D5.3

DNS-based Publication of Delegations

Document Identification	
Date	28.02.2018
Status	Final
Version	Version 1.0

Related WP	WP 5	**Related Deliverable(s)**	D5.1
Lead Authors	TUG	**Dissemination Level**	PU
Lead Participants	TUG	**Contributors**	USTUTT, FHG, NLNET, IBM
Reviewers	CORREOS, ATOS		

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. Disclaimer

During the cause of the project we came to the conclusion that is better to not use DNS for the publication of delegations. Therefore it contradicts the LIGHTest reference architecture [1]. All deviations from [1] are intentional and update the reference architecture. Furthermore, this document also outdates the current version of the conceptual framework [2]. The second iteration of the conceptual framework will contain the updated information on the publication of delegations.

While the Trust Scheme and the probably required translation can be discovered from the certificate [3] [4] [5] [6], this assumption does not hold for the delegation. While the trust scheme can be determined by the evaluation of country codes stored in the certificate, we would need to extend the certificate for delegations. We think that the certificate should not be altered for the use of LIGHTest and thus need to find other ways to publish and discover delegations.

DNS, DNSSec and DANE will further be used within this work package to discover the location (IP address) of the delegation provider, given that the user knows the correct domain name. With DNSSec and DANE we can determine if the Delegation Provider is the correct one accessible from the given domain and not an adversary trying to impersonate the Delegation Provider we want to access.

2. Executive Summary

Delegation is the process of transferring legal powers (or powers of attorney), duties, etc. from an entity A to an entity B, allowing entity B to act on behalf of entity A. This can be done in various ways. Either with or without an arbitrary number of Intermediaries. Intermediaries act between the Mandator and the Proxy and can select a Proxy for a Mandator, only if the Mandator allows this.

Section 6 defines the terminology of the most important concepts and terms used in this document. This section is thought to be a basis for the further understanding of the document. For a full terminology please refer to D2.14 [1].

Section 7 describes the notation used for mathematical and cryptographic formulas. They provide the reader with a common understanding what different constructs throughout the document mean and how they should be understood.

In Section 8, we describe the consolidated data model used throughout WP3, WP4, and WP5. Although the decision to not use DNS for the publication of delegations, this section provides an insight into the problems that occurred while this document was written.

Section 9 describes the delegation data model and the processes behind the publication in detail. First this section defines the new role of the Delegation Provider in the project and explains why the Delegation Provider can be seen as an insecure storage. Further this section describes the need for the secure storage of delegation data and how this can be done on an insecure storage.

In Section 9.1,we explain the new role of the Delegation Provider. The Delegation Provider needs to store and provide Delegations and does not need to alter any DNS zone files to discover and publish delegations. We further describe why the Delegation Provider shall be seen as an untrusted source (Section 9.1.1) and how delegations can be stored with the Delegation Provider (Section 9.1.2).

In Section 9.2, we work on the publication process for delegations. To delegate powers a Mandator chooses a Proxy or Intermediary. The Mandator then chooses the attributes that should be delegated and signs the delegation document to have some proof that the Mandator has created the delegation and that it has not been altered. Next, the Mandator creates a symmetric key to encrypt the delegation document. Now the Delegation Provider is not able to see the details of the delegations as such, but neither is the Proxy. Therefore, the Mandator encrypts the symmetric key with the public key of the Proxy and uploads the key to the Delegation Provider. The Proxy can then download the encrypted symmetric key, together with the delegation document and decrypt both. If the delegation is issued via an Intermediary, the Intermediary executes the same steps as the Proxy to receive the delegation. The chosen Proxy is then signed entered in the delegation document and signed by the Intermediary.

In Section 9.3 we describe the delegation tools, namely the Application Programming Interface (API) that the server needs to implement and we provide a prototype for the Mandator's and Intermediary's user interface to create delegations.

3. Document Information

3.1 Contributors

Name	Partner
Georg Wagner	TUG
Reinhard Lüftenegger	TUG
Olamide Omolola	TUG
Stefan More	TUG
Martin Hoffmann	NLNET
Stephanie Weinhardt	USTUTT
Heiko Roßnagel	FHG

3.2 History

Version	Date	Author	Changes
0.1	27.11.2017	Georg Wagner	Disclaimer added Table of Content added First draft of the document created
0.2	10.1.2018	Georg Wagner, Olamide Omolola	Software Components added, ASiC Container description
0.3	14.1.2018	Georg Wagner, Reinhard Lüftenegger, Stefan More	Rework of delegation process
0.4	25.1.2018	Georg Wagner, Stephanie Weinhardt	Delegation Provider GUI design and description
1.0	9.2.2018	Georg Wagner	Final spellchecking and semantic review

4. Table of Contents

4.1 Table of Figures

4.2 Table of Tables

4.3 Table of Acronyms

ATV	Automated Trust Verifier
DP	Delegation Provider
TSPA	Trust Scheme Publication Authority
TTL	Trust Translation List
DNS	Domain Name System
DNSSEC	DNS Security Extensions
DANE	DNS based Authentication of Named Entries
OEBPS	Open EBook Forum Publication Structure
OCF	OEBPS Container Format
ODF	Open Document Format - Open Office
UCF	Universal Container Format
ASiC	Associated Signature Container
IDPF	International Digital Publishing Forum

5. Scope of the Deliverable

This deliverables describes the design of the publication of delegations without the use of the data elements and infrastructure of the DNS. The publication design is based on the conceptual framework for delegations, except the DNS part, as described in D5.1 [2]. In D5.1 a concept for delegations is elaborated, which consists of two components: a DNS name server with DNSSEC extensions and a Delegation Provider. The current design for the publication of delegations has changed so that the delegation provider now does not use a DNS name server with DNSSEC extensions anymore (see section 1 for a more detailed explanation).

For this deliverable, a consolidated approach together with WP3 and WP4 was chosen. Therefore, this document contains a general approach to trust related publications and is thus specified in all three deliverables. This document starts with this approach. However, this approach is similar to D3.3 and D4.3 which leads to some duplications, but provides standalone and self-contained documents for the reader.

The remainder of this document then contains the design for the publication of delegations. For this design, only a web component is used as the delegation provider. This web component provides and stores the delegations for later use. A Mandator has to fulfil certain steps to create a delegation. As everybody can create a delegation provider, we do not know how honest the delegation provider or its administrators are, as they have access to the plain text data. Therefore, several encryption schemes are introduced to overcome this problem.

This document also contains a component description for the delegation provider. This consists of two classes, one for management tasks and one for user related tasks. We also provide a user interface to create delegations.

6. Terminology

This section contains the terminology used throughout the deliverable. It is important, that all readers of this document have the same understanding of the concepts. Furthermore, it makes communication between partners and participants easier. Parts of the terminology are reused from D2.14 [1] and D5.1 [2] to provide a comprehensive terminology dictionary in this document.

6.1 Delegation Provider

The Delegation Provider is a web component, which is used to publish delegations. It is used by other components by querying it. The Delegation Provider contains a repository for delegations and one for revoked delegations.

6.2 Verifier

Person or Entity that wants to check if a transaction is valid or not.

6.3 Automated Trust Verifier

Component used by the Verifier to verify documents. This component queries the delegation provider to verify the existence of the delegation and verifies that the delegation has not been revoked.

6.4 Mandator

Person or entity empowering another person or entity to act on behalf of itself. The Mandator is the creator of delegations and publishes delegations at the Delegation Provider of his choice.

6.5 Proxy

Person or entity empowered by a Mandator to act on behalf of another person or entity. The Proxy is the person or entity executing the delegation.

6.6 Intermediary

Person or entity empowered by a Mandator to find another person or entity to act on behalf of the Mandator. The intermediary acts as a link between Mandator and Proxy in the selection process. The parameters of the delegation are provided by the Mandator.

7. Notation

The following section describes the mathematical notation used in this document:

- All upper case letters define a specific function.
- Lower case letters usually specify some data, e.g. parameters or results of a function.
- E and D denote encryption and decryption, where E stands for encryption and D for decryption.
- H defines an arbitrary hash function.
- Lower case d is the delegation message, with the parameters.
- The letters *pk* define a public key. The letters *sk* define a secret or private key from a person or entity.
    - A small letter in the subscript an additional subscripted index to a key determines whom the key belongs.
- The letter *k* denotes a symmetric key

8. A Consolidated Approach to Publishing Trust-related Information in the DNS

The trust framework created by LIGHTest verifies the trustworthiness of an electronic transaction by attempting to establish a chain of trust from a set of pre-configured, well-known trust sources to this transaction. The links in this chain are assurances that the trust into a source can be extended to another entity. This section looks at the basic properties these assurances exhibit independently of their concrete contents and introduces an underlying, fundamental framework.

Within the LIGHTest project, three working packages look at different aspects of such assurances. WP3 examines the most basic form where a trust source known as a trust scheme declares that some issuer of trust services, such as certificates or time stamps, conforms to the conditions and rules set out by the scheme and thus extends trust placed into it onto this trust service. WP4 assesses the relationship between multiple trust schemes allowing a trust scheme or some other trusted entity to declare if and how trust into one scheme extends to trust into another scheme. WP5, finally, looks into how individual entities can empower other entities to act on their behalf – extending trust into themselves onto that other entity within certain well-defined limits.

8.1 Trust Declarations

However different they may appear, each of these aspects follows a similar pattern: some entity makes a *trust declaration* stating that trust into a certain entity extends to another entity, possibly providing conditions and limits of such an extension of trust. To simplify further discussion, it will be helpful to label the three entities involved in the process. The entity issuing the declaration shall be the *originator,* the entity that is already trusted is the *source* and the entity trusted as a result of the declaration is the *target.*

Within the aspects discussed as part of the LIGHTest project, in many cases the originator of a declaration is identical to the source. This is certainly true for trust membership publication in WP3, where the trust scheme itself declares which trust services are a member. In the aspects of the other two work packages, similarly originator and source are most often identical. However, there may be use cases where this is not the case. For instance, a third party may declare a trust translation independently of the trust schemes that are source or target of this declaration. Similarly, a third party may declare a trust delegation. For instance, business registries often provide information about individuals that are allowed to sign on behalf of a company. Such information can be modeled as third-party trust delegation.

This definition specifies a single declaration to extend trust from exactly one source to exactly one target. In practice, the document formats used often contain multiple declarations according to this definition. For instance, the trusted lists defined in ETSI TS 119 612 used for the declarations in WP3 contain a list of all the targets a trust scheme as a source wishes to extend trust to. This published form of declarations shall be called *trust declaration documents.* At least

in principle each such document can contain one or many declarations with any number of sources and targets.

For a trust declaration to be considered when building the chain of trust during validation of an electronic transaction, the declaration itself needs to be trusted. If a declaration document is treated like any other electronic transaction, this trust can in turn be established through verification using the LIGHTest framework. That is, there needs to be a chain of trust from pre-configured trust sources to the originator of the declaration document for the particular aspect of the declaration in question.

Note that in most cases where the originator of the declaration is identical to the source of the declaration, this chain exists implicitly and no extra checks are needed. It may, however, be possible that conditions for trusting the source as such and trusting declarations made by it are different and thus need to be verified independently.

8.2 Publication of Trust Declarations

When using trust declarations for verifying an electronic transactions, a validator needs to construct a chain of trust declarations leading from any of the trusted entities to those entities appearing in the transaction. It does so by recursively finding and adding applicable trust declarations that have an originator that is either a trusted entity or the target of the a declaration already part of the chain. In order to do this in an unaided, automatic way, the validator needs a way to gain access to all declarations that are potentially usable in this process.

There are two fundamental strategies for the verifier to find declarations when they are needed: a declaration could either be actively supplied as part of the input or configuration or it is left to the verifier itself to discover it.

The prime example for the former case is that a declaration is provided as part of the electronic transaction to be verified. This is particularly useful if the creator of the transaction is aware that the declaration is necessary for verification. For instance, if an entity signs a document in their function as a proxy, the transaction will only ever verify if the declaration of trust delegation – the mandate – is known to the verifier. The proxy may very well include the mandate in the transaction right away.

For declarations that are potentially applicable to a large amount of transactions or if the sender of the transaction doesn't know the trusted entities the receiver will base their verification on, such a strategy isn't very practical. Instead, it is better to make the declarations available publicly and provide means for a verifier to discover how and where it can retrieve them. The verifier can then decide itself which declarations it needs and try and find them as needed.

While the most likely method for publishing currently is the Hypertext Transfer Protocol (HTTP), other protocols may become available in the future. An extensible standard exists to describe both the method used for accessing a resource and all necessary parameters for a successful retrieval in the form of a Uniform Resource Identifier (URI). It encodes all information into a single string which can be easily stored or transmitted.

8.3 Discovering Trust Declarations

In order to build a chain of trust from published declarations, the verifier needs to be able to discover their existence. Given a transaction and a set of already trusted entities, this chain can be built from two sides: either the verifier starts with a trusted entity, tries to discover all the declarations that have this entity as their source, and repeats this process until it arrives at a declaration that includes the transactions as its target or runs out of declarations to apply. Alternatively, it can start with the transaction, attempts to discover all declarations that have the transaction as their target, and continues by recursively trying to find declarations that have the sources of already discovered declarations as their target until it arrives at an already trusted entity as a declaration's source or, again, runs out of declarations.

In both cases, the verifier needs a mechanism to search for the URI of a declaration based on a given entity. Such a mechanism will have to take some information that identifies that entity as input. Given that entities are typically identified by X.509 certificates, it should be possible to include such identifying information in the entity's certificate. Once of the possible options is to use a domain name as the entity's identifier. This has the advantage that the name can be used directly as a search input for the DNS, allowing to use the DNS as a global, highly available, distributed, and independently managed data store.

A domain name can be stored in a certificate either in the subject alternative name or issuer alternative name extensions. The subject alternative name indicates the domain name identifying the entity using the certificate while the issuer alternative name identifies the entity having issued the certificate.

Using this domain name as input, the URIs to the declarations for differing aspects should be stored in the DNS. As the DNS uses record types to distinguish between different types of information stored for a domain name, one option is to register an individual record type for each aspect of trust declarations. However, this would clutter the space of types. As the data stored is the same in every case – a URI – an alternative approach can be used whereby the aspect is encoded as a prefix to the domain name. This is already used for instance with the SRV record type for discovering the host name and port where a certain networking service is available for a domain. As such services sometimes are available over different transport protocols, a two-layer prefix of the form *_service._protocol* is used where the latter describes the transport protocol to be used and the former the networking service.

In keeping with this concept, LIGHTest proposes to use the DNS for providing pointers to all kinds of declarations via a pair of prefixes of the form *_aspect._application*. Here, the type of application for which a declaration is published is the second part of the prefix while the first part defines the particular aspect within that application. For LIGHTest itself, the application is, of course, trust-related declarations, which is identified via using the literal label *_trust* as the second part. Each of the aspects of trust declarations defines its own label to be used as the first part.

Under the name constructed by concatenating the prefix with the entity's own identifying domain name, the entity can now publish pointers to trust declarations of the corresponding aspect that relate to it. It can use the already existing URI resource record type for this purpose. This type is defined in section 4 of RFC 7553 [7]. Its record data contains exactly one URI. While section 5 of the RFC describes a different use of the record, this use is limited by a different set of prefixes, allowing its reuse for declaration publication based on the prefixes defined above.

If the entity in question is not the originator of a declaration it may not control the URI under which the declaration is published. In this case, it may be beneficial to only point to the originating entity rather than burden itself with tracking whether the originators URI has changed. This can be done easily if the originator is an entity identified by a domain name, too. In this case, instead of publishing URI resource records under the its domain name prefixed by the declaration aspect, the entity will publish PTR resource records. Such record types, part of the original DNS specification in RFC 1035 [8], contain another domain name as their record data. If such records are present, they instruct the verifier to continue discovery for declaration at the entity identified by these domain names. Note that as these names in the PTR record's data identify entities, they, too, need to be prefixed with the correct declaration prefix before continuing querying.

8.4 Authenticity of Trust Declarations

If a verifier retrieves a declaration from somewhere in the network, it needs to make sure that the data it received is indeed the declaration made by the originator. Since the URI resource records are stored under the domain name identifying the originator, it is reasonable to assume they are authentic if DNSSEC validation succeeds, guaranteeing that the URI is indeed the one intended by the originator.

In the next step, where the verifier contacts the server indicated in the URI, it needs to ensure that it communicates with the correct server. When using encrypted transport via the TLS protocol, the server will identify itself via a certificate. In order to deal with shortcomings of the traditional method of certificate verification, a protocol called DNS-based Authentication of Named Entities or DANE for short allows a server operator to publish information about the certificates used in DNS.

Since, however, the server is not necessarily operated under authority of the originator of the declaration – for instance because the declarations are hosted by a third party that provides better availability, this does not guarantee that the declaration received is indeed the one that the originator intended.

This final link can be provided if the declaration itself is a signed document. The originator can then publish the certificates that it uses for signing declarations of a certain aspect using a slightly adapted version of the DANE protocol.

To do so, it adds SMIME resource records under the same domain name it placed the URI records pointing to the declarations. These records define conditions a certificate has to fulfill to be

accepted. By placing these records, the originator declares that all documents retrieved via the pointers have to verify considering these conditions.

9. Delegation Data Model

9.1 The new Role of the Delegation Provider

9.1.1 The Delegation Provider as Untrusted Source

The main tasks of the Delegation Provider are

- storing delegation information and
- responding to queries regarding the discovery of delegations.

To fulfil its purpose the Delegation Provider needs **not** to know the explicit content of a delegation and in fact should not know it. Despite the fact that the Delegation Provider is assumed to guarantee the availability of delegation information, no further trust hast to be put in it. This is subsumed under the phrase “The Delegation Provider as Untrusted Source”. Whether the Delegation Provider is self-hosted or hosted via a third-party service, it is as a public source of information. Therefore, to prevent leakage or alteration of potentially confidential delegation information, each delegation will be stored encrypted on the Delegation Provider. The exact flow of information for each delegation type is provided in the corresponding subsection of section 9.2. To query particular delegations, the delegation is part of the transaction (see Section 9.2.4).

9.1.2 Storage of Delegations on the Delegation Provider

Regardless which type of delegation one considers, e.g. with or without an intermediary, each step of a delegation can be thought of a *bilateral arrangement*, i.e. a smallest building block. In the basal case of a Bilateral Delegation Type (see 9.2.1) there are only two involved parties, therefore this type of delegation is by the very definition a bilateral arrangement. In a Substitution Delegation Type (see 9.2.2) the overall information flow can be broken down into two individual bilateral arrangements: the first arrangement involves the delegation from the Mandator to the Intermediary, the second arrangement is the delegation from the Intermediary to the Proxy. Eventually, in a Delegation Type (see 9.2.3) the Intermediary exists only temporary, which means in the end it looks like the Proxy has received its delegation directly from the Mandator. This contrasts the result in a Substitution Type of delegation, because there the Proxy’s delegation document contains a mandate from the Intermediary and **not** from the Mandator. However, the information flow in a Delegation Type is the same as in a Substitution Type: there is a bilateral arrangement between Mandator and Intermediary and an arrangement between Intermediary and Proxy.
That said, care must be taken when it comes to the actual bilateral arrangement in each type of delegation: on a high level, the Substitution and Delegation Type of delegation can be described in the same manner, but it is important to note that in a more low-level perspective all involved bilateral arrangements can differ! The details, including the appropriate encryption of the delegation information, will be discussed in the next section.

9.2 Publication of Delegations on the Delegation Provider

Starting from the conceptual framework for delegations [2] we know that three different types of delegations do exist. Namely, the three types are (i) bilateral, (ii) substitution, and (iii) delegation type. These three types are published at the Delegation Provider. The Delegation Provider acts as a data storage for those delegations. We want to keep the burden of having the Proxy managing all given delegations by itself. Therefore, the Delegation Provider needs to store the delegation data itself. Although, we are going to embed it in the signature container later on, we still do not want to burden the Proxy with the need to save its delegations locally.

In a general perspective, the publication process of a delegation can be divided into two steps. The first publication is from the Mandator to the Delegation Provider. This is required so that the Proxy can query the delegation provided by the Mandator later. This first publication step makes the delegation publicly available for the Proxy and the Verifier. The second publication step takes place between the Proxy and the Verifier, where the Proxy reveals the Delegation Details to the Verifier, so that the verifier is able to verify the delegation.

9.2.1 Bilateral Type

The bilateral type is the most basic type of delegation. This delegation has two participants; namely Mandator and Proxy; who interact with each other. As with every delegation, the Mandator delegates some of its attributes to the Proxy. The Proxy can then act on behalf of the Mandator and sign legally binding contracts in the name of the Mandator. Figure 1 shows the creation of a bilateral type of delegation, including its publication by the Mandator and retrieval by the Proxy.

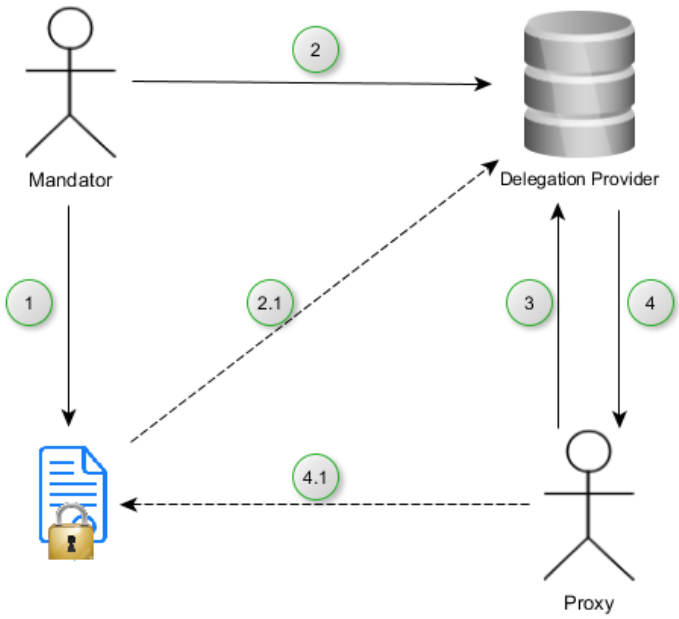

Figure 1: Creating, Publishing, and Usage of a delegation flow for the bilateral type

In the first step, the Mandator needs to create the delegation. The delegation contains the attributes the Mandator wants to transfer to the Proxy. The Mandator needs to sign the delegation file. The signature is later validated by the Proxy when he receives the delegation,

and the Verifier when the delegation is verified. With the Mandator's signature it is verified that the delegation has not been altered since it was issued. Encrypting the delegation is mandatory. In section 9.1.1, the Delegation Provider's definition is given as untrusted source. Therefore, the delegation data must be saved in a way that the DP cannot read the content of the delegation. As the Mandator only needs to communicate the delegation details to the Proxy, it is sufficient enough to encrypt with the pk_{Proxy}. When the Proxy downloads the delegation, the Proxy can easily decrypt the delegation details by using sk_{Proxy}. However, the Mandator should be able to decrypt the delegation in case of dispute between the Mandator and the Proxy over the details of the delegations. Therefore, a different system is required.

For this system, the Mandator needs to create a symmetric key *k*. With this symmetric key, the Mandator encrypts the delegation. The encrypted delegation is then published on the Delegation Provider. Further, the Mandator encrypts *k* with pk_{Proxy}, obtains $k' = E(k, PK_{Proxy})$ and publishes *k*' on the Delegation Provider as well. The Proxy can then download *k*' and decrypt it with sk_{Proxy}, i.e. $k = D(k', sk_{Proxy})$. Now Mandator and Proxy share the same key *k* and both can decrypt the delegation data.

In a second step, the Mandator publishes the delegation on the Delegation Provider. To do so, the Mandator needs to upload the encrypted symmetric key *k*' for the Proxy and the encrypted delegation attributes as well. The Mandator does not require any authentication at the Delegation Provider, as the Delegation Provider will only host the delegation information and provide a search functionality to find the delegations.

The third step is the querying of the delegation by the Proxy. To do this, the Proxy sends a search query to the Delegation Provider. The details of the search request are not part of this document and can be found in [9].

The fourth step answers the query from step three with the encrypted delegation. The Proxy receives two different sets of information. The first set of information is the encrypted key *k*'. The second set of information is the encrypted delegation attributes. The Proxy now needs to decrypt the key *k*' in order to decrypt the delegation attributes. To decrypt *k*' into *k*, the Proxy requires its secret key sk_{Proxy}. Now $k = D(k', sk_{Proxy})$ can be calculated easily. With this new symmetric key *k*, the Proxy can decrypt the delegation attributes and attach them to the signature container, where the verifier can discover them.

9.2.2 Substitution Type

The substitution type is very similar to the bilateral type described in 9.2.1. The delegation is created as two separate bilateral delegations. The first delegation is created between the Mandator and the Intermediary and the second bilateral delegation is created between the Intermediary and the Proxy. The process to create those delegations is similar to the bilateral type in 9.2.1, with the difference that an Intermediary is required. The process is depicted in Figure 2.

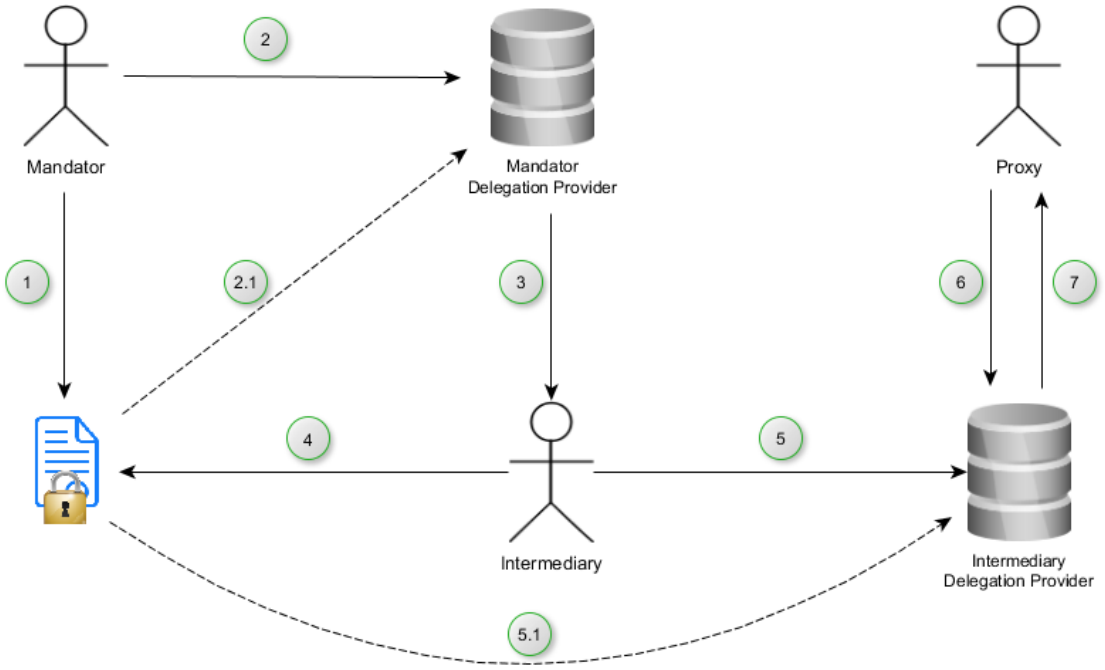

Figure 2: Creating, Publishing, and Usage of a delegation flow for the substitution type

In the first step, the Mandator needs to create the delegation. The delegation contains the attributes the Mandator wants to transfer to the Proxy via the Intermediary. The Mandator needs to sign the delegation file. The Intermediary and Proxy when the delegation is received, and the Verifier later validate the signature. With the Mandators signature we verify that the delegation has not been altered since it was issued. Encrypting the delegation is mandatory. In section 9.1.1, the delegation provider's definition is given as untrusted source. Therefore, the delegation data must be saved in a way that the DP cannot read the attributes of the delegation. As the Mandator only needs to communicate the delegation details to the Intermediary, it is sufficient to encrypt with the $pk_{Intermediary}$. When the Intermediary downloads the delegation, the Intermediary can easily decrypt the delegation details by using $sk_{Intermediary}$. However, the Mandator should be able to decrypt the delegation in case of dispute between the Mandator and the Intermediary over the details of the delegations. Therefore, a different system is required.

Now the Intermediary is allowed to choose a Proxy. Therefore, the Intermediary first loads the delegation, together with the key, from the Mandators delegation provider. The Intermediary decrypts the symmetric key with $sk_{Intermediary}$. The Intermediary decrypts the delegation data from the Mandator and chooses the Proxy. If the Proxy is chosen, the Intermediary encrypts the symmetric key for the Proxy with pk_{Proxy} and uploads this to the Delegation Provider.

This system poses the threat, that the Intermediary can change the whole delegation, without the Mandator knowing. As right now, the Intermediary can change all parameters and sign the delegation for the Proxy. Proxy and Mandator can never really find out how the delegation has changed. The delegation seems legit for the Proxy. To solve this, the Mandator signs all fields, except the Proxy field in the delegation. This ensures that the Proxy and the Verifier can detect intentional changes by the Intermediary. Furthermore, the chain of trust is not violated. The Intermediary now only signs the fields that the Intermediary changes. In this particular case the Intermediary only signs the Proxy field. The Verifier can now ensure that no delegation data has

been altered, by verifying the Mandator's signature. The Verifier can also verify that the Proxy selected by the Intermediary is a valid proxy.

In the next step, the Intermediary has to upload the delegation again. As we are not using DNS anymore to publish delegations, we can simply upload the delegation to the Mandators Delegation Provider, where the Proxy can download it.

The Proxy has the opportunity to download the delegation from both, the Mandators and the Intermediaries Delegation Provider. This depends, where the Intermediary has uploaded the final delegation. As shown in Figure 2, point 5, the Intermediary will use his own Delegation Provider. As both parties, the Mandator and the Intermediary, are free to choose their Delegation Provider it could be that they are using the same Delegation Provider for this purpose.

The decryption of the key and the usage of the delegation for the Proxy is now similar to the bilateral type described in 9.2.1. The Proxy can download the symmetric key used for the encryption of the delegation from the Delegation Provider. As a next step, the Proxy has sk_{Proxy} and can decrypt the symmetric key. Now the Proxy can verify if the signatures from the Intermediary and the Mandator are correct. If the signatures are correct, the Proxy can use the delegation.

9.2.3 Delegation Type

The delegation type is the last type that can be represented with a delegation. It is very similar to the substitution type; described in section 9.2.2; when it is set up, while the delegation type is executed more like a bilateral type; described in section 9.2.1. For the delegation type, two bilateral delegations need to be created. The first bilateral delegation is created between the Mandator and the Intermediary. The Intermediary then creates a bilateral delegation between the Intermediary and the Proxy. This last delegation is then recreated to look like a bilateral delegation between the Mandator and the Proxy for verification. The intermediary is not used anymore after that link between Mandator and Proxy has been established. An outlook of the information flow is given in Figure 3.

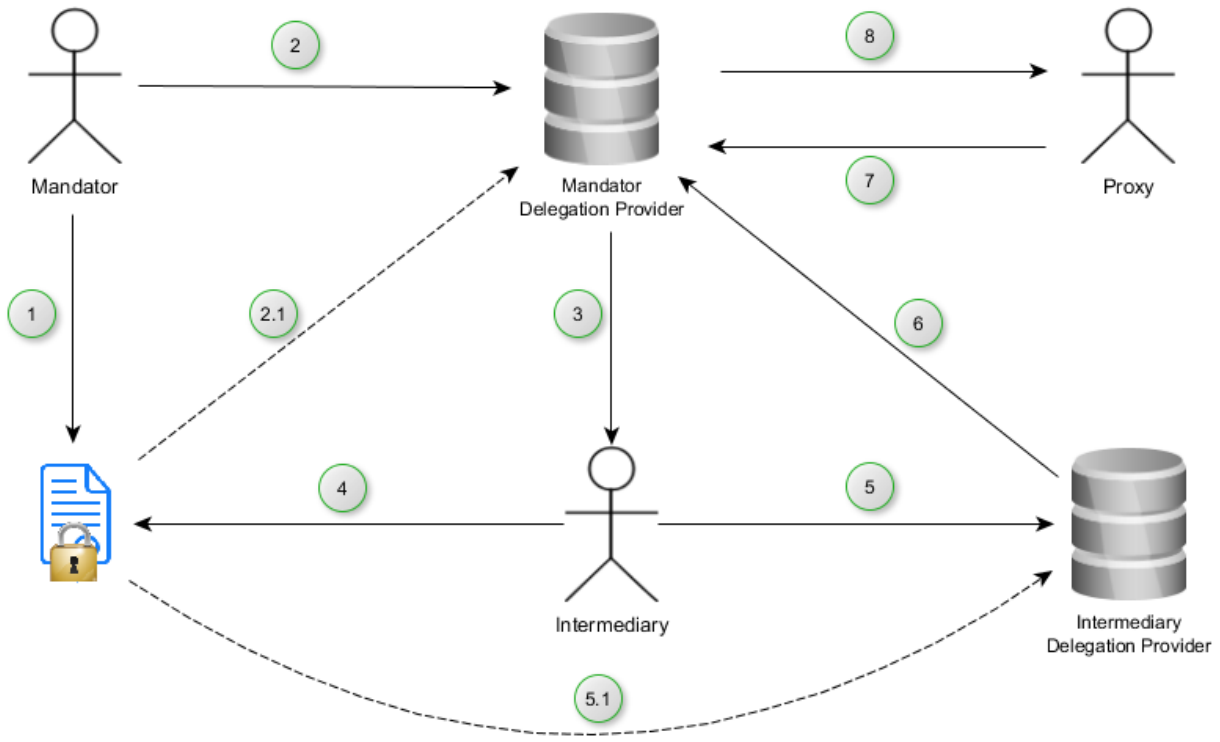

Figure 3: Creating, Publishing, and Usage of a delegation flow for the delegation type

In the first step the Mandator creates the delegation. The Mandator chooses the correct domain and the correct parameters for the delegation and fills them in. The Mandator then chooses the Intermediary and allows the delegation type to be created. The Mandator now signs the delegation and creates a secret key *k*, which the Intermediary and later the Proxy can use to decrypt the delegation data.

In the second step, the Mandator uploads the delegation and the key *k* to the Mandator's delegation provider and publishes it. The delegation provider now has the delegation data and the key *k*. In order for the delegation provider to not know the key *k* as such, the Mandator encrypts the key *k* for the Intermediary with $pk_{Intermediary}$. This way the delegation provider does not gain any knowledge of the key *k* as such.

Next, step three, the Intermediary downloads the delegation data and the corresponding key *k* from the delegation provider. As the Intermediary must now select the Proxy for the delegation, the Intermediary needs to decrypt the delegation as such. Therefore, the Intermediary first decrypts the key *k*, which is encrypted with $pk_{Intermediary}$ and can be decrypted with $sk_{Intermediary}$. With the key *k,* the Intermediary can now decrypt the delegation data.

Now the Intermediary can select the Proxy, which is step four. To select the Proxy, the Intermediary only needs to fill the proxy field inside the delegation data. A thread arises, if the delegation provider can also change other parameters of the delegation. This must be forbidden

and the Intermediary must only be able to write to the proxy field. Therefore, the Intermediary must sign the delegation again, and the Mandators signature must not be overwritten.

After the Intermediary has selected the Proxy, the Intermediary needs to republish the delegation data. Therefore, the Intermediary first encrypts the delegation data using key *k*. Now the Proxy does not have the key *k* available, so the Intermediary must encrypt the key *k* with pk_{Proxy}, the Proxy's public key. In step five, the Intermediary uploads the delegation data and encrypted key to its own delegation provider. This should only show the possibility, that an Intermediary can use its own delegation provider. If the Intermediary chooses to use the Mandator's delegation provider instead, step six is not executed. In step six, the Intermediary's delegation provider replicates the delegation data to the Mandators delegation provider.

In step seven, the Proxy searches for the delegation provided by the Mandator (and completed by the Intermediary) on the Mandator's delegation provider. The discovery of delegations as such is not part of this document, but mentioned for completeness. If a delegation is found on the delegation provider, the Proxy can download and use it.

In order to use the delegation, the Proxy must download it. The previously found delegation is encrypted. Therefore, the Proxy downloads the encrypted key *k*. With the sk_{Proxy}, which only the Proxy has, the Proxy decrypts the key *k*. With this key *k*, the Proxy can decrypt the delegation data, and embed it into a signature container.

9.2.4 Extending the Signature Container

OEBPS Container Format (OCF) [10] which was originally designed for use by eBooks but has been adopted as the basis for other containers including that used by Open Document Format - Open Office (ODF) [11] and Universal Container Format by Adobe Systems (UCF) [12].

Signature containers provide a means to link detached signatures with its associated data, i.e. the data over which the signature has been created. ASiC is currently the chosen container for use in Light[est]. ASiC provides a standardized container format for linking detached digital signatures to their associated data. It was standardized by ETSI in ETSI TS 102 918 [13]. However, some containers were proposed before ASiC and they were used in diverse domains. Some of the containers are OEBPS Container Format (OCF), Open Document Format - Open Office (ODF) and Universal Container Format (UCF). ASiC is essentially a derivative of ODF, UCF and OCF.

A valid OCF container must contain a META-INF root folder that defines the properties of the container. An example taken from the format specification document [reference] is given below.

```
META-INF/
   container.xml – Note: includes multiple <rootfile> elements
   [manifest.xml]
   [metadata.xml]
   [signatures.xml]
   [encryption.xml]
   [rights.xml]
OEBPS/
   Great Expectations.opf
   cover.html
   chapters/
      chapter01.html
      chapter02.html
      … other OPS files for the remaining chapters …
PDF/
   Great Expectations.pdf
```

Two physical container methodologies used to represent the OCF format is the Zip Container and the File System Container [10]. This format was standardized by the idpf.

The OpenDocument Format (ODF) [11] format specifies the standard format for text, spreadsheet, database front-end, presentation, formula, chart and drawing documents[ref]. It was created as a successor to the vendor-specific formats such as .doc, .wpd, .xls and .rtf. A valid ODF package must contain at least a content.xml and styles.xml file. These describe the structure and style of the document. The ODF format is used in the OpenOffice software package and is supported by Microsoft Office. The ODF format was standardized by OASIS and adopted by ISO/IEC JTC1 SC34.

Universal Container Format (UCF) [12] collects related set of files into a single container. The UCF container also has a META-INF folder contained at the root of the container. This folder contains information about the content, signature and encryption of the container. Adobe Systems standardized this format.

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 ZIP format is a good way of storing and transferring files and folders.

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 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. Two kinds of ASiC containers are defined: ASiC-S (ASiC-Simple) and ASiC-E (ASiC-Extended). The

major difference between the two containers is that ASiC-S contains only one data object while the ASiC-E contains multiple data objects. An example of ASiC-S taken from the specification document is shown in Figure 4.

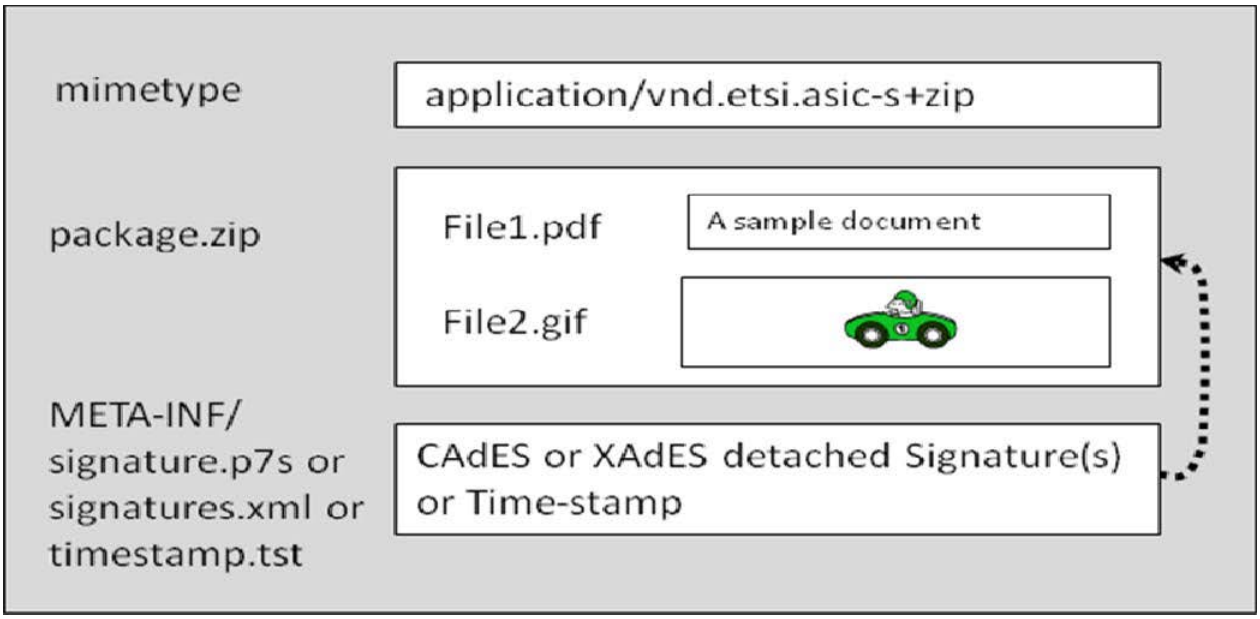

Figure 4: Example of an ASiC container

An extension to support LIGHT^{est} will be to add a new XML data structure in the META-INF folder describing the delegation information (in XML) proposed by LIGHT^{est}. The structure of a delegation in XML, taken from [14], is shown below:

```
<?xml version="1.0"?>
<delegation version="1.0">

    <!-- Mandatory Information -->
    <issuedDate> 2017-05-14T23:59:59 </issuedDate>
    <proxy> Bob </proxy>
    <issuer> Alice </issuer>
    <intermediary />
    <substitutionAllowed>false</substitutionAllowed>
    <delegationAllowed>false</delegationAllowed>
    <validity>
        <notBefore> 2017-06-15T00:00:00 </notBefore>
        <notAfter> 2017-10-06T23:59:59 </notAfter>
    </validity>

    <domain name="purchase" version="0">
       ...
    </domain>

    <ds:signature>
    </ds:signature>
```

```
</delegation>
```

This format allows the detection of the delegation, together with the required attributes, Mandator, Intermediary, and Proxy. It further provides information about the validity period of the delegation within the file. The file is also signed by the Mandator; or Intermediary; to provide prove that no information in the file has been changed.

9.3 Delegation Tools

This section describes the tools and software interfaces in use to communicate with a Delegation Provider. In section 9.3.1 we describe the management interface. The management interface is used to upload the delegation itself and the delegation en-/decryption key to the Delegation Provider. The user interface described in section 9.3.2 is then used to retrieve the delegation by the Intermediary, or Proxy respectively. Figure 5 shows how the Delegation Provider can be constructed from the management and the user interface.

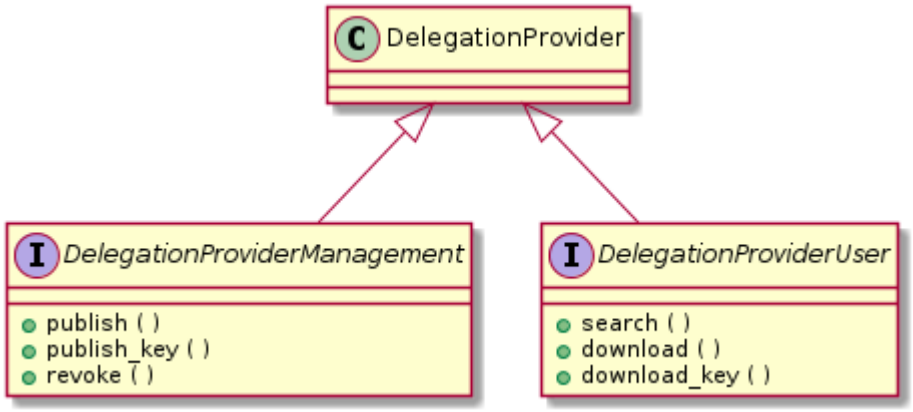

Figure 5: Delegation Provider

9.3.1 Delegation Provider Interfaces

The management interface fulfils two tasks. The first task is the publication of delegations. This task can also be described as an upload of the delegation data to the Delegation Provider, together with the encrypted secret key. This data is then queried by the Intermediary, or the Proxy via the User Interface as described in 9.3.2. The second task is the revocation of delegations. Revoking a delegation may be required during the lifetime of a delegation if for example a Proxy leaves the company in which the Proxy has a delegation from a Mandator. It is of course important to revoke the delegation immediately to prevent any damage / misuse of the delegation. This is depicted in Figure 6.

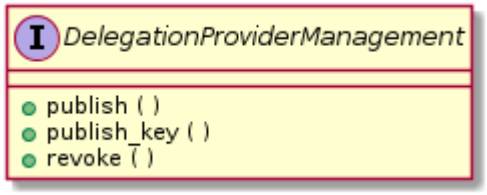

Figure 6: Management Interface for Delegation Provider

The second part of the interfaces is the user interface. Via this interface a user, in this case either Proxy or Intermediary, can search and download delegations that are for them. The first call to the interface would be a call to the search method, to find out if a delegation exists for a particular user. In the later step, this delegation and the key is downloaded via the download() and download_key() methods. As the key is encrypted with the PK_{Proxy} ($PK_{Intermediary}$ for the Intermediary), only the correct Proxy can decrypt the delegation correctly. The interface is shown in Figure 7.

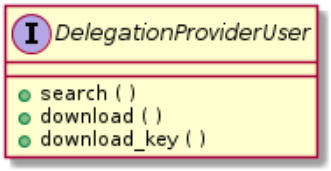

Figure 7: Interface for the User Interface for the Delegation Provider

9.3.1.1 Function: publish()

This function publishes the delegation on the Delegation Provider. It takes the signed and encrypted delegation as a parameter. The key has to be uploaded separately in order to keep the interfaces for the delegation and the key separate from each other.

9.3.1.2 Function: publish_key()

This function publishes the key for the decryption of the delegation on the Delegation Provider. The key is encrypted with the pk_{Proxy} or $pk_{Intermediary}$, so that the Delegation Provider cannot gain any knowledge and decrypt the delegation.

9.3.1.3 Function: revoke()

This function revokes the given delegation. The delegation is then transferred to a certificate revocation list, which a Verifier can query to find revoked delegations.

9.3.1.4 Function: search()

The search function searches for a given delegation on the Delegation Provider. Given several attributes, the function searches for the delegation.

9.3.1.5 Function: download()

The function downloads the given delegation. A hash value of the delegation will be passed into the function. With this hash, the function then searches in the internal database for the delegation and returns it if found.

9.3.1.6 Function: download_key()

This function downloads the published key for a delegation. The key is required to decrypt the delegation. The key itself is provided encrypted with pk_{Proxy} (or $pk_{Intermediary}$) and needs to be decrypted after download. The decrypted key is a symmetric key and can decrypt the delegation. It is crucial that the key is kept secret after the delegation has been decrypted!

9.3.2 User Interface

This section describes a possible user interface for the publication of a delegation. The actual implementation of this should be up to the pilot applications within LIGHTest. This Graphical User Interface (GUI) will only give an idea of how a GUI for the publication can look like. The GUI is found in Figure 8.

The intention of the GUI is to show a possible layout and implementation. The GUI provides all the fields that are required to create a delegation. The source code provides information for the use of LIGHTest libraries and their usage and can be used as a basis for further implementations.

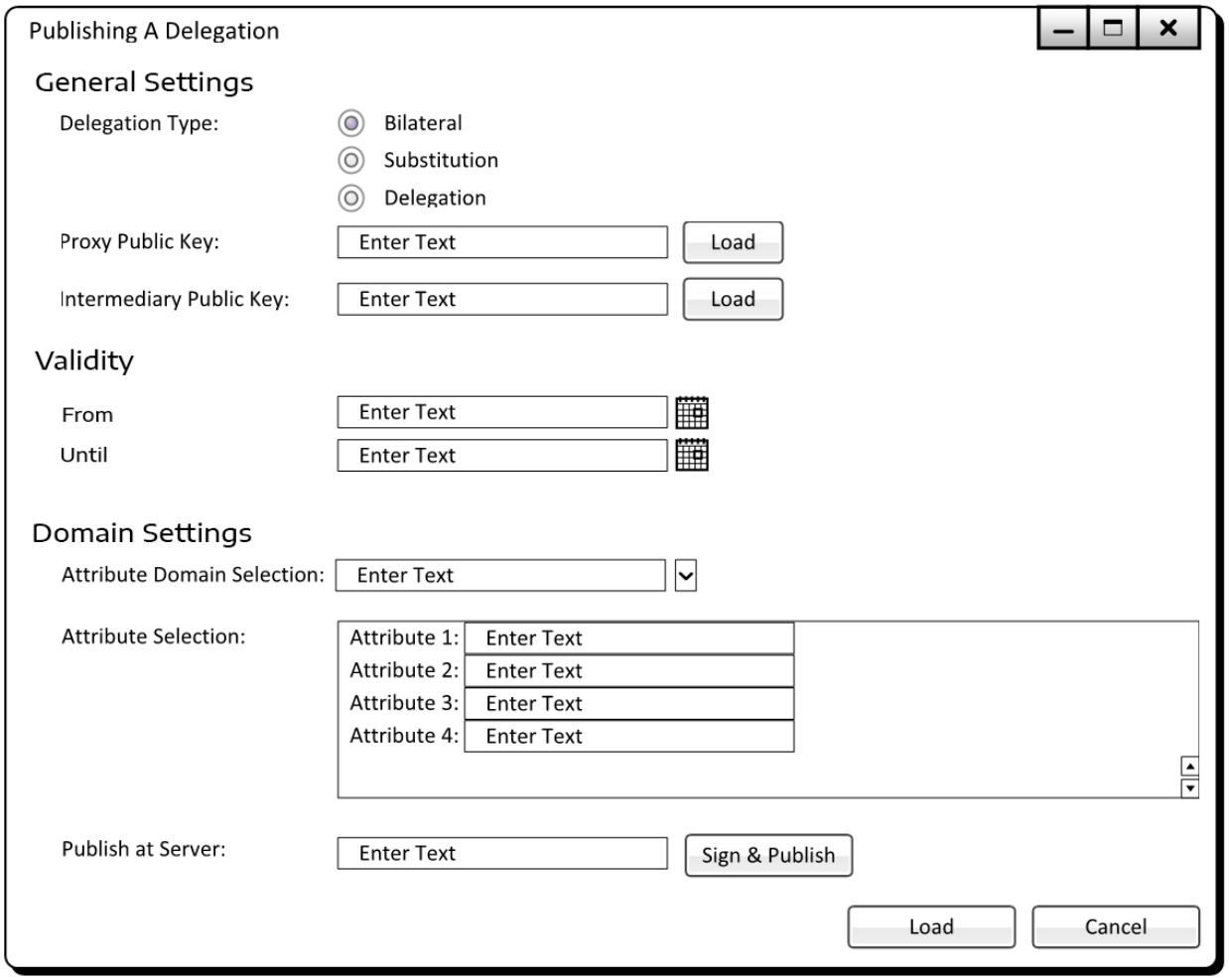

Figure 8: Prototype of the User Interface for the Publication of Delegations

As a first step, the user (in this case the Mandator) selects the delegation type that should be created. Three different types of delegations can be created. The bilateral type, where the Mandator needs to name the Proxy immediately, the substitution type, and delegation type, where the Mandator only needs to choose the Intermediary. As a second step, the Mandator would load the correct public key certificate for either Proxy or Intermediary, depending on the choice made in the first step.

A delegation should always have a validity period. This can be set in the validity section. The notBefore field defines the starting period. The delegation will not verify as valid if it is used before the specified date and time. The not after field defines the end of the delegation. After this date and time has passed, the delegation cannot be executed anymore and the verifier will not let the delegation pass. It is a good advice to set the not after date to the expiration date of the Proxies or Intermediaries public key certificate. This requires to renew the delegation by the Mandator in any way and the delegation has an expiration date as well. If the public key certificate expires the delegation cannot be executed by the Proxy anyway.

As the ATV can only verify attributes that are known by the ATV and required for a certain domain, the Mandator chooses the Attribute Domains. These are predefined lists of attributes that are loaded into the box below. This should give the ATV the chance of having the correct set of attributes for a specific task. The GUI loads the attribute fields into the area below the drop down box. Depending on the domain, only those attributes are visible.

As the last step the Mandator chooses the Delegation Provider that should publish the publication and enters the address of the Delegation Provider. As every LIGHTest user can host a Delegation Provider, we cannot provide a comprehensive list of Delegation Providers. We further think, that most users have a small list of Delegation Providers to choose from. Thus, this field can also be preloaded via a configuration file. This eases the burden of entering a Delegation Provider name. However, it does not free the user from not choosing a Delegation Provider.

If all fields are set and the delegation can be published, the Mandator presses the “Sign & Publish” button. The delegation is now signed by the Mandator. Depending on the choice made in the first step all fields, or only a selection of fields are signed (only the fields the Intermediary can choose are not signed, which is only the Proxy field). A new secret key is created to encrypt the delegation data. The delegation data is the uploaded to the Delegation Provider. Further, the Proxy or Intermediary needs to have the same key as well. Therefore, the secret key is also encrypted, but this time with the pk_{Proxy} or $pk_{Intermediary}$. After the encryption of the secret key, the key is also uploaded to the Delegation Provider.

10. Examples

This section contains examples of the delegation types described in previous sections. The examples cover the basic cases of the delegations and can be used as a reference.

10.1 Bilateral Delegation

The following listing shows an example delegation between a Mandator and a Proxy. The domain field is not filled as it is a general delegation. This means that the Proxy is allowed to do whatever is required to represent the Mandator and does not have any limits. So it would theoretically be possible for the Proxy to sell the House of the Mandator without the Mandator wanting this. Further, the delegation is only valid during a specific time and neither substitution nor delegation are allowed. Finally the delegation is signed by the Mandator.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<ns2:delegationType xmlns:ns2="http://www.w3.org/2000/09/xmldsig#">
  <information>
    <version>1.0</version>
    <sequence>0</sequence>
  </information>
  <issuedDate>2018-03-01T14:53:37.897+01:00</issuedDate>
  <proxy>VGVzdCBQcm94eTE=</proxy>
  <issuer>VGVzdCBJc3N1ZXIx</issuer>
  <intermediary>Tm9uZQ==</intermediary>
  <substitutionAllowed>false</substitutionAllowed>
  <delegationAllowed>false</delegationAllowed>
  <validity>
    <notBefore>2018-01-29T00:00:00</notBefore>
    <notAfter>2019-12-31T00:00:00</notAfter>
  </validity>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<ds:SignedInfo>
<ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-
xml-c14n-20010315"/>
<ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-
sha1"/>
<ds:Reference URI="">
<ds:Transforms>
<ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature"/>
<ds:Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315#WithComments"/>
</ds:Transforms>
<ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<ds:DigestValue>bT7S2y/xFbqOj/h6qiDXwqk0jmQ=</ds:DigestValue>
</ds:Reference>
```

```
<ds:Reference URI="http://www.w3.org/TR/xml-stylesheet">
<ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<ds:DigestValue>vNmUYNLrGa5/IhrqzcKqMltNwDc=</ds:DigestValue>
</ds:Reference>
<ds:Reference URI="http://www.nue.et-inf.uni-siegen.de/index.html">
<ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<ds:DigestValue>V/WVTjFWobmyF6qIqMnSwVodMjk=</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue>KBbjBRrfXEUCTowysVCmhtbm39oTJiRqBTxB6qFuHg/COGRHv0gaww==</ds:SignatureValue>
<ds:KeyInfo>
<ds:X509Data>
<ds:X509Certificate>
MIIC9jCCArQCBDruqiowCwYHKoZIzjgEAwUAMGExCzAJBgNVBAYTAkRFMR0wGwYDVQQKExRVbml2&#13;
ZXJzaXR5IG9mIFNpZWdlbjEQMA4GA1UECxMHRkIxMk5VRTEhMB8GA1UEAxMYQ2hyaXN0aWFuIEdl&#13;
dWVyLVBvbGxtYW5uMB4XDTAxMDUwMTEyMjA1OFoXDTA2MTAyMjEyMjA1OFowYTELMAkGA1UEBhMC&#13;
REUxHTAbBgNVBAoTFFVuaXZlcnNpdHkgb2YgU2llZ2VuMRAwDgYDVQQLEwdGQjEyTlVFMSEwHwYD&#13;
VQQDExhDaHJpc3RpYW4gR2V1ZXItUG9sbG1hbm4wggG3MIIBLAYHKoZIzjgEATCCAR8CgYEA/X9T&#13;
gR11EilS30qcLuzk5/YRt1I870QAwx4/gLZRJmlFXUAiUftZPY1Y+r/F9bow9subVWzXgTuAHTRv&#13;
8mZgt2uZUKWkn5/oBHsQIsJPu6nX/rfGG/g7V+fGqKYVDwT7g/bTxR7DAjVUE1oWkTL2dfOuK2HX&#13;
Ku/yIgMZndFIAccCFQCXYFCPFSMLzLKSuYKi64QL8Fgc9QKBgQD34aCF1ps93su8q1w2uFe5eZSv&#13;
u/o66oL5V0wLPQeCZ1FZV4661FlP5nEHEIGAtEkWcSPoTCgWE7fPCTKMyKbhPBZ6i1R8jSjgo64e&#13;
K7OmdZFuo38L+iE1YvH7YnoBJDvMpPG+qFGQiaiD3+Fa5Z8GkotmXoB7VSVkAUw7/s9JKgOBhAAC&#13;
gYASWfn+G1k/nWntj9jX7Nk5JKaiLZ9BLR16eJJxqff33THLfdGs98Xmh2oRWZVh9PMV8oTP3hpR&#13;
cRipjZUZVEIqsBlOGTVLCg4H5TJ81JWOiprh+mkhClNqUr8l5Hu7FBSvQB6inryeva7j0aKNiIvK&#13;
8vfHTiUZpnyNRhkveBlM0jALBgcqhkjOOAQDBQADLwAwLAIUPDd/UmB9GeHqvGjny30Bvjt0AkUC&#13;
FA9ab72kKuB5geYGeckbBrcgPnZk
</ds:X509Certificate>
</ds:X509Data>
<ds:KeyValue>
<ds:DSAKeyValue>
<ds:P>
/X9TgR11EilS30qcLuzk5/YRt1I870QAwx4/gLZRJmlFXUAiUftZPY1Y+r/F9bow9subVWzXgTuA&#13;
HTRv8mZgt2uZUKWkn5/oBHsQIsJPu6nX/rfGG/g7V+fGqKYVDwT7g/bTxR7DAjVUE1oWkTL2dfOu&#13;
```

```
K2HXKu/yIgMZndFIAcc=
</ds:P>
<ds:Q>l2BQjxUjC8yykrmCouuEC/BYHPU=</ds:Q>
<ds:G>
9+GghdabPd7LvKtcNrhXuXmUr7v6OuqC+VdMCz0HgmdRWVeOutRZT+ZxBxCBgLRJFnEj6Ew
oFhO3&#13;
zwkyjMim4TwWeotUfI0o4KOuHiuzpnWRbqN/C/ohNWLx+2J6ASQ7zKTxvqhRkImog9/hWuW
fBpKL&#13;
Zl6Ae1UlZAFMO/7PSSo=
</ds:G>
<ds:Y>
Eln5/htZP51p7Y/Y1+zZOSSmoi2fQS0deniScan3990xy33RrPfF5odqEVmVYfTzFfKEz94
aUXEY&#13;
qY2VGVRCKrAZThk1SwoOB+UyfNSVjoqa4fppIQpTalK/JeR7uxQUr0Aeop68nr2u49GijYi
LyvL3&#13;
x04lGaZ8jUYZL3gZTNI=
</ds:Y>
</ds:DSAKeyValue>
</ds:KeyValue>
</ds:KeyInfo>
</ds:Signature>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<ds:SignedInfo>
<ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-
xml-c14n-20010315"/>
<ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-
sha1"/>
</ds:SignedInfo>
<ds:SignatureValue/>
</ds:Signature>
</ns2:delegationType>
```

11. Conclusion

This deliverable has shown the creation and publication of delegations on a Delegation Provider. The Domain Name System; and the extensions DNSSEC and DANE; is used only to discover the IP address of the Delegation Provider. The IP address is used for the communication with the Delegation Provider. Before that, DNSSEC and DANE will ensure the correctness of the Delegation Provider. Meaning, that the Delegation Provider is the one found via the given domain and not an impostor.

Further, we defined the types of delegations that can be created and provided protocols for the publication of the delegation. In our protocol, the Mandator either knows who will represent him, or another person or entity that knows a good proxy. The Mandator creates the delegation and a secret symmetric key to encrypt the delegation data. This key is then send together with the encrypted delegation to the Delegation Provider, where it is stored. The Proxy or Intermediary can download the data from there. As the key is encrypted with the public key of the Proxy or Intermediary, they can easily decrypt the key with their secret key.

A document can be signed with a delegation. In order to sign the document with the delegation, the Proxy needs to download the delegation data from the Delegation Provider first. When the Proxy creates the signature container, which is an ASiC container, the Proxy embeds the delegation data in the container. The ATV can later find and evaluate the delegation data found in the signature container and issue queries to the Delegation Provider if the delegation is still valid.

12. References

[1] The LIGHTest Project, “D2.14 - Architecture and Technical Coordination,” *Project Deliverable,* 2017.

[2] The LIGHTest Project, “D5.1 Conceptual Framework for Delegations,” 2017.

[3] The LIGHTest Project, “D3.1 - Design of a Conceptual Framework for Trust Schemes,” *Project Deliverable,* 2017.

[4] The LIGHTest Project, D3.3 - Design of DNS-based Publication of Trust Schemes, 2018.

[5] The LIGHTest Project, D4.1 - Design of a Conceptual Framework for Trust Scheme Translation, 2017.

[6] The LIGHTest Project, D4.3 - Design of DNS-based Publication of Trust Translation Schemes, 2018.

[7] O. K. P. Falstrom, The Uniform Resource Identifier (URI) DNS Resource, RFC 7553, Internet Engineering Task Force, June 2015.

[8] P. Mockapetris, Domain names – implementation and specification, RFC 1035, Internet Engineering Task Force, November 1987.

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

[10] International Digital Publishing Forum, “Open Container Format (OCF) 2.0.1 v1.0.1,” International Digital Publishing Forum, 2010.

[11] International Organization for Standardization, “Information technology — Open Document Format for Office Applications (OpenDocument) v1.2,” International Organization for Standardization, 2015.

[12] Adobe Systems, “XMP Specification Part 3: Storage in Files,” Adobe Systems, 2016.

[13] European Telecommunications Standards Institute, “Electronic Signatures and Infrastructures (ESI): Associated Signature Containers (ASiC),” European Telecommunications Standards Institute, 2011.

[14] G. Wagner, O. Omolola and S. More, “Harmonizing Delegation Data Formats,” in *Open Identity Summit 2017*, Karlstad, Sweden, 2017.

[15] 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.

13. Project Description

LIGHTest 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 LIGHTest 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 LIGHTest 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 LIGHTest open source software to easily query this trust information to verify trust in simple signed documents or multi-faceted complex transactions.

The three-year LIGHTest 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 LIGHTest consortium consists of 14 partners from 9 European countries and is coordinated by Fraunhofer-Gesellschaft. To reach out beyond Europe, LIGHTest 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 Globalsign (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.