Sometimes it's better to be STUCK! SAML Transportation Unit for Cryptographic Keys *

Christopher Meyer, Florian Feldmann, and Jörg Schwenk

Horst Görtz Institute for IT-Security, Ruhr-University Bochum
{christopher.meyer, florian.feldmann, joerg.schwenk}@rub.de

Abstract. Over the last decade the Security Assertion Markup Language (SAML) framework evolved to a versatile standard for exchanging security statements about subjects. Most notably, SAML facilitates the *authentication* of users, and is thus deployed in both Webservice (SOAP, WS-Security) and REST-based (SAML SSO webbrowser profile, SAML Bearer token in OAuth) services. But at least SAML provides no standardized, overall solution to transport key material.

This paper recommends an extension, STUCK - the SAML Transportation Unit for Cryptographic Keys, to the SAML framework which provides an easy way to transport cryptographic key material bound to assertions issued by particular subjects. The proposal fits into existing solutions and is fully compliant with the Security Assertion Markup Language, XML Digital Signature and XML Encryption standards.

Keywords: SAML, XML, Key Transportation, Key Distribution, SAML Extension

1 Introduction

SAML In the world of Single Sign-On (SSO), and authentication of users in general, the Security Assertion Markup Language (SAML) [1] evolved to be a successful standard. Companies like Google[1] and Salesforce[2] rely on its flexibility and benefits. SAML's ability to map security statements about subjects to XML provides an easy and human readable solution for demands concerning authentication and authorization of data exchange. But exchanging subject and issuer information, as well as assertions about them, is just one possible application of SAML's extensive features.

* This work was partially funded by the *Sec*^{2} project of the German Federal Ministry of Education and Research (BMBF, FKZ: 01BY1030).

[1] http://www.google.com

[2] http://www.salesforce.com

AKE Multiple real-world applications depend on an *authenticated key exchange (AKE)*, which usually consists of a key agreement protocol combined with a corresponding authentication protocol. It is necessary to combine identity management and federation with key exchange capabilites between the participants in a secure way.

Contribution This paper describes how to perform authenticated key transport within the SAML framework. More precisely, it provides the following contributions:

- It is shown how to embed key information into SAML Assertions, in a fully standard compliant way. Thus key management can easily be integrated in any SSO/IDM system.
- A proof of concept implementation of the proposed solution is available within the *Sec*^{2} project[3] which aims at adressing the issue of user encrypted cloud storage by performing en- and decryption exclusively at client side (and by using hardware enabled key stores). For this to work it is necessary to exchange key material (the solution will be introduced in detail in section 6).

2 Related work

The idea of combining SAML and key management/distribution capabilities is not new and has already been subject of several other publications such as the *SAML V2.0 Kerberos Web Browser SSO Profile Version1.0* [2] specification. The aforementioned standard aims at a seamless integration of Kerberos into the browser world in combination with SAML usage. Thus, Kerberos already provides a complete solution for symmetric key management and distribution. Due to its great success and wide spread distribution, Kerberos can be seen as the de facto standard for key distribution in the symmetric world. Technologies like Microsoft ActiveDirectory[4] rely on the security of the Kerberos protocol. Additionally, many vendors such as e.g., Oracle[5] or SAP[6] offer Kerberos support in their products - mostly for authentication purposes. However, the main drawback of Kerberos - from this paper's point of view - is that it is limited to be used with symmetric keys only.

For use with asymmetric keys, there is an existing standard for key management, the *XML Key Management Specification (XKMS 2.0)* [3]. The main focus of XKMS is to define a protocol for distribution and registration of public keys. The goal is to provide a WebService for the management of public key material so that other WebServices can obtain public keys for encryption and

[3] `http://www.sec2.org`
[4] `http://www.microsoft.com/en-us/server-cloud/windows-server/active-directory.aspx`
[5] `http://www.oracle.com`
[6] `http://www.sap.com`

verification of digital signatures. This WebService protocol can be compared to the well known public key server functionality introduced by *PGP* [4]. Since this standard is solely based on asymmetric public keys it is also not applicable for this proposal, as this paper aims for a technology independent solution regarding both, symmetric and asymmetric key material.

