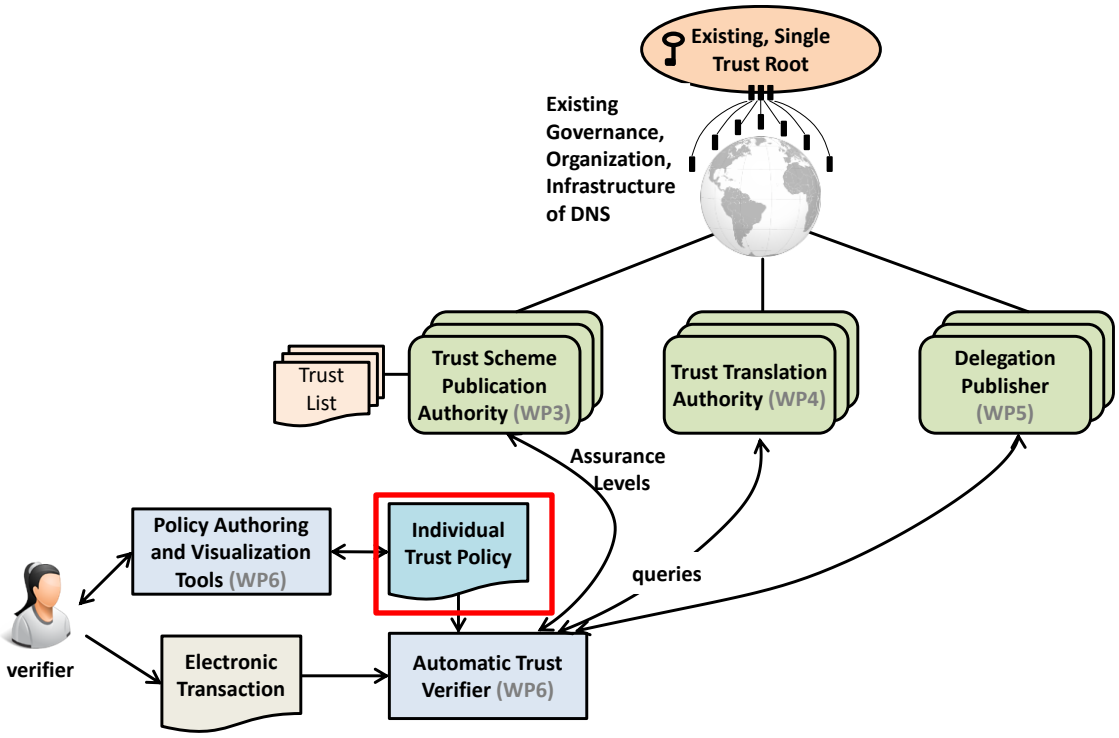

Figure 3: Trust Policy

A Trust Policy is a recipe, expressed in a Trust Policy Language, that takes an Electronic Transaction and potentially multiple Trust Schemes, Trust Translation Schemes and Delegation Schemes as input and creates a single Boolean value (trusted [y/n]) and optionally an explanation (e.g., why not trusted) as output. In LIGHTest, a trust policy is evaluated by the Automatic Trust Verifier component.

6.2 Trust Policy Languages

A Trust Policy Language is a formal language with well-defined semantics that is based on a mathematical formalism and is used to express the recipe of a trust policy.

6.2.1 Existing Trust Policy Languages

Many trust policies that are used in Industry today are of a rather simple nature, for instance, trust lists, especially when using somewhat private, company-specific solutions. For transactions across such islands, languages for managing trust are needed, and it was recognised that the problem of trust has many similarities to the issue of access control [21]. Therefore several trust languages have been proposed that borrow from access control management frameworks like XACML and OASIS, for instance [21] [22] [23].

Some of the first languages are somewhat like a programming language where one can for instance program filters to represent trust management. Since this is slightly technical and low-level, several languages have been proposed that move away from the programming language flavour. The proposed languages focus on a more logical (as opposed to operational) view of languages, describing properties of certificates. These proposed languages gives a set of specific construct for trust management like requiring a particular number of certificates for a favourable trust decision (e.g. for becoming a member of a club, one needs the support of two existing members) or blacklists for negative decisions. The logical flavour is emphasised for instance the description of [23] explicitly refers to way Prolog uses variables in conditions.

6.2.2 Proposed Trust Policy Language for LIGHTest

We have been inspired by these approaches and the fact that many policy languages for access control are based on logical languages and in particular Horn clauses, i.e., rules for the form

Conclusion: - Premises.

So that a conclusion (e.g. that an entity is trusted or gets access) is true if all of the premises hold. Both conclusion and premises can contain variables, and the scope of each variable is the rule it occurs in so that several occurrences of the same variable in a rule means that it must be the same value. The semantics of this is that the conclusion holds for every instantiation of the variables that makes the premises right. This allows us to describe for instance that an entity is trusted if specific certificates/signatures are present and certain conditions about the credentials' contents (issuer, receiver, attributes) are met. This can be combined so that premises can themselves be obtained as the conclusion of another rule application, e.g. the policy that one is trusted if one either satisfies certain conditions or is the delegatee of somebody who is already trusted.

We propose therefore to use for our trust policy language LIGHTest TPL (or just TPL in the following when there is no danger of confusion with the paper [21]) merely the logic programming language Prolog that is based on only these Horn clauses. Concerning languages like [21] or [23] we thus open the language to be not limited to a few trust-specific constructs with the burden to extend the language whenever we run into a concept that is not expressible: since Prolog is Turing complete, we can specify any (Turing-) computable policy. Another considerable advantage of this is that the machinery for evaluating policies is already there, as Prolog is a well-understood language with excellent tool support. As an example, for accountability, one may want policy decision points to document (log) how they made their choices, e.g. what input certificates they

have seen, what information they obtained from a server, and how their decision then followed from this input logically from the rules of the policy. While this is just an everyday "book-keeping" task, the use of Prolog allows for a generic way to implement this, as all steps of a policy decision point are thus logical derivation steps that can be written down.

We see also a clear difference to those of the previous approaches that have a programming language flavour: our language is based on logic (programming) language. For what concerns the specification of policies, we do not plan to use any of the more procedural aspects of Prolog (like the cut operator to restrict the search for conclusions) except for one well-defined use: namely the not-operator for checking against membership in a black-list.

Of course, the use of a full-fledged programming language allows us to also describe more procedural or operational aspects, for instance, when to issue a query to a particular server to look up a certificate or a trust list membership on that server. These are not logical operations, but their result can be used in a trust policy by specifying a logical predicate for this result. In this way, we can define more than just the trust policy, but also the necessary operational aspects when to look something up and where, but we can at the same time separate logical and procedural aspects and thereby have clear and clean specifications.

It is possible to embed many existing policy languages into LIGHTest TPL: one can translate their rules often without much effort into a set of Prolog rules in size linear to the original. This ability allows us to connect to LIGHTest all kinds of restricted but more simple policy languages -- such as for instances graphical languages -- that may be much easier to use but on the other hand, have not such high expressive power. This is done by defining a translation from that new language into TPL, and immediately all tools that work on TPL are available.

However, one downside of having such an expressive language (although it is based on a minimal set of concepts) is that users may divert upon using it and that general tool support for it could become difficult. For this reason we will have the following more structuring elements in TPL:

- For policies we recommend to stay within the *Datalog* fragment: The Datalog fragment is the fragment of first-order Horn clauses without function symbols; this has the advantage of being decidable and better accessible for automation in general. For TPL, we can actually allow some of the built-in function symbols like addition, e.g. constraints like X+15<=Y.
- We define a vocabulary and design patterns that have proved sufficient for case studies. These vocabulary and design patterns are represented as a kind of "paved road system" for the language, and whenever one feels the need to leave these paved roads, we (in the project for now) discuss if this is the best way to do it and if so extend the design patterns accordingly. Our initial setting is described in Section6.2.2.1.
- The vocabulary also gives an interface to the data-formats of certificates and the like, so that we can separate between concrete and abstract syntax and have the connection between them automated, as explained in Section 6.3.

A more formal introduction of TPL is found in Deliverable D2.4.

6.2.2.1 Policy Vocabulary Model

This section discusses some of the most important constructs that will be used for the creation of trust policy documents.

extract

We provide a kind of library for the most important paths or design patterns in policy specifications for LIGHTest. First, we shall have a generic interface for accessing the contents of certificates, signatures, and formatted clear-text -- independent from the concrete syntax. For this reason, we have the general predicate of arity 3:

extract(Document, Fieldname, Value)

This example assumes that the given Document can be regarded as a list attribute-value pair, i.e., like a form where each field has a name and can be filled with a particular value. For instance, a public-key certificate can be a document that has fields like "issuer" and "holder" of type "DistinguishedName", fields "issuer_public_key" and "holder_public_key" of type "PublicKey", field "expiration" of type "Date" and so forth. We here say here "of type..." to indicate that the Value that the extraction will yield is of said type. The types of available fields and their types depend on the document type. To make our mechanism applicable to data of arbitrary formats we provide a *virtual* attribute "format", thus with extract(Document, format,x_509_certificate) one could require that a given document is an x509 certificate. Since Prolog itself does not have types, we plan to introduce a type system in TPL that checks (at design time) that a trust policy only tries to extract fields of a document that are actually defined in that type of document, while without this specification, this undefined fields would result in a run-time error. Such a run-time error will lead to the failure of a policy decision, and this needs to be *mathematically strict* in the following sense: if a predicate (e.g. membership in a blacklist) fails with an error, then also its negation fails with an error.

lookup

The general setup, as far as we are concerned here, is that trust lists are stored relative to a domain, such as `qualified.trust.admin.eu` for the trustlist of qualified eIDAS signers. We do not want to have to download the entire trust list to check if somebody is a member, thus we assume every entry to have a particular search-key in form of a subdomain, for instance

`PX2NO4LVPA4WHCBLYXHIKRWVRE. qualified . trust .admin.eu`

We assume that we can lookup such a URL to get a trust list entry using a predicate

`lookup (URL, TrustListEntry )`

Note that this predicate triggers a server lookup, i.e., at the moment the policy checker (the TPL interpreter) reaches this lookup predicate, it interacts with the server. The predicate will succeed and return the respective `TrustListEntry` if this entry indeed exists, and otherwise the predicate will fail. It can thus act as a requirement in a policy.

Since a lookup query to a server is a time consuming task (as compared to the other checks that are made locally), there is the risk that an inexperienced user specifies policies that get stuck in checking many queries repeatedly. For this reason, we may integrate a caching mechanism into the lookup predicate: all queries done over a certain period (say 5 minutes) are stored with their results, so that repeated queries within the time frame are answered from the cached result.

A certificate can now claim a trust list membership, for instance qualified signatures. For that, it shall include a field `trustList` that contains the URL for looking up the membership e.g. `PX2NO4LVPA4WHCBLYXHIKRWVRE.qualified.trust.admin.eu`. For existing certificate formats like X.509, we may use a field like "issuer alternate name" to play the role of the `trustList` field; this is then a matter the parser interfacing concrete and abstract syntax.

trustscheme

As also described in D2.5 in more detail, there is a pitfall of just checking the trustlist entry given in a certificate without checking that it refers to the correct URL of the desired trust scheme. Therefore, we also use the predicate trustscheme to check that the membership claim is for the intended trustlist, for instance

```
trustscheme(URL,eIDAS_qualified)
```

should yield true on the example URL

`PX2NO4LVPA4WHCBLYXHIKRWVRE.qualified.trust.admin.eu`

while it would not on e.g. `PX2NO4LVPA4WHCBLYXHIKRWVRE.intruderspace.tk` .

The combined use of the above predicates would typically look as follows on a given Certificate:

```
extract(Certificate,trustList, URL),

trustscheme(URL,eIDAS_qualified),

lookup(URL,TrustListEntry)
```

followed potentially by some checks on the `TrustListEntry` if it is not a boolean trustscheme.

verify_signature

To check signatures: it has two arguments, namely a document and a public key and verifies that the document is indeed signed with the corresponding private key. We assume for this purpose

that a format that is meant to bear a signature has a dedicated field for the signature, and the concrete syntax can extract the part that is being signed (normally all remaining fields, but this is up to the designer of the respective format). Thus, we can indeed abstractly write in TPL the following predicate:

```
verify_signature(Document, PublicKey)
```

and rely that this predicate triggers the extraction of the data being signed and the signature and verify it against the given public key.

6.3 Data Formats for Trust Policy

We have already described a general interface for accessing different kinds of data formats. In fact, all that is required to support a given format is a parser for the given format (and a pretty printer) that is accessible via the extracted format. This one has defined a unique format name, and a name for each (top-level) field of the format for instance "issuer" and "bearer".

In some cases, a format may be more complicated, for instance, contain lists or even a hierarchy of sub-documents. In these cases, we can define formats for each type of sub-document, or use data constructors like a list.

A very desirable property is if formats are pairwise disjoint, i.e., no string can be parsed as two different formats. For instance, XML-based formats satisfy this by construction if the top-level XML-tags of the formats are different; this is for instance easily ensured by using two disjoint namespaces. We do not make it a general requirement since sometimes there will be overlaps in practice. But then these are also cases where we have to spend extra care that a document is not accidentally understood in a completely different way than it was meant by the receiver because an attacker (or an honest mistake) has to lead to the document being used in a different context. This is excluded whenever we have the disjointness property, however.

6.4 Existing Trust Policy Authoring Tools

This section gives a short market overview of existing Trust Policy Authoring Tools. A list of existing tools is in Table 1 in alphabetic order. From this list, only two instruments have a rigorous language backing like the TPL in LIGHTest. These are LaSCO and Poliseer, which are shortly introduced in the following. The information is from the corresponding websites listed in Table 1.

Table 1: General Overview of Existing Policy Authoring Tools

Tool name	Link
Allot Communications NetPolicy	https://www.allot.com/products/traffic-management-and-optimization/netpolicy-provisioner/
Infrastructure Management and eTrust Solutions	https://www.avepoint.com/solutions/infrastructure-management/
irondetect	https://github.com/trustathsh/irondetect

LaSCO	[24]http://seclab.cs.ucdavis.edu/projects/arpa/LaSCO/welcome.html
Microsoft Trusted Root Certificate	https://technet.microsoft.com/en-us/library/cc751157.aspx
NetAPT	https://www.perform.illinois.edu/netapt/
Nortel Optivity	http://www.cascadetel.com/productcatalog/nortel/enterprise_network_management_portfolio.aspx
OneTrust	https://onetrust.com/
OpenDOAR	http://www.opendoar.org/
PFIRES	http://www.cerias.purdue.edu/apps/reports_and_papers/view/2805
PoliSeer	[25]http://www.cse.usf.edu/~ligatti/papers/psr-tr.pdf
SASI (Security Automata SFI Implementation)	http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.147.556&rep=rep1&type=pdf
Tivoli Management Framework	https://www.ibm.com/support/knowledgecenter/en/SSTFWG_4.3.1/com.ibm.tivoli.itcm.doc/icm42mst19.htm
TrustArc-the new TRUSTe-TrustArc Data Privacy Management Platform	https://www.trustarc.com/products/privacy-platform/
VisITMeta	https://github.com/trustathsh/visitmeta

LaSCO is the abbreviation for “Language for Security Constraints on Objects”. LaSCO was developed and published in a dissertation by James A, Hoagland at UC Davis in 2000 as a new approach to security policy specification and enforcement [24]. With LasCO, users can write their constraint policies for many application and systems and the policy description, the constraints are stated as directed graphs. Technically, the LaSCO policy is separated into two components: the domain and the requirement, which specify under which situation the policy is applied and what the corresponding requirement is. The policy statements are clear and unambiguous, and a formal operational semantics has been defined. To enable a broad and diverse range of applications, LaSCO has been implemented for Java programs.

PoliSeer is a tool for Managing Complex Security Policies [25]. Lomask and Ligatti from the University of Florida developed PoliSeer in 2009. The idea behind PoliSeer is to decompose complex security policies into simpler modules. Therefore, PoliSeer enables users the specification, visualisation, modification and enforcement of complex policies into a set of arbitrary compositions of simpler sub policies by modularisation. PoliSeer uses Polymer, which is a tool allowing the specification of cross-domain policies however without visualisation capabilities, as

the basis. The authors see PoliSeer as a simple IDE for security policies, similar to the traditional IDEs for software engineers as visualisation and analysing tool.

The comparison of this overview of existing Trust Policy Authoring Tools with the requirements within LIGHTest indicates that the use of LaSCO or PoliSeer does not fulfil all requirements as Trust Policy Authoring Tool. However, they provide helpful and useful ideas and modules which support the development of an own Trust Policy Authoring Tool in Lightest.

6.5 Proposed LIGHTest Trust Policy Authoring Tool

The Trust Policy Authoring tool will be used to create and edit trust policies based on the TPL. The tool should be able to cater to a broad spectrum of users, i.e., both technical users and non-technical users. It should be user-friendly to make the task of creating trust policies in TPL as hassle-free as possible.

The tool will be created with the capability of writing policies directly in TPL which experts in the TPL language will find more beneficial or specify trust policies in either natural language or using a graphical language. It should be noted that the natural language and graphical language would be converted to TPL as soon as the need arises. The tool does this automatically. A high-level structure of the tool is shown below

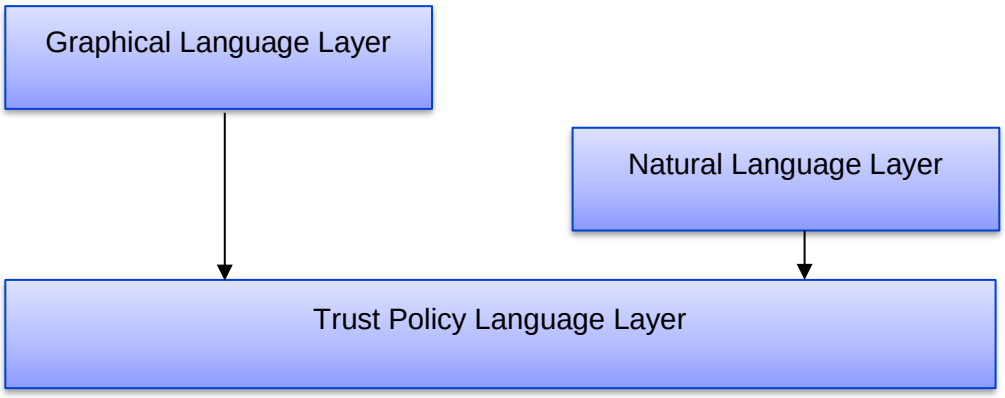

Figure 4 Structure of the Trust Policy Authoring Tool

The lowest layer is the Trust Policy Language (TPL) layer which will be used by technical users. This layer provides the most power and freedom in specifying a trust policy, but it has a steep learning curve. The next layer above the TPL layer is the Natural Language Layer. This layer is less complicated compared to the lowest layer but still complicated and might have a steep learning curve. It offers less power and freedom compared to the TPL layer but more than the next layer which is the graphical language layer. The graphical language layer is the next and highest layer of abstraction in the tool. The graphical layer is for the least technical user, and it does not require a lot of training to start using it to write basic trust policies for electronic transactions. However, out of the three layers, this layer provides the least freedom and flexibility. Most of the policies can be designed in this layer, but complex policies have to be written at a lower layer.

To create the right tool, some high – level functional requirements were derived. The functional requirements for the tool are:

1. It must be able to generate trust policies
2. It must be able to view trust policies
3. It must be able to edit trust policies
4. It must compile the graphical language into TPL
5. It must be able to save the graphical language using a distinct file format
6. It must compile the natural language into TPL
7. It must be able to save the natural language into using a distinct file format
8. It must save the compiled TPL in a .tpl format

This tool will be implemented in Task 6.3: Open Source Tool for Visualization and Editing of Trust Policies.

6.6 Trust Policy Templates

6.6.1 Business

Following, there are listed several business use cases that have been modelled with TPL

6.6.1.1 Use Case (Purchasing Order 1):

Company A will process a purchase order of over 1 million euro from Company B only if it is electronically signed by the CEO with an eIDAS qualified signature Trust Policy.

> if <amount> of <purchase order> is greater than **1 million euro**, and <signature> is <eIDAS_qualified_signature>, and <signature> is of <Company_B_CEO>, then process <purchase order>

6.6.1.2 Use Case (Purchasing Order 2):

Company A will process a purchase order of less than 1 million euro from Company B only if it is electronically signed by the CEO or their delegate with a qualified signature belonging to eIDAS or translatable to it Trust Policy.

> if <amount> of <purchase order> is less than **1 million euro**, and <signature> is <eIDAS_qualified_signature> or <signature> is translatable to <eIDAS_qualified_signature>, and <signature> is of <Company_B_CEO> or <signature> is of a delegate of <Company_B_CEO>, then process <purchase order>

```
valid_purchaseorder(PurchaseOrder) :-
      % UBL standard is included in the company trust policy.
      extract(PurchaseOrder, format, ubl_format),

      % eIDAS signature: qualified. It could be also the eIDAS seal, why not!

      extract(PurchaseOrder,certificate,Cert),
      is_EIDAS_qualified_or_equiv(Cert),
```

```
      % Searching in the related sub-domain
      extract(Certificate, trustList, URL),
      trustscheme(URL, eIDAS_qualified),
      lookup(URL, TrustListEntry)
      extract(TrustListEntry, format, business_entry),

      extract(PurchaseOrder, issuer, Company),
      extract(TrustListEntry, entity, Company),
      extract(PurchaseOrder, currency, Currency),
      extract(TrustListEntry, currency, Currency),
      extract(PurchaseOrder, amount, Amount),
      extract(TrustListEntry, upperThreshold, UpperThreshold),

      correctLevel(TrustListEntry, PK, UpperThreshold),
      verify_signature(PurchaseOrder, PK).

correctLevel(TrustListEntry,PK,UpperThreshold) :-
      % Signature of a delegee is sufficient only if Amount <= UpperThreshold
      Amount <= UpperThreshold,
      extract(TrustListEntry, deleg_pub_key, PK)).

correctLevel(TrustListEntry,PK,UpperThreshold) :-
      % Signature of the CEO is always sufficient
      extract(TrustListEntry, ceo_pub_key, PK)).

lookup (SD,TrustList,TrustListEntry) :-
      encode (SD,TrustList,URL),
      query(URL,TrustListEntry).

is_EIDAS_qualified_or_equiv (Cert) :-
      extract(Cert,trustscheme,TrustScheme),
      eIDAS_equal_or_equiv(TrustScheme),
      extract(Cert,altIssuerName,TrustMemClaim),
      lookup(TrustMemClaim,TrustScheme).

eIDAS_equal_or_equiv(["TrustScheme","signature","trust","europa","eu"]).
eIDAS_equal_or_equiv(TS) :-
      translation(TS,["TrustScheme","signature","trust","europa","eu"]).

translation(RTS,TTS) :-  % translation from recognized trust scheme(RTS) to
trusted trust scheme (TTS)
      encode(RTS,TTS,URL),  % lookup trust scheme URL for this
      query(URL,TrustListEntry). % Checking this is contained

% Other predicates to be defined - out of scope of this use case
% translate_signature (TargetScheme, SourceScheme)
% translate_seal
% verify_signature (Document, PublicKey)
% verify_seal
```

6.6.1.3 Use Case (Correos MyMailbox):

Correos will accept a "send document" order from Company A only if they are a Correos certified sender, the document is sealed with the Company A qualified seal, and the document is sent to not more than the number of users specified by Company A contract Trust Policy.

> if <Company A> belongs to <Correos certified sender list> and <seal> is <eIDAS_qualified_seal> and <seal> is of <Company_A> and <user list> length is less or equal than <maximum users> for <Company_A> then send <document> to <user list>

```
valid_document (DocToSendByCorreos) :-
      extract(DocToSendByCorreos, format, correos_doc_format),
      % The document is e-sealed.

      extract(DocToSendByCorreos, seal, ESeal),
      is_EIDAS_qualified_or_equiv(ESeal),

      % Searching in the related sub-domain
      extract(DocToSendByCorreos, trustList, URL),
      trustscheme(URL, eIDAS_qualified),
      lookup(URL, TrustListEntry)
      extract(TrustListEntry, format, correos_company_entry),

      % Verifying the Company ordering the massive document mailing
      extract(DocToSendByCorreos, issuer, CorreosCompany),
      extract(TrustListEntry, entity, CorreosCompany),
      extract(TrustListEntry, seal_key, SealK),
      verify_seal(DocToSendByCorreos, SealPK),

      % Checking the number of sendings ordered by the company
      extract(DocToSendByCorreos, numTimes, NumTimesOrdered),
      extract(TrustListEntry, numTimes, NumTimesContracted),
      NumTimesOrdered <= NumTimesContracted.

lookup (SD,TrustList,TrustListEntry) :-
      encode (SD,TrustList,URL),
      query(URL,TrustListEntry).

is_EIDAS_qualified_or_equiv (ESeal) :-
      extract(ESeal,trustscheme,TrustScheme),
      eIDAS_equal_or_equiv(TrustScheme),
      extract(ESeal,altIssuerName,TrustMemClaim),
      lookup(TrustMemClaim,TrustScheme).

eIDAS_equal_or_equiv(["TrustScheme","seal","trust","europa","eu"]).
eIDAS_equal_or_equiv(TS) :-
      translation(TS,["TrustScheme","seal","trust","europa","eu"]).
translation(RTS,TTS) :-
      encode(RTS,TTS,URL),
      query(URL,TrustListEntry).
```

6.6.1.4 Use Case (Correos MyVerifiedCommunications):

A user wants to verify that a gas bill they received from the gas company is genuine and that it was delivered less than one day after being issued, all at the level of eIDAS qualified trust services Trust Policy.

> if <bill> is addressed to <user name> and <seal> is <eIDAS_qualified_seal> and <seal> is of <gas company> and <timestamp> is eIDAS_qualified_timestamp> and <timestamp> less than **1 day** sooner than <today> then send OK message to <user>

```
valid_bill (GasBill) :-
      % Checking the structure of the bill.
      extract(GasBill, format, bill_format), % The bill is addressed to a
user...

      % The bill is e-sealed.
      extract(GasBill, seal, ESeal),
      is_EIDAS_qualified_or_equiv(ESeal),

      % Searching in the related sub-domain
      extract(GasBill, trustList, URL),
      trustscheme(URL, eIDAS_qualified),
      lookup(URL, TrustListEntry)
      extract(TrustListEntry, format, gas_business_entry),

      extract(GasBill, issuer, GasCompany),
      extract(TrustListEntry, entity, GasCompany),
      extract(GasBill, country, Country),
      extract(TrustListEntry, country, Country),

      extract(TrustListEntry, seal_key, SealK),
      verify_seal(GasBill, SealPK),

      % Checking the timestamp of the seal.
      extract (ESeal, timestamp, Timestamp),
      is_EIDAS_qualified_or_equiv(Timestamp),

      extract (Timestamp, datetime, DateTime),
      DateTime + one_day >= today().
      % Assuming we have a function "today()" that returns today's date and
time,
      % and "one_day" is an appropriate constant in this data format.
is_EIDAS_qualified_or_equiv (ESeal) :-
      extract(ESeal,trustscheme,TrustScheme),
      eIDAS_equal_or_equiv(TrustScheme),
      extract(ESeal,altIssuerName,TrustMemClaim),
      lookup(TrustMemClaim,TrustScheme).

eIDAS_equal_or_equiv(["TrustScheme","seal","trust","europa","eu"]).
```

```
eIDAS_equal_or_equiv(TS) :-
      translation(TS,["TrustScheme","seal","trust","europa","eu"]).
translation(RTS,TTS) :-
      encode(RTS,TTS,URL),
      query(URL,TrustListEntry).
```

6.6.1.5 Use Case (**Credit Card**)

A credit card company verifies a point of sale purchase: if the purchase is for more than 500 euro, then an eID of the person is required with at least eIDAS Substantial level Trust Policy.

> if <user name> matches <cardholder list> for <credit card number> and <expiration date> is greater than <today> and (<amount> is less than **500 euro**) or (<amount> is equal or more than **500 euro** and <eID level> is equal or greater than <eIDAS Substantial>), then authorise purchase

```
valid_creditcard(Creditpurchase) :-
      % UBL standard is included in the bank trust policy.
      extract(Creditpurchase,format,ubl_format),

      % Searching in the related sub-domain
      extract(Creditpurchase, trustList, URL),
      trustscheme(URL, eIDAS_qualified),
      lookup(URL, TrustListEntry)

      extract(TrustListEntry, format, creditcard_company_entry),

      %Credit card is not expired
      extract(Creditpurchase,datetime,Expirationdate),
      (Expirationdate>Today),

      extract(Creditpurchase, amount, Amount),
      extract(TrustListEntry, upperThreshold, UpperThreshold).

correctThreshold (Creditpurchase, PersoneID,UpperThreshold) :-
      Amount => UpperThreshold,

      extract(Creditpurchase, eid, PersoneID),
      is_EIDAS_substantial_or_equiv(PersoneID).

is_EIDAS_substantial_or_equiv (PersoneID) :-
      extract(ESeal,trustscheme,TrustScheme),
      eIDAS_equal_or_equiv(TrustScheme),
      extract(PersoneID,altIssuerName,TrustMemClaim),
      lookup(TrustMemClaim,TrustScheme).

eIDAS_equal_or_equiv(["TrustScheme","eid","trust","europa","eu"]).
eIDAS_equal_or_equiv(TS) :-
      translation(TS,["TrustScheme","eid","trust","europa","eu"]).
```

```
translation(RTS,TTS) :-
      encode(RTS,TTS,URL),
      query(URL,TrustListEntry).
```

6.6.1.6 Use Case (Trusted Billing)

Company A receives an invoice from Company B with bank details changed with respect to the usual procedure and wants to check if the document is genuine and the new bank is already known and trusted by their Trust Policy

if <bank> belongs to <trusted banks list> and <seal> is eIDAS_qualified_seal> and <seal> is of <Company B> then process <invoice>

```
valid_invoicebank(Invoice) :-
      % UBL standard is included in the bank trust policy.
      extract(Invoice,format,ubl_format),

      extract(Invoice, trustList, URL),
      trustscheme(URL, eIDAS_qualified),
      lookup(URL, TrustListEntry)

      extract(TrustListEntry, format, bank_entry),

      extract(Invoice,name,Bankname),
      extract(TrustListEntry,name,Bankname),

      extract(Invoice, seal, CompanyB_seal),
      is_EIDAS_qualified_or_equiv(CompanyB_seal),

      extract(TrustListEntry,pub_seal,Seal),
      verify_seal(Invoice,Seal).

is_EIDAS_qualified_or_equiv (CompanyB_seal) :-
      extract(CompanyB_seal,trustscheme,TrustScheme),
      eIDAS_equal_or_equiv(TrustScheme),
      extract(ESeal,altIssuerName,TrustMemClaim),
      lookup(TrustMemClaim,TrustScheme).

eIDAS_equal_or_equiv(["TrustScheme","seal","trust","europa","eu"]).
eIDAS_equal_or_equiv(TS) :-
      translation(TS,["TrustScheme","seal","trust","europa","eu"]).
translation(RTS,TTS) :-
      encode(RTS,TTS,URL),
      query(URL,TrustListEntry).
```

6.6.1.7 Use Case (**Notary**):

Government A wants to prove the validity of a certificate provided by a user from State B Trust Policy.

if <certificate issuer> is <Government B> and <seal> is translatable to <eIDAS_qualified_seal> and <seal> belongs to <Government B> and <signature> is translatable to <eIDAS_qualified_signature> and <signature> is of a delegate of <Government B> for <public notary services>, then send OK message

```
valid_notarydocument(Document) :-
      extract(Document,format,notary_format),

      extract(Document, trustList, URL),
      trustscheme(URL, eIDAS_qualified),
      lookup(URL, TrustListEntry)
      extract(TrustListEntry, format, government_entry),

      extract(Document, seal, ESeal),
      is_EIDAS_qualified_or_equiv(ESeal),

      extract(Document, signature, ESignature),
      is_EIDAS_qualified_or_equiv(ESignature),

      extract(Document,issuer,Statename),
      extract(TrustListEntry,entity,Statename),

      extract(TrustListEntry,pub_key,PK),
      verify_signature(Document,PK).

is_EIDAS_qualified_or_equiv (ESeal) :-
      extract(ESeal,trustscheme,TrustScheme),
      eIDAS_equal_or_equiv(TrustScheme),
      extract(ESeal,altIssuerName,TrustMemClaim),
      lookup(TrustMemClaim,TrustScheme).

eIDAS_equal_or_equiv(["TrustScheme","seal","trust","europa","eu"]).
eIDAS_equal_or_equiv(TS) :-
      translation(TS,["TrustScheme","seal","trust","europa","eu"]).
translation(RTS,TTS) :-
      encode(RTS,TTS,URL),
      query(URL,TrustListEntry).
```

6.6.1.8 Use Case (**Cross Borders Banking**):

A national of State A travels to State B for study and wants to open a bank account using an ID from Government B Trust Policy.

if <eID issuer> is <Government B> and <eID level> is translatable to equal or greater than <eIDAS Substantial> and <State B> is not in <Government A rogue state list> and <user name> is not in <Government A money laundering watch list>, then open account to <user name>

```
valid_newAccount (UserID) :-
```

```
        % Checking the level of trust of the eID. At least, Substantial.
        is_EIDAS_substantial_or_equiv(UserID),

        extract (UserID, state, StateB),
        extract (UserID, userName, UserName),

        % Searching in the related sub-domain.
        extract(StateA, trustList, URL),
        trustscheme(URL, eIDAS_qualified),
        lookup(URL, TrustListEntry)
        extract(TrustListEntry, format, state_business_entry),

        % Checking the StateA requirements for opening an account.
        extract(TrustListEntry, rogue_state_list, Roguelist),
        not(blacklisted(StateB, Roguelist)),
        extract(TrustListEntry, money_laundering_watch_list, Laundrylist),
        not(blacklisted(UserName, Laundrylist)).

blacklisted(Entity, List) :-
        lookup(Entity,List).

is_EIDAS_substantial_or_equiv (UserID) :-
        extract(UserID,trustscheme,TrustScheme),
        eIDAS_equal_or_equiv(TrustScheme),
        extract(UserID,altIssuerName,TrustMemClaim),
        lookup(TrustMemClaim,TrustScheme).

eIDAS_equal_or_equiv(["TrustScheme","eID","trust","europa","eu"]).
eIDAS_equal_or_equiv(TS) :-
        translation(TS,["TrustScheme","eID","trust","europa","eu"]).
translation(RTS,TTS) :-
        encode(RTS,TTS,URL),
        query(URL,TrustListEntry).
```

6.6.2 Health

Below, there is a listed health use case that has been modelled with TPL

6.6.2.1 Use Case (Cross-border Health services)

Doctor Ackermann from Country A wants to **verify** that Patient Bonnie Clyde from Country B has valid insurance from Country B. Patient Bonnie Clyde **provides** Doctor Ackermann with an EU approved Health Card. Doctor Ackermann **verifies** that if the Health Insurance from Patient Bonnie Clyde is on the list of Approved list of Health Insurances from Country B. Doctor Ackermann **verifies** if Patient Bonnie Clyde is a patient from the fore stated Health Insurance by electronic Signature.

Natural Language Representation:

if patient <C_Surname> <C_GivenName> born on <C_Date of birth> in <country> has a valid <signature> from <health insurance> in <country> and <health insurance> is listed on <certified health insurances> in <country>, then patient <C_Surname> <C_GivenName> born on <C_Date of birth> in <country> has valid health insurance

TPL Representation:

```
valid_healthinsurance(Certificate) :-
    extract(Certificate,format,healthcare_cert_format),

    extract(Certificate, trustList, URL),
    trustscheme(URL, eIDAS_qualified),
    lookup(URL, TrustListEntry)

    extract(TrustListEntry, format, healthcare_entry),

    extract(Certificate, issuer, InsuranceOrg),
    extract(TrustListEntry, entity, InsuranceOrg),
    extract(Certificate, country, Country),
    extract(TrustListEntry, country, Country),

    extract(TrustListEntry, pub_key, PK),
    verify_signature(Certificate, PK).

lookup (SD,TrustList,TrustListEntry) :-
    encode (SD,TrustList,URL),
    query(URL,TrustListEntry)
```

6.6.3 Academics

Below, there is a listed academic use case that has been modelled with TPL

6.6.3.1 Use Case

The University of Yale from the USA needs to verify if Student Abigail Schmidt from Germany has a valid Diploma from University of Berlin from Germany. Student Abigail Schmidt provides University of Yale with a Diploma certificate document. University Yale from the USA verifies that the University of Berlin is on the trusted list of Accredited Universities published by Germany. Second, University Yale verifies that Student Abigail Schmidt is on the published list of graduates from the University of Berlin from Germany.

Natural Language Representation:

if <U_NameofUniversity> is listed on <trusted list of accredited universities> published by <U_Residence> and student <AlumniID> is listed on <published list of graduates> from university <U_NameofUniversity> in <U_Residence>, then student <AlumniID> has a valid diploma

TPL Representation:

```
valid_university(Certificate) :-
    extract(Certificate,format,university_cert_format),
    extract(Certificate, trustList, URL),
    trustscheme(URL, eIDAS_qualified),
    lookup(URL, TrustListEntry)

    extract(TrustListEntry, format, university_entry),

    extract(Certificate, issuer, University),
    extract(TrustListEntry, entity, University),
    extract(Certificate, country, Country),
    extract(TrustListEntry, country, Country),
    extract(TrustListEntry, pub_key, PK),
    verify_signature(Certificate, PK).

valid_universitydiploma(Certificate) :-
    extract(Certificate,format,universitydiploma_cert_format),
    extract(Certificate, trustList, URL),
    trustscheme(URL, eIDAS_qualified),
    lookup(URL, TrustListEntry)
    extract(TrustListEntry, format, universitydiploma_entry),
    extract(Certificate, alumniID, Student),
    extract(TrustListEntry, entity, Student),
    extract(Certificate, country, Country),
    extract(TrustListEntry, country, Country),
    extract(TrustListEntry, pub_key, PK),
    verify_signature(Certificate, PK).

lookup (SD,TrustList,TrustListEntry) :-
    encode (SD,TrustList,URL),
    query(URL,TrustListEntry)
```

7. Automatic Trust Verifier

7.1 Introduction to Automatic Trust Verifier

Automatic Trust Verifier (see Figure 5 Automatic Trust Verifier) takes an Electronic Transaction and Trust Policy as input, outputs trustworthy [y/n], possibly with an explanation of its reasoning (in particular if not trustworthy). It uses a pluggable parser for Electronic Transactions as sub-component.

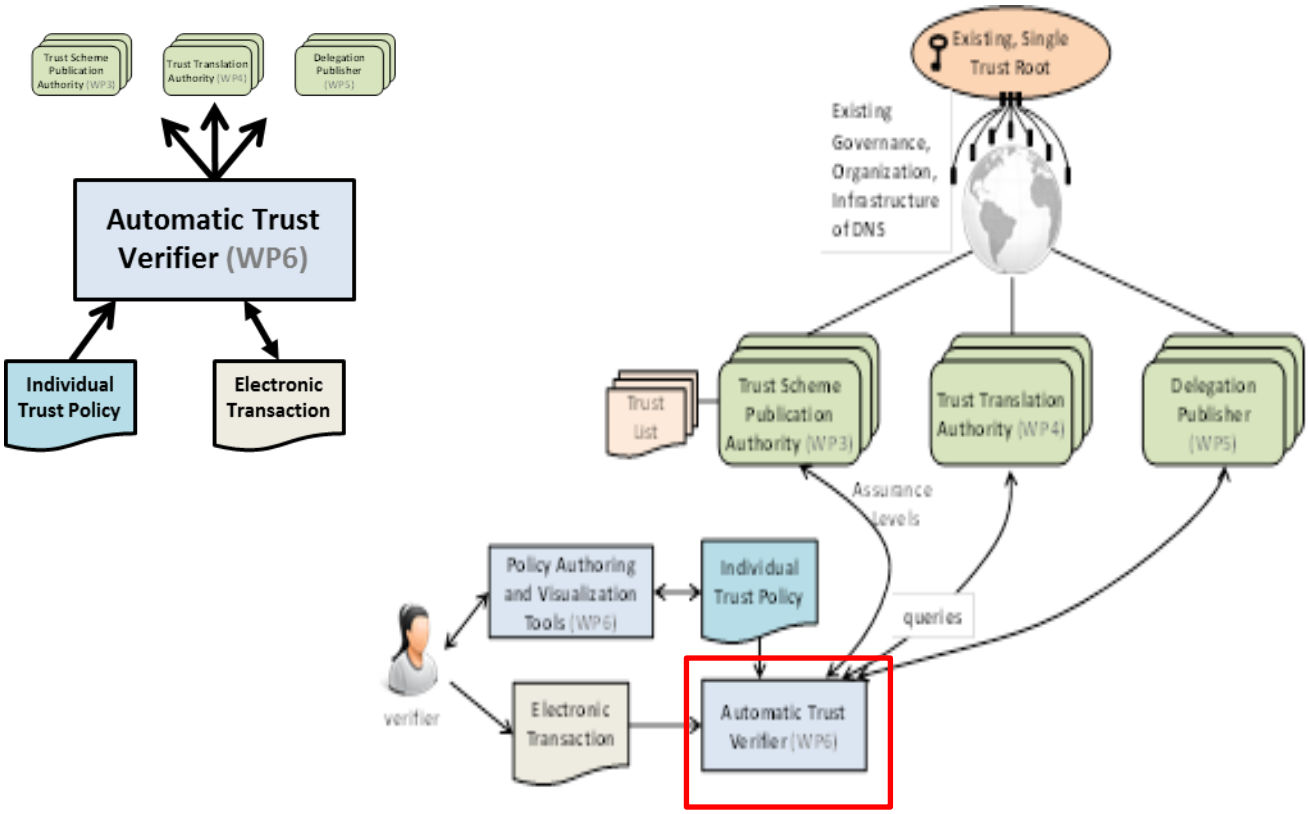

Figure 5 Automatic Trust Verifier

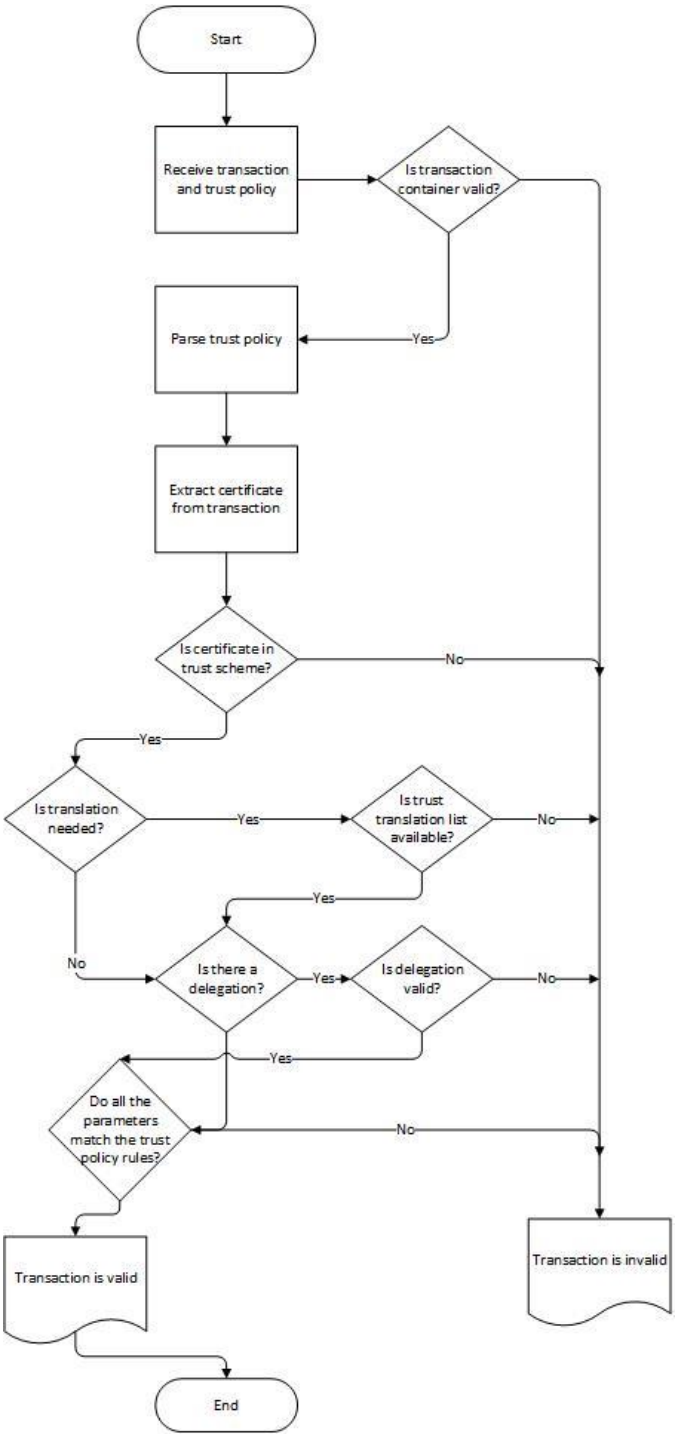

Figure 6 ATV process flow

7.2 Related Components

7.2.1 Trust Scheme Publication Authority

The task of the Trust Scheme Publication Authority (TSPA) is the discovery and verification of trust scheme memberships. For this purpose, the TSPA consist of two components. It uses

I. a standard DNS Name Server with DNSSEC extension, and
II. a server, which publishes multiple trusts lists under different sub-domains of the authority's domain name.

The DNS Name Server is used to discover the Trust Scheme Provider that operates a Trust scheme. The Trust Scheme Provider then provides the signed Trust List, which indicates that a certificate Issuer is trusted under the scheme operated by the Trust Scheme Provider. Also, it gives the Tuple-Based representation of the Trust Scheme. D3.1 [22] provides more detailed information on the concept for Trust Scheme Publication as well as the data model of Tuple-Based Trust Scheme Publication Authorities.

The interaction between the ATV and the TSPA is illustrated in Figure 7: Sequence Diagram for Trust Publication of a Qualified Signature (Boolean) [2], which is taken from D2.14 [2] and also described in D3.1 [22]. The interaction occurs at

Step 6 and Step 7.

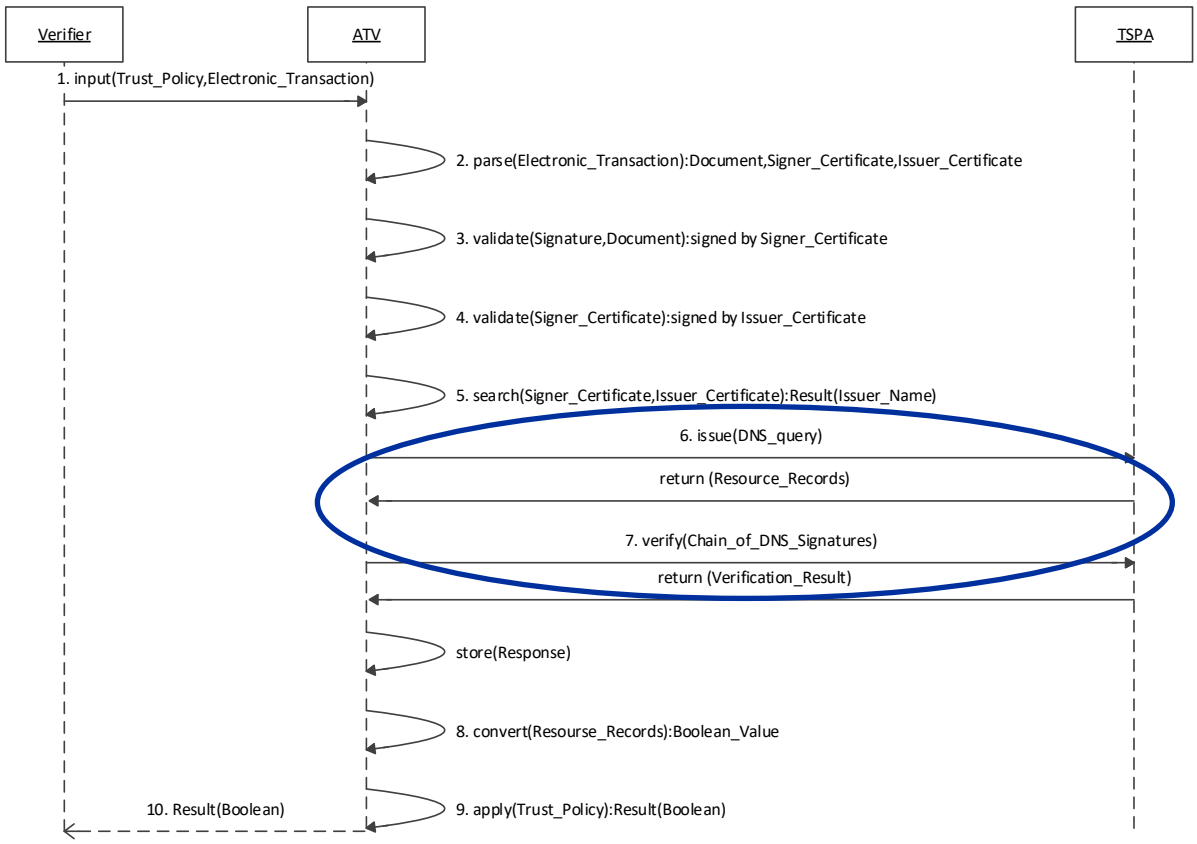

Figure 7: Sequence Diagram for Trust Publication of a Qualified Signature (Boolean) [2]

At Step 6, the ATV has already extracted the signer certificate (Step 2) and the Issuer Name (Step 5). Hence, at Step 6, the ATV knows the Issuer Name but not the associated Trust Scheme. Therefore, the discovery of the Trust Scheme and the operating Trust Scheme Provider is required, which is described in the following in more detail:

For the process of querying trust schemes, the ATV contacts the TSPA as follows for Step 6 (see also D3.4 [26]):

I. The ATV issues a query to the DNS Name Server with the Issuer Name.
II. The DNS Name Server delivers the record for the Issuer Name that contains the name of the associated trust scheme.
III. The ATV queries the DNS Name Server for the associated trust scheme.
IV. The DNS Name Server provides the record for the Trust Scheme Provider, which contains the Location of the Trust Scheme Provider.
V. The ATV queries the Trust Scheme Provider (HTTP component) and provides the Issuer Name to the Trust Scheme Provider.

VI. The Trust Scheme Provider queries its Trust List and verifies whether the Issuer Name is listed as operating under its Trust Scheme. The Trust Scheme Provider finally provides a signed statement of association of an Issuer Name with a Trust Scheme to the ATV.
VII. The ATV queries the DNS Name Server for the associated trust scheme again.
VIII. The DNS Name Server provides the SMIMEA record for the Scheme Name that contains constraints, which limit the accepted certificates for the signature of the trusted list.
IX. The ATV can now verify if the certificate used for signing the trusted list was valid.

As a result of Step 6, the Trust Scheme Provider provides in the positive case of a signed statement of association of an Issuer Name with a Trust Scheme. The ATV can then verify that the statement of association of an Issuer with a Trust Scheme was made by the Trust Scheme Provider. The ATV further verifies that the certificate used for signing the trust list is valid making use of the SMIMEA record data.

In Step 7, the ATV contacts the TSPA to verify the chain of signatures from the DNS trust root of the DNS response using a validating resolver and stores the response as a “receipt” for future justifications of its decision.

Each DNS resource record set that is used by the ATV needs to be DNSSEC validated. This provides a formal proof that the record set has indeed been published by an authorised party. Validation is based on digital signatures associated with each resource record set. The public keys necessary for verifying the signatures are published in each zone. The resource record set for publishing the keys is signed as well. A fingerprint of the key used for this latter signature is published in the parent zone as part of the delegation data. It is signed by the parent zone, which in turn publishes a key fingerprint in its parent zone, up to the root zone. The acceptable keys for the root zone have to be configured and form the trust anchor for validation.

Section 6 of deliverable D2.7 [27] provides a more detailed overview of DNSSEC validation. The actual algorithm is described in section 5 of RFC 4035 [28]. It turned out to be so complicated and full of subtleties that RFC 6840 [29] was published to provide ‘Clarifications and Implementation Notes for DNS Security.’ As a consequence, DNSSEC validation shouldn’t be implemented from scratch as part of the ATV but should be left to a proven library.

Since DNS provides no facility to query for past data, the ATV needs to keep all the DNS data used during validation if it is deemed necessary to be able to repeat the validation process at a later time. Since all validation data except the root zone trust anchor is gathered using DNS, the ATV can store all the DNS responses to the queries made during validation.

Each of these responses will contain the answer to the query, either positive or negative, as well as all DNSSEC necessary the start verifying the response. For a positive answer, this is the signature for the resource record set. For negative answers, this is one or more records that can be used to prove that there is no answer formally.

As part of validation, the ATV will see and therefore have to store the following responses:

- the response to the originally requested query,
    - if it is a positive response, the resource record set containing the records answering the query, and one or more RRSIG resource record providing a signature, key identifier for verification of the signature, and validity of the signature,
    - if it is a negative response, some NSEC or NSEC3 records that can be used to prove the non-existence as well as one or more RRSIG records for each of those records;
- the DNSKEY resource record set for the zone the originally requested domain name is part of as well as one or more RRSIG records for this record set;
- the DS resource record set for the delegation to that zone as well as its RRSIG records; these records are part of the parent zone and provide the link to it;
- the DNSKEY resource records sets for each parent zone all the way up to the root zone, each including their RRSIG records;
- the DS resource records sets for each parent zone up to the top level domain under the root zone, each, again, including its RRSIG records.

Apart from the response to the original query, each response must be a positive response containing one or more resource records.

The description of the interaction between the TSPA and the ATV indicates that currently, the TSPA provides four operations to the ATV:

I. getTrustSchemeID (in issuerNameID, out trustSchemeID): Using the Issuer Name, which contains a claim that the issuer operates under a certain trust scheme, the TSPA provides the name of the associated trust scheme.
II. getTrustSchemeProviderID (in trustSchemeID, out trustSchemeProviderID): Using the associated trust scheme, the TSPA provides the record for the Trust Scheme Provider, which contains the location of the Trust Scheme Provider.
III. verifyTrustSchemeMembership (in trustSchemeProviderID, in issuerNameID, out trustSchemeMembership): Using the Trust Scheme Provider and Issuer Name, the TSPA queries its Trust List and verifies whether the Issuer Name is listed under its Trust Scheme.

IV. getSMIMEArecord (in trustSchemeID, out certificateConstraints)
V. verifySignature (in signatureTrustList, in certificateConstraints, out signatureVerificationResult)

VI. verifyChainOfSignatures (in signatureChain, out result): implemented by the DNSSEC software used.

For the communication between the ATV and the TSPA the security measures of TLS are sufficient.

In addition to the discovery and verification of trust scheme memberships, the TSPA component will provide further interfaces for uploading, modifying and removing trust schemes, but such services are not to be used by the operator of the ATV component, but by the LIGHTest administrator or some similar role.

7.2.2 Trust Translation Authority

As it has been described in D4.4 [30], Trust Translation Authority provide translations between Trust Schemes at the height of Service-Level. This definition fits with Ordinal Trust Schemes, which provides a set of named levels for each service, but makes necessary to look for a way to manage Boolean and tuple based (attribute/value) trust schemes.

Boolean trust schemes will be considered as having one only level which corresponds to the name of the service itself. E.g. timestamp service of eIDAS.

In the case of tuple-based Trust Schemes we should consider that a relation of translation is set for a specific set of tuples, this is, for a set of attributes with a fixed value. The next example comes from D4.4 section 7.3.2.

Considering the FIDO trust scheme. Two Authorities sign an agreement setting that eIDAS scheme, signature service, QUALIFIED can be translated into FIDO when the following set of FIDO tuples match:

- authenticatorVersion 2.
- Limiting attempts for 30 seconds after five false trials.
- The (fingerprint) matcher is implemented in TEE.

Where TSPA authority of LIGHTest provides the name *fido2fingerprint* as the service-level name for this set of tuples.

This means that there is a translation relationship where eIDAS trust scheme is the source of trust and FIDO is the target. This translation relationship is named *fido2fingerprint.*

When ATV has to check a transaction where FIDO scheme is used with the following tuples:

- authenticatorVersion 2.
- Limiting attempts for 30 seconds after five false trials.
- The (fingerprint) matcher is implemented in TEE.

It can happen that the policy for such transaction does not consider FIDO as a valid scheme, then ATV can query LIGHTest to look for alternatives. More specific, ATV needs to find a relation of trust where:

- FIDO, with that attribute set, is the target of a trust relation,
- It has a source of trust valid for the policy.

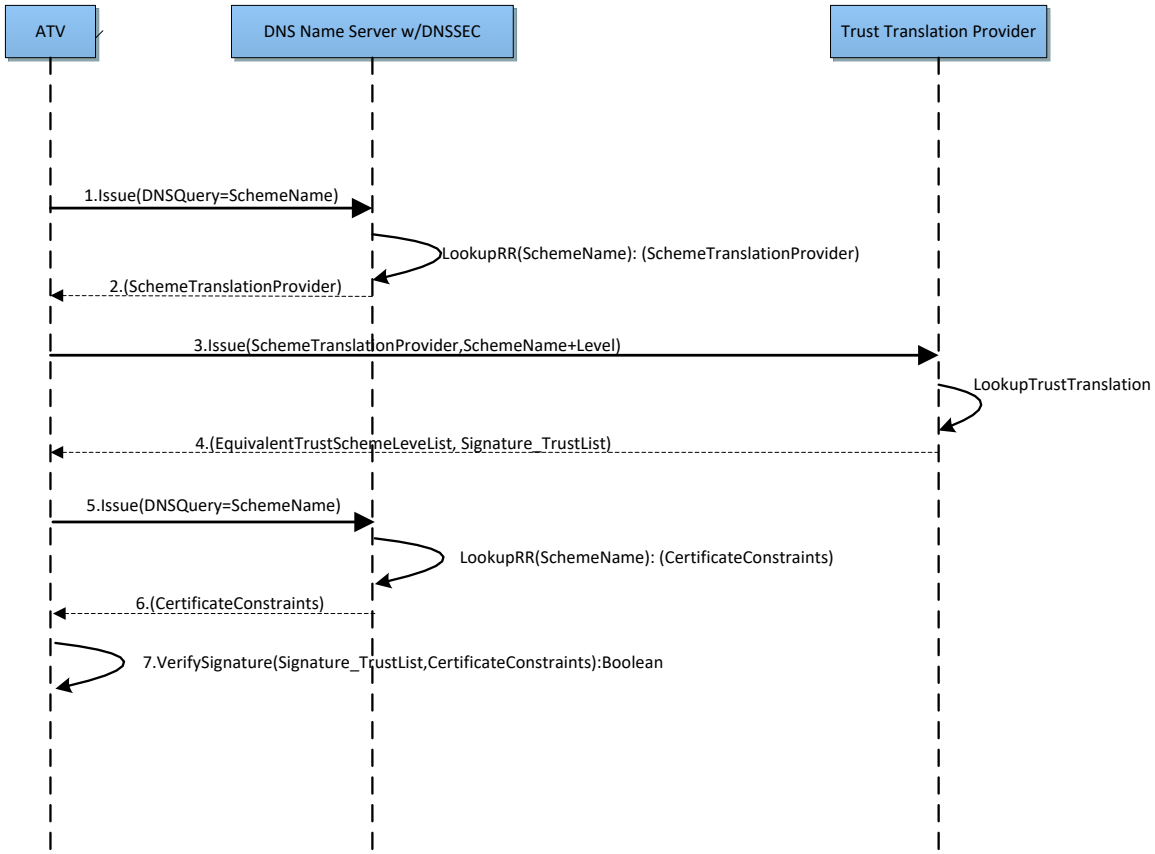

Figure 8: DNS queries in TTA (from D4.4)

Figure 7 shows how ATV query TTA to look for translations. Before this process can take place, ATV has first to query TSPA to look for a valid name for the scheme used in the transaction.

Following with the example, we assume that TSPA has provided the name *fido2fingerprint.attestation.fidouaf.eu* for FIDO scheme and the set of tuples specified before. Then ATV must query TTA about the available translation for that name. In other words, ATV query about the target of Trust, as this is the one not valid for the policy and it needs to be trusted.

To this end ATV sends a DNS query to LIGHTest DNS servers building the domain name as specified in D4.4 section 6.2 [30]:

- _translate._trust. *fido2fingerprint.attestation.fidouaf.eu*

DNS servers start analyzing the domain name by the end; first the look for the DNS serving for *EU* then for *fidouaf*, etc. till found the DNS holding a URI for that name.

```
;; QUESTION SECTION: Client/ATV to the TTA
;_translate._trust.fido2fingerprint.attestation.fidouaf.eu.  IN  URI

;; ANSWER SECTION: from the TTA
_translate._trust.fido2fingerprint.attestation.fidouaf.eu.   IN  URI
              https://lightest.eu/ttl_fido2fingerprintAttestation1.tpl
_translate._trust.fido2fingerprint.attestation.fidouaf.eu.   IN  URI
              https://lightest.eu/ttl_fido2fingerprintAttestation1.xml
```

Figure 9: steps 1 and 2 of Figure 7

The URIs provided point to a TPL and XML files where the translation is described.

ATV can use these URIs with an HTTP/HTTPS GET method to retrieve the files.

```
GET /ttl_fido2fingerprintAttestation1.tpl HTTP/1.1
Host: lightest.eu
Connection: keep-alive
Cache-Control: max-age=0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: …
Accept-Encoding: …
Accept-Language: en-US,en;q=0.8
```

Figure 10: steps 3 and 4 of Figure 7

Finally, TTA provides the mechanism to check the validity of the information provided by using DANE extension in the DNS server. This checking procedure is introduced in D2.7 section 7.3 [27].

This checking procedure which corresponds to steps 5 and 6 in figure 7 is performed in the same way than in TSPA to present a consistent tool.

7.2.3 Delegation Publisher

A Delegation Publisher operates an off the shelf DNS server with DNSSEC extensions turned on together with a web component providing the details of a delegation. A Delegation Publisher publishes multiple delegations.

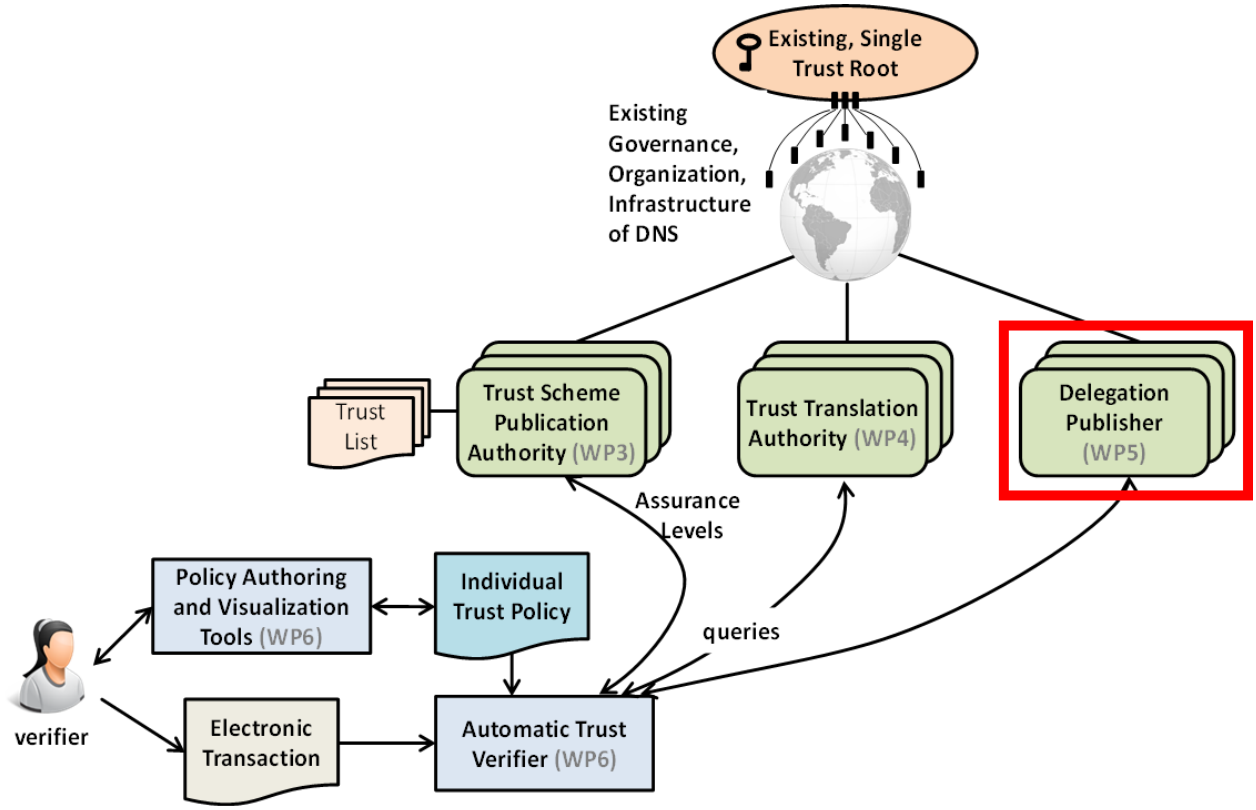

Figure 11: Delegation Provider

The Delegation Publisher; described in [31], and shown in Figure 11: Delegation Provider; consists of two components. One component is the DNS Server with DNSSEC extensions turned on. This is only required to find the Delegation Provider as such. The other component is the Delegation Provider.

The Delegation Provider, which provides a RESTful API to query for delegations and their status. Especially since the status of a delegation is of interest for the ATV.

The Delegation Publisher provides the ability to create the most basic types of delegations, namely (i) bilateral type, (ii) substitution type, and (iii) delegation type.

The sequence diagram in Figure 12: Sequence Diagram for Trust Delegation Scenario (from [26]) shows the communication between ATV and Delegation Publisher. For a better readability, the messages send between the ATV and the Delegation Publisher relevant for retrieving the delegation details are highlighted. The relevant interaction is taking place from step 10 to step 12.

In step 10, the ATV discovers the correct Delegation Publisher. As a company can run multiple Delegation Publishers, it is required for the ATV to identify the correct one. Therefore, the ATV extracts the information from the delegation embedded in the transaction container. In general,

this is a normal URL, identifying the Delegation Provider. With this URL, the ATV can query the Delegation Provider for the status of the delegation.

In the next step, step 11, the ATV issues a query to the Delegation Publisher's revocation list. Depending on the trust policy, the ATV requests the different attributes of the revocation if available.

As a last step; Step 12; the ATV needs to verify the signatures included in the delegation to verify its correctness. This is usually done without the Delegation Provider.

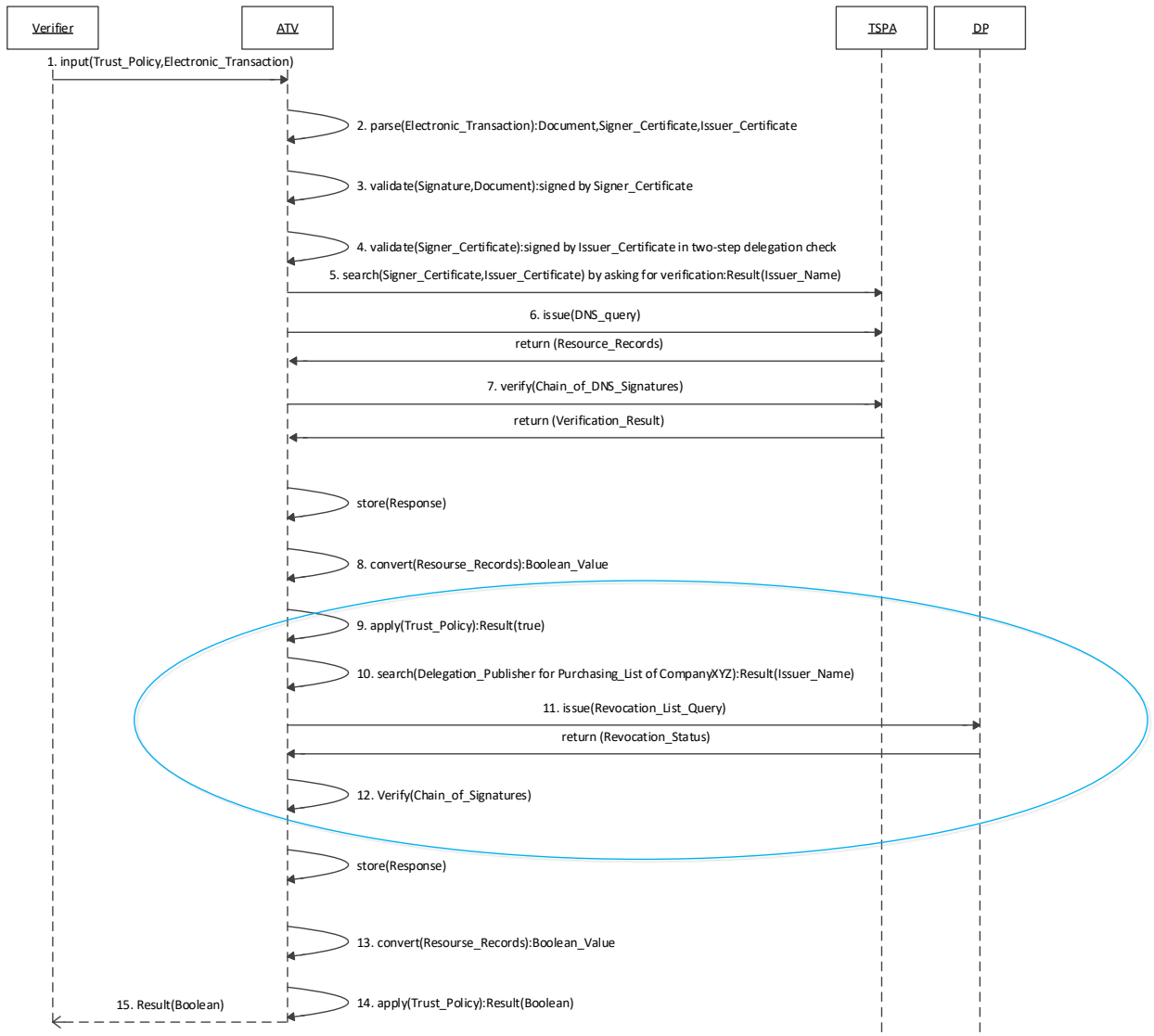

Figure 12: Sequence Diagram for Trust Delegation Scenario (from [26])

7.2.3.1 Communication between the Delegation Provider and the ATV

The design of the communication with the Delegation Provider in full detail is found in [32] [33]. The communication with the delegation provider can be kept at a minimum, as the required data is already contained in the transaction itself. The transaction is saved in an ASiC container. This container is extended to hold the delegation information. This information is saved in a file named *delegation*.xml. The XML is formatted as described in [31]. From this file, the ATV can then extract all the necessary information required for the verification of the delegation.

Depending on the policy the user has set, the ATV has to do several other steps with the delegation. For the most basic step, the verification of the delegation, the ATV has all necessary information at hand and does not need any communication with the delegation provider. However, if the ATV needs to check if the delegation is still valid at the delegation provider, meaning that the delegation has not been revoked by the mandator, the ATV needs to communicate with the delegation provider. Therefore, ATV needs the domain name of the delegation provider. The domain name is found in the delegation file. Next, the ATV calculates a hash value (SHA256) of the delegation data contained in the delegation.xml file. This information is then sent to the delegation provider via its RESTFUL API. The RESTFUL API provides an endpoint called revoked, which receives the delegation data hash as a parameter. With this parameter the delegation provider then searches if this particular delegation has been revoked in the past and is not valid anymore. The result of this query is either a JSON/XML file with the result false if the delegation is not revoked, or true if the delegation has been revoked. In this case the delegation provider can also provide a reason for the revocation in the result. The tags for this are result and reason.

The result file:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
    <result>true</result>
    <reason>Terminated contract</reason>
    <date>26. Jun. 2018</date>
```

Now the ATV knows if the delegation is revoked or not. In the case, that the delegation is not revoked, the ATV can start verifying the signatures embedded in the delegation. If the signatures there are valid as well, the ATV can trust the delegation.

8. References

[1] LIGHTest, "Project Deliverable - D2.2 Inventories (2)," 2017.

[2] LIGHTest, "Project Deliverable - D2.14 Reference Architecture," 2017.

[3] OECD Expert Group on Defining and Measuring E-commerce, “Electronic Transaaction,” Organisation for Economic Co-operation and Development, [Online]. Available: https://stats.oecd.org/glossary/detail.asp?ID=758. [Accessed April 2017].

[4] “REGULATION OF THE GOVERNMENT OF THE REPUBLIC OF INDONESIA NUMBER 82 OF 2012,” April 2017. [Online]. Available: http://www.flevin.com/id/lgso/translations/JICA%20Mirror/english/4902_PP_82_2012_e.html.

[5] Wikibooks, “E-Commerce and E-Business/Concepts and Definitions,” [Online]. Available: https://en.wikibooks.org/wiki/E-Commerce_and_E-Business/Concepts_and_Definitions. [Accessed April 2017].

[6] American Bar Association, Digital Signature Guidelines, Chicago: American Bar Association, 1995, 1996.

[7] J. Davies, “It’s time to talk about eIDAS…,” June 2016. [Online]. Available: https://www.signable.co.uk/blog/eidas-what-is-it-how-does-it-affect-your-electronic-signatures. [Accessed April 2017].

[8] V. P. K. Shyamasundar, “Trust management for e-transactions,” *Sādhanā,* vol. 30, no. 2, pp. 141-158.

[9] E. Union, “Building online trust and confidence: Electronic signatures, seals and trust services now valid throughout EU,” European Union, [Online]. Available: https://ec.europa.eu/digital-single-market/en/news/building-online-trust-and-confidence-electronic-signatures-seals-and-trust-services-now-valid. [Accessed 18 4 2017].

[10] D. R. K. a. D. F. F. Vincent C. Hu, “Attribute-Based Access Control,” *Computer,* vol. 48, no. 2, pp. 85-88, 2015.

[11] D. R. K. David F. Ferraiolo, “Role-Based Access Controls,” in *15th National Computer Security Conference*, Baltimore, 1992.

[12] “Secure Electronic Transactions : An Overview,” [Online]. Available: http://www.davidreilly.com/topics/electronic_commerce/essays/secure_electronic_transactions.html. [Accessed 18 04 2017].

[13] L.-l. W. J.-f. Z. X. M. Xun-yi Ren, “The Improvement of SET Protocol based on Security Mobile Payment,” *Journal of Convergence Information Technology,* vol. 6, no. 7, 2011.

[14] M. T. S. C. S. S. S. Rajdeep Borgohain, “TSET: Token based Secure Electronic Transaction,” *INTERNATIONAL JOURNAL OF COMPUTER APPLICATIONS,* 2012.

[15] E. F. A. E. a. Y. A. Hassan M. Elkamchouchi 1, “AN IMPROVEMENT TO THE SET PROTOCOL BASED ON SIGNCRYPTION,” *International Journal on Cryptography and Information Security,* vol. 3, no. 2, 2013.

[16] European Telecommunications Standards Institute, “Electronic Signatures and Infrastructures (ESI); Associated Containers (ASiC); Part 1: Building blocks and ASiC baseline containers (ETSI EN 319 162-1 V1.1.0),” European Telecommunications Standards Institute, Sophia Antipolis Cedex, 2016.

[17] Adobe Systems Incorporated, *The AdES family of standards: CAdES, XAdES, and PAdES,* San Jose, 2009.

[18] Internet Engineering Task Force, “rfc3161: Time-Stamp Protocol,” Internet Engineering Task Force, 2001.

[19] Internet Engineering Task Force, “rfc4998: Evidence Record Syntax (ERS),” Internet Engineering Task Force.

[20] Internet Engineering Task Force, “rfc6283: Extensible Markup Language Evidence Record Syntax (XMLERS),” Internet Engineering Task Force, 2011.

[21] M. M. N. R. Herzberg, “Access control meets public key infrastructure, or: Assigning roles to strangers.,” *21th IEEE Computer Society Symposium on Research in Security and Privacy,* 2000.

[22] M. Blaze, “RFC 2704 - The KeyNote trustmanagement system,” IETF.

[23] W. F. Yao, “A Policy-Driven Trust Management Framework. In Proceedings of Trust Management,” in *In Proceedings of Trust Management, LNCS 2692*.

[24] J. Hoagland, “Specifying and Implementing Security Policies Using LaSCO, the Language for Security Constraints on Objects,” 2000.

[25] D. a. L. J. Lomsak, “PoliSeer: A Tool for Managing Complex Security Policies.,” in *Journal of Information Processing*, 2011.

[26] LIGHTEST, “Project Deliverable - D3.4 Discovery of Trust Scheme Publication Authorities,” LIGHTEST, 2018.

[27] LIGHTEST, “Project Deliverable - D2.7 Relevant DNSSEC Concepts and Basic Building Blocks,” Lightest, 2017.

[28] R. Arends, T. Instituut, R. Austein, ISC, M. Larson, VeriSign, D. Massey, C. S. University, S. Rose and NIST, “Protocol Modifications for the DNS Security Extensions,” March 2005. [Online]. Available: https://tools.ietf.org/html/rfc4035. [Accessed 25 July 2018].

[29] S. Weiler, SPARTA, D. Blacka and eriSign, “Clarifications and Implementation Notes for DNS Security (DNSSEC),” Internet Engineering Task Force (IETF), 2013.

[30] LIGHTEST, “Project Deliverable - D4.4 Discovery of Trust Translation Authorities,” LIGHTEST, 2018.

[31] G. Wagner, “D5.1 Conceptual Framework for Delegations,” 2017.

[32] The LIGHTest Project, D5.3 - DNS-based Publication of Delegations, 2018.

[33] The LIGHTest Project, D5.4 - Discovery of Delegations, 2018.

[34] L. Rosa, “D4.1 Conceptual Framework for Trust Scheme Translation,” 2017.

[35] H. Roßnagel, "D2.14 Reference Architecture," 2017.

[36] LIGHTest, "Project Deliverable - D6.1 Requirements and Design of a Conceptual Framework for Trust Policies," 2017.

9. Project Description

LIGHT^{est} project to build a global trust infrastructure that enables electronic transactions in a wide variety of applications

An ever increasing number of transactions are conducted virtually over the Internet. How can you be sure that the person making the transaction is who they say they are? The EU-funded project LIGHT^{est} addresses this issue by creating a global trust infrastructure. It will provide a solution that allows one to distinguish legitimate identities from frauds. This is key in being able to bring an efficiency of electronic transactions to a wide application field ranging from simple verification of electronic signatures, over eProcurement, eJustice, eHealth, and law enforcement, up to the verification of trust in sensors and devices in the Internet of Things.

Traditionally, we often knew our business partners personally, which meant that impersonation and fraud were uncommon. Whether regarding the single European market place or on a Global scale, there is an increasing amount of electronic transactions that are becoming a part of peoples everyday lives, where decisions on establishing who is on the other end of the transaction is important. Clearly, it is necessary to have assistance from authorities to certify trustworthy electronic identities. This has already been done. For example, the EC and Member States have legally binding electronic signatures. But how can we query such authorities in a secure manner? With the current lack of a worldwide standard for publishing and querying trust information, this would be a prohibitively complex leading to verifiers having to deal with a high number of formats and protocols.

The EU-funded LIGHT^{est} project attempts to solve this problem by building a global trust infrastructure where arbitrary authorities can publish their trust information. Setting up a global infrastructure is an ambitious objective; however, given the already existing infrastructure, organization, governance and security standards of the Internet Domain Name System, it is with confidence that this is possible. The EC and Member States can use this to publish lists of qualified trust services, as business registrars and authorities can in health, law enforcement and justice. In the private sector, this can be used to establish trust in inter-banking, international trade, shipping, business reputation and credit rating. Companies, administrations, and citizens can then use LIGHT^{est} open source software to easily query this trust information to verify trust in simple signed documents or multi-faceted complex transactions.

The three-year LIGHT^{est} project starts on September 1st and has an estimated cost of almost 9 Million Euros. It is partially funded by the European Union's Horizon 2020 research and innovation programme under G.A. No. 700321. The LIGHT^{est} consortium consists of 14 partners from 9 European countries and is coordinated by Fraunhofer-Gesellschaft. To reach out beyond Europe, LIGHT^{est} attempts to build up a global community based on international standards and open source software.

The partners are ATOS (ES), Time Lex (BE), Technische Universität Graz (AT), EEMA (BE), G&D (DE), Danmarks tekniske Universitet (DK), TUBITAK (TR), Universität Stuttgart (DE), Open Identity Exchange (GB), NLNet Labs (NL), CORREOS (ES), IBM Danmark (DK) and Ubisecure (FI). The Fraunhofer IAO provides the vision and architecture for the project and is responsible for both, its management and the technical coordination.

The Fraunhofer IAO provides the vision and architecture for the project and is responsible for both, its management and the technical coordination.