Binding keys to identities is not only a major goal of this proposal, but also the *X.509* standard [5] and *PGP* [4] address this topic. Keys should be undoubtedly connected to the corresponding entities. But one has to keep in mind that these bindings are static and non flexible. In contrast to this kind of key binding, STUCK is flexible since it binds keys to Assertions (which are themselves bound to static identities, but the keys are only implicitly bound to these identities via the Assertion). As Assertions are in general only short-lived, this can be turned into an advantage. Binding keys to Assertions and not directly to certificates offers much more flexibility and introduces a new kind of abstraction layer.

Another standard to mention is *WS-Trust* [6]. Though similar ideas of this paper could also be realized by using the WS-Trust specification, this proposal is based on SAML due to its wider usage and acceptance at major companies.

3 Motivation

With emerging new capabilities of servers and clients transporting keys or key material over the internet, in a secure and reliable way, will become more and more important in the following years. For example, the proposal of the *web crypto API*[7] or the suggestions made by the *Web Cryptography Working Group*[8] will provide clients and servers with cryptographic capabilities. In these scenarios it is often mandatory to securely exchange keys between multiple parties. Therefor standardized means for secure key transportation are necessary. Regarding this SAML recommends itself, due to its flexibility and wide-spread deployment.

3.1 Advantages of the proposal

This proposal offers the option to bind key material to an `Assertion`. Key transportation, whether encrypted or unencrypted (as in case of asymmetric public keys), can now easily be done in the same communication process and same protocol as SAML. An additional step for key management or distribution can be ommitted.

Further on, a clean standardized way may ease and facillitate identity federation beyond company borders. Not only identities could be shared, but key material, too. It could even be possible to offer key establishment facilities as an additional benefit on an identity provider's side.

The practical need for key transportation, management and distribution can be seen in the previous work that has been done, as for example the already

[7] `http://html5.creation.net/webcrypto-api/`
[8] `http://www.w3.org/2012/webcrypto/`

mentioned *SAML V2.0 Kerberos Web Browser SSO Profile* [2]. But proposing a standard for only a single usage scenario is not sufficient, since it does not care about the needs of solutions not relying on this specific scenario (in this case Kerberos). What is really needed is a very flexible approach decoupled from SAML profiles and existing key establishment and distribution systems. The proposed solution in this paper is open for every possible usage scenario.

Binding keys to SAML `Assertion`s as explained in the prior sections eliminates the necessity for additional transport media encryption since the confidential parts of the key structure can be protected at application level - either by securing the key material itself using `EncryptedKey` elements, or obfuscating the whole `KeyInfo` structure by using `EncryptedAttribute`.

Also, one has to be aware that transport encryption is not equal to identity binding since the transferred data is neither bound to an identity, nor protected after the transport has been performed. Identities and keys obtained before or after transportation (e.g., through malware or careless data processing) can be used independent of corresponding identities. Channel binding approaches may solve this issue, but add an additional server side requirement: Servers need to support both, standard conformity concerning XML processing and means like SSL/TLS for transport layer encryption. It should be noted that XML Encryption and XML Signature are partly necessary for SAML to work properly, thus it remains a valid assumption that those two standards are already available at server side.

4 Technological Foundations

The following section will introduce the major technologies utilized by the proposal. Readers familiar with XML, XML Signature, XML Encryption, SAML and key management capabilities of these standards may skip this section.

XML The eXtensible Markup Language (XML) [7] represents a human readable and machine processable language for data structuring. Data can be organized in a tree-based manner and tagged with attributes. As a major benefit, XML offers the option to be automatically validated against XML Schema Definition (XSD) files to guarantee conformance with particular data structuring rules. Both XML and optional XSD files are highly flexible and adjustable to fit nearly every scenario regarding data structuring.

XML Signature For applying the concept of digital signatures to XML documents the XML-Signature Syntax and Processing Standard (XML Signature) [8] was created. By using XML Signature it is possible to sign parts of XML documents or even the whole document.

An XML Signature is introduced by adding a `<ds:Signature>` element into an XML document. In most cases this element consists of three main subelements: The `<ds:SignedInfo>` element specifies the necessary setup for signature creation and verification such as an optional canonicalization - a document

restructuring option -, the signature algorithm and the references - the signed document parts which can be referenced e.g., via ID or XPath. Within the references also the digest method and possibly transformation methods as well as the digest value can be specified. The `<ds:SignatureValue>` element contains the actual signature associated with the referenced document parts. Information about the public key, which can be used to validate the signature, can be stored in the `<ds:KeyInfo>` element.

XML Encryption With the capabilities provided by the XML Encryption Syntax and Processing Standard (XML Encryption) [9] the security goal of confidentiality can be achieved. XML Encryption includes the features of popular encryption solutions such as DES [10] or AES [11] into the XML world. Parts of XML documents can be encrypted and decrypted by using XML Encryption.

The `<enc:EncryptedData>` element introduces an encrypted part or subpart of the XML document. Obviously, this element is not added to the XML structure as a signature would be, but it replaces the encrypted cleartext. The three main components of an encrypted data block are the `<enc:EncryptionMethod>`, which specifies the cipher algorithm used for encryption and decryption, the `<ds:KeyInfo>` and the `<enc:CipherData>`. The latter contains the encrypted data itself, while `<ds:KeyInfo>` holds information about the key which has to be used for decryption of the ciphertext, this may also be an encrypted key.

Mostly, *hybrid encryption* [12] schemes are used i.e., the symmetric key is encrypted with the recipients public key. This aims at combining the speed advantages of symmetric encryption schemes with the absence of shared secrets offered by asymmetric schemes.

SAML The Security Assertion Markup Language (SAML) standard is based on XML and defines a framework for delivery of issuer and security statements. Authentication and authorization statements can be modeled around subjects. Therefore the standard defines a `<saml:Assertion>` element which can nest security statements and additional information. Moreover SAML can be bound to underlying transport media and ships with some predefined usage protocols for example a protocol implementing the popular Single Sign On use case. The SAML pre-defined protocols offer XSD files exactly defining the message structure. Message integrity and validness are achieved by using optional digital signatures via the XML Signature standard, whereas confidentiality can be achieved by using optional encryption, according to the XML Encryption standard.

5 Get STUCK - the SAML Transportation Unit for Cryptographic Keys

After having listed and explained the existing structures and technologies in the previous section, the following sections explain how these structures can be utilized to enable secure key transportation in a fully SAML 2.0 compatible way.

5.1 Goals of the contribution

The main focus while designing the STUCK solution was to provide a standardized way how to transport keys securely without breaking existing technologies. Therefore the proposal has to come with terms of XML and especially SAML compatibility, as well as major security goals when exchanging confidential key material.

SAML 2.0 compatibility The extension proposal focuses on a solution without modifications to the existing XML Schema definitions of SAML. The solution must not break existing implementations and has to be fully compatible to the SAML 2.0 specification (*Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0* [1]).

The SAML Standard provides flexible extension points within the `Assertion` element. As mentioned in chapter 4, these `Assertion`s are one of the core features of SAML and used for security statements about subjects. An `Assertion` can be digitally signed so that integrity protection can be guaranteed. The proposed extension puts the key information inside this element to utilize this integrity protection. Together with the subject information and a digital signature over the element the key information is inextricably bound to an entity identified by the information of the `Subject` element.

Due to the fact that a common extension point is used, the additional key information neither breaks the SAML Schema (XSD files), nor influences `Assertion` or signature processing. Existing implementations do not need to be adjusted. The application logic behind has simply to deal with additional key information inside of an `Assertion`.

Security goals The proposed solution addresses multiple design goals valuable when dealing with key transportation mechanisms.

- **Confidentiality** - provided by utilizing XML Encryption on the key material (`EncryptedKey` elements are used)
- **Integrity** - provided by utilizing XML Signature on the transfered `Assertion`
- **Authentication** - provided by utilizing XML Signature on the transfered `Assertion` which contains `Subject` and `Issuer` elements (the information should be equal to the one of the Issuer's certificates)

5.2 Identification of extension point

In order to combine a SAML `Assertion` with cryptographic key information, the necessary extension point has to be identified.

Within an `Assertion` there can be any amount of `AttributeStatement` elements with an `unbounded` number of `Attribute` elements as child nodes. An `Attribute` element requires the presence of an XML attribute of type `Name` identifying the content and a sequence of zero to `unbounded` `AttributeValue`

elements. An `AttributeValue` can hold content of type `anyType`, which weakens the strict schema definition and allows any well-formed XML data at this place. This is the extension point used by STUCK to integrate key information into an `Assertion`.

For clarity reasons, figure 1 provides a schematic illustration of a SAML `Assertion` containing key information.

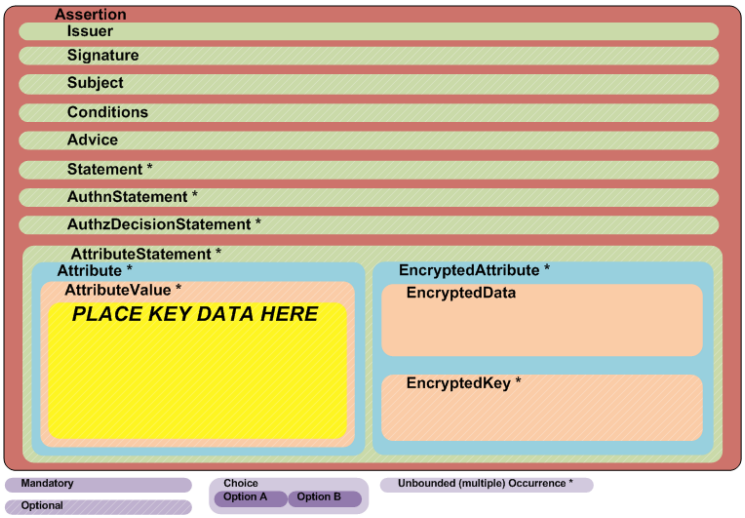

Fig. 1. Schematic illustration of a SAML `Assertion` with highlighted extension point

5.3 XML key data structure

Additional to the identified extension point for including key data into a SAML `Assertion`, a suitable XML structure for holding cryptographic keys is required. For this purpose, XML Signature already offers versatile structures for keys and certificates. Supplemented by XML Encryption and its capabilities to define encrypted keys, all necessary structures for key distribution, management and transport are present yet. No additional structures have to be defined.

In the following the existing structures are briefly discussed. We mainly focus on a single element of the XML Signature Standard, the `ds:KeyInfo` element.

ds:KeyInfo The `ds:KeyInfo` element, taken from the XML Signature Standard (here denoted as namespace `ds`), can be used to carry data somehow relevant for cryptographic keys. This includes several predefined data structures for storing information regarding e.g., key data for RSA, DSA, PGP or SPKI, as well as key related meta data like e.g., X.509 certificate data, key names, retrieval methods for externally located keys or general management information.

The following child elements from `ds:KeyInfo` are important for the STUCK proposal:

– **ds:KeyName**
  This element may contain a key identifer string which identifies key material.
– **ds:KeyValue**
  Originally defined to contain public keys used for signature verification, this element may also contain symmetric key material or any data structure defined in a namespace differnet from `ds`. The following child nodes are allowed in the schema:
  - **ds:DSAKeyValue**
    Defines how to store DSA [13] public keys.
  - **ds:RSAKeyValue**
    Defines how to store RSA [14] public keys.
  - **any ##other**
    The `ds:KeyValue` element offers the option to include additional elements from arbitrary namespaces (other than the one refered to by `ds`). This allows to extend this element by including an `enc:EncryptedKey` element from the XML Encryption Standard (see below).

enc:EncryptedKey This approach aims for a flexible solution able to carry all kinds of keys or key material, but the number of predefined key data structures in the `ds:KeyInfo` element is limited. This can be remedied by utilizing the extension point found in `ds:KeyInfo`: The element can easily be extended to allow key data usually unsuitable for these predefined data structures by adding elements from a differing namespace which provide a data structure for the desired keys. In this approach, an element from the XML Encryption Standard (here denoted by namespace `enc`), `enc:EncryptedKey`, is used.

This element offers support for transportation and storage of encrypted key material. Since it is obviously not advantageous to transport critical keys (such as private or secret keys) in an unencrypted manner, this element remains essential for a complete key distribution solution (such as Kerberos [15]).

5.4 Putting the pieces together - Extended SAML Assertion

After having identified the required XML structures and their respective extension points, the STUCK approach combines these into a single solution for secure key transportation in the SAML context.

The first step in the STUCK approach is to insert the key or key material which is to be transported into a `ds:KeyInfo` element. For this purpose, the previously identified extension point within `ds:KeyInfo` can be used to include an `enc:EncryptedKey` element which can hold any type of key or key material.

Note also that in case where a key of a predefined type for `ds:KeyInfo` should be transported (e.g., DSA or RSA keys as stated above), `enc:EncryptedKey` can be used instead of the predefined structures to utilize its inherent encryption features. Thus, the confidentiality of the key material itself is provided by XML Encryption.

The next step in the STUCK approach is to insert the `ds:KeyInfo` element including the `enc:EncryptedKey` element into a SAML `Assertion`. This is done by utilizing the previously defined extension point within a SAML `Assertion`, i.e., the `ds:KeyInfo` structure holding the key or key material is inserted into an `AttributeValue` element within the `Assertion`.

Thus, the associated key material is explicitly secured by the same means that protect the `Assertion` itself. This means integrity and authenticity of the key or key material within this extension point are implictly protected by the (optional) digital signature that protects the `Assertion`.

If further confidentiality beyond the content of an `Attribute` is necessary (as for example to obfuscate the structures behind an `Attribute`, so that not only the key material itself will be confidential, but also the accompanying additional information like e.g., key name, management data etc.) the whole `ds:KeyInfo` element can be secured by applying encryption using the `enc:EncryptedData` element from the XML Encryption Standard before embedding it into the `Assertion`.

As an alternative, the application of `EncryptedAttribute` as child of `AttributeStatement` can be used instead of `Attribute`. This approach, however, does not have any benefits over the usage of XML Encryption to secure the `ds:KeyInfo` structure and is not considered any further in this paper.

An example SAML `Assertion` including key information according to our contribution is depicted in figure 7 in the appendix. The `Assertion` is extended with an `AttributeStatement` which holds an `Attribute` with `Name="desired Key"`. This `Attribute` contains an encrypted key as `AttributeValue`. The whole content (including the key element) is protected by a `Signature` refering to `URI="#referToMe"` which targets the `Assertion` itself. In addition the `CipherData` element following the `KeyInfo` element may contain data encrypted with the transferred key (e.g., key confirmation/information data etc.)

5.5 Usage in the SAML Assertion Query and Request Protocol

The modified `Assertion` can be used with any of the predefined SAML protocols. Figure 3, gives a simplified example scenario on how a Key Requestor (KR) is able to obtain key material from a Key Server (KS) using only SAML compliant messages:

- KR sends a SAML Attribute Query to KS, authenticated with an XML signature. This request contains a reference to the requested key.
- After validating signature and request at Key Server side, KS may decide to deliver the requested key, in an encrypted form, to KR via the corresponding SAML Response, including an Assertion. For this purpose, the encrypted key is included in a SAML Attribute Statement within this Assertion to provide maximum compliance with the SAML standard. The requested key is encrypted with e.g., KR's public key preserving confidentiality.

The detailed messages (c.f., Figures 5 and 6), as well as a detailed explanation are listed in the appendix. We will come back to this scenario in more detail in

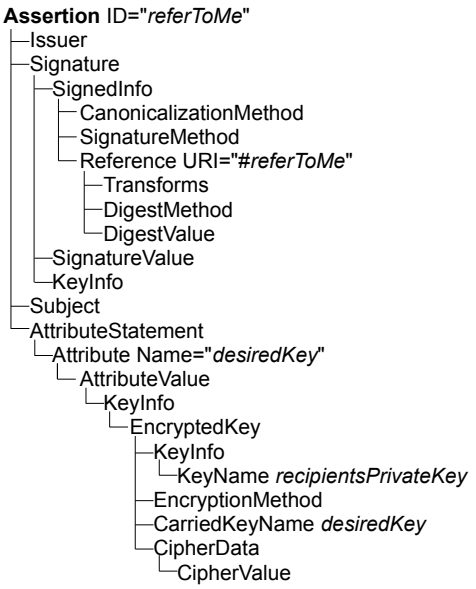

Fig. 2. Proof of concept SAML Assertion

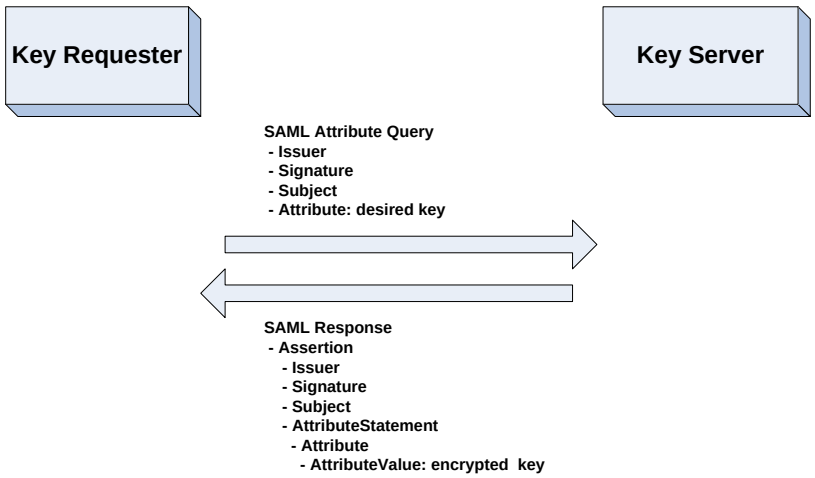

Fig. 3. Example scenario on the usage of the proposal

the Case Study (c.f., section 6) when the solution is embedded to a real world application.

6 Case study

A reference implementation of STUCK is implemented within a research project where key transport capabilities in conjunction with SAML are required.

6.1 *Sec*^{2} research project

The *Sec*^{2} research project[9] provides a hardware supported solution for secure mobile storage on public clouds. Therefore the user is able to define confidential parts of data which will then be encrypted before they are stored in the cloud. An underlying middleware handles the en-/decryption process transparently *before* the data leaves the device. For reasons of convenience the key management and distribution should be kept as automated as possible. Part of the solution is a publicly available trusted key server as depicted in figure 4.

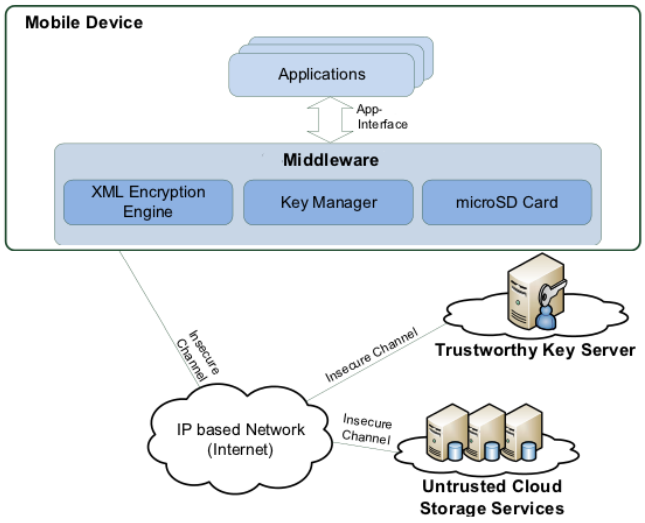

Fig. 4. *Sec*^{2} system architecture.

This key server is used for key distribution to clients. The key distribution follows the principle of *hybrid encryption* - the key server wraps (encrypts) a symmetric secret key with a mobile device's (client) individual public key. The asymmetric keypair at the client side is bound to an entity (device owner) and delivered together with a special microSD Card that has to be installed at the mobile device. The microSD Card can be considered as a Smart Card that stores key material in an unextractable way. All cryptographic operations that utilize the key material have to be performed on the card. The corresponding public

[9] `http://www.sec2.org`

key is deposited at the key server. The client is able to unwrap (decrypt) the delivered key because she is in possession of the corresponding private key.

Since the whole communication between client and key server is SAML based the proposal of this paper is applicable and used for key transportation from the key server to its clients. Additionally, another major goal is to give up transport security and render its usage optional. The (wrapped) keys should be bound to SAML `Assertion`s to provide integrity and authentication at the same time. And all that has to be in line with the SAML specification(s). So to recap, the following requirements are given:

- (encrypted) keys have to be delivered from a key server to the client
- all critical parts (such as the encrypted key or authentication information) have to be authenticated and their integrity must be ensured
- deviations from the SAML standard have to be avoided

The solution within the project combines all requirements and integrates key transport mechanisms seamlessly into SAML without Schema validations or specification of extensions. The solution uses the approach introduced in section 5. In the following an example communication procedure is outlined:

1. The middleware fetches data from a public cloud storage and determines necessary key(s) for decryption - *Key X*.
2. After a lookup the middleware is informed by the microSD Card that it is not in possession of *Key X*.
3. A SAML attribute query including authorization data and identifier of the desired key is built (c.f., figure 5) and sent to the key server.
4. The key server validates the signature and checks for the corresponding access rights of the requesting client.
5. If all preconditions are met, the client's public key together with the key identifier of the desired key is passed to a Hardware Security Module (HSM) attached to the key server - The client's public key was deposited previously at the key server during client registration. The HSM is not directly accessible by the client and can only be contacted in case of sufficient access rights - only the key server can access the HSM.
6. The HSM wraps the key identified by the key identifier with the passed public key and returns the encrypted key to the key server.
7. The wrapped key is included in a signed SAML response (c.f., figure 6) and returned to the client.
8. The client verifies the SAML response, validates the digital signature, extracts the wrapped key and passes it to the microSD Card.
9. The microSD Card unwraps the desired key by utilizing the private key and stores it unextractable after successful unwrapping.
10. The middleware is now in posession of the necessary key and can proceed as if the key had been present at the beginning.

For further information you are invited to visit the project homepage and have a look at the papers and information material. Criticism, tips and feature requests are very welcome!

7 Conclusion

The proposed solution for identity bound key material and key information offers major enhancements to the Security Assertion Markup Language. Additional means for key transport can be skipped and instead directly mapped to the SAML level. A reference implementation is integrated within the Sec^2 research project and will be soon available as open source. The proposal offers key management and distribution capabilities without schema violation, thus no adjustments to existing standards have to be made.

References

1. Cantor, S., Kemp, J., Philpott, R., Maler, E.: Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. Technical report (March 2005)
2. Hardjono, Klingenstein, Howlett, Scavo: SAML V2.0 Kerberos Web Browser SSO Profile Version 1.0. Technical report (March 2010)
3. Hallam-Baker, P., Mysore, S.H.: XML Key Management Specification (XKMS 2.0). W3C Recommendation, W3C (June 2005)
4. Garfinkel, Simson: PGP: Pretty Good Privacy. O'Reilly Media (November 1994)
5. Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., Polk, W.: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard) (May 2008)
6. Lawrence, K., Kaler, C.: WS-trust specification. Technical report (March 2007)
7. Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F.: Extensible Markup Language (XML) 1.0 (Fifth Edition). World Wide Web Consortium, Recommendation REC-xml-20081126 (November 2008)
8. Eastlake, D., Reagle, J., Solo, D.: XML-Signature Syntax and Processing. XML Signature Working Group. (2002)
9. Imamura, T., Dillaway, B., Simon, E.: XML Encryption Syntax and Processing. Technical report, W3C XML Encryption Working Group (December 2002)
10. US Department of Commerce: Data Encryption Standard (DES) (December 1993)
11. National Institute for Science, Technology (NIST): Advanced Encryption Standard (FIPS PUB 197) (November 2001)
12. Wikipedia: Hybrid cryptosystem — Wikipedia, The Free Encyclopedia (2011) [Online; accessed 12-March-2012].
13. National Institute of Standards and Technology (NIST): NIST FIPS PUB 186 – Digital Signature Standard (May 1994)
14. Rivest, R., Shamir, A., Adleman, L.: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM **21** (1978) 120–126
15. Miller, S.P., Neuman, B.C., Schiller, J.I., Saltzer, J.H.: Kerberos Authentication and Authorization System. In: In Project Athena Technical Plan. (1988)

8 Appendix

A Using STUCK within protocols

STUCK can easily be integrated into existing SAML protocols. To demonstrate the usage please have a look at the corresponding messages in figures 5, 6 while

reading.

As mentioned, a requester simply queries for a key by passing the key name as `Attribute` name (or as `AttributeValue` of a predefined `Attribute` for key queries `<saml:Attribute Name="requestKey"> <saml:AttributeValue> desiredKey</saml:AttributeValue></saml:Attribute>`). The KS will return the desired key (in an encrypted from) as `AttributeValue` carried by an `Assertion` inside a `Response`.

```
<samlp:AttributeQuery ID="myQueryID"
 Version="2.0" IssueInstant="2012-07-11T17:05:40Z"
 xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
 xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol">
  <saml:Issuer> ... </saml:Issuer>
  <ds:Signature> ... </ds:Signature>
  <saml:Subject> ... </saml:Subject>
  <saml:Attribute Name="requestKey">
    <saml:AttributeValue>desiredKey</saml:AttributeValue>
  </saml:Attribute>
</samlp:AttributeQuery>
```

Fig. 5. Proof of concept SAML AttributeQuery

```
<samlp:Response ID="myResponseID"
 Version="2.0" IssueInstant="2012-07-11T17:10:40Z"
 xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
 xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol">
  <saml:Assertion ID="referToMe"
   Version="2.0" IssueInstant="2012-07-11T17:10:40Z"
   xmlns:ds="http://.../xmldsig#">
    <saml:Issuer> ... </saml:Issuer>
    <ds:Signature> ... </ds:Signature>
    <saml:Subject> ... </saml:Subject>
    <saml:AttributeStatement>
      <saml:Attribute Name="desiredKey">
        <saml:AttributeValue> ... </saml:AttributeValue>
      </saml:Attribute>
    </saml:AttributeStatement>
  </saml:Assertion>
</samlp:Response>
```

Fig. 6. Proof of concept SAML Response

```
<saml:Assertion ID="referToMe"
 Version="2.0" IssueInstant="2012-03-01T12:59:48Z"
 xmlns:ds="http://.../xmldsig#" xmlns:enc="http://.../xmlenc#">
  <saml:Issuer> ... </saml:Issuer>
  <ds:Signature>
    <ds:SignedInfo>
      <ds:CanonicalizationMethod
       Algorithm="http://.../xml-exc-c14n#" />
      <ds:SignatureMethod
       Algorithm="http://.../xmldsig#rsa-sha1" />
      <ds:Reference URI="#referToMe">
        <ds:Transforms>  ... </ds:Transforms>
        <ds:DigestMethod Algorithm="http://.../xmldsig#sha1" />
        <ds:DigestValue> ... </ds:DigestValue>
      </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue> ... </ds:SignatureValue>
    <ds:KeyInfo> ... </ds:KeyInfo>
  </ds:Signature>
  <saml:Subject> ... </saml:Subject>
  <saml:AttributeStatement>
    <saml:Attribute Name="desiredKey">
      <saml:AttributeValue>
        <ds:KeyInfo>
          <enc:EncryptedKey>
            <ds:KeyInfo>
              <ds:KeyName>recipientsPrivateKey</ds:KeyName>
            </ds:KeyInfo>
            <enc:EncryptionMethod Algorithm=".../xmlenc#rsa-1_5" />
            <enc:CarriedKeyName>desiredKey</enc:CarriedKeyName>
            <enc:CipherData>
              <enc:CipherValue> ... </enc:CipherValue>
            </enc:CipherData>
          </enc:EncryptedKey>
        </ds:KeyInfo>
      </saml:AttributeValue>
    </saml:Attribute>
  </saml:AttributeStatement>
</saml:Assertion>
```

Fig. 7. Proof of concept SAML Assertion

```
<complexType name="EncryptedElementType">
  <sequence>
    <element ref="xenc:EncryptedData"/>
    <element ref="xenc:EncryptedKey"
     minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
</complexType>

<element name="Assertion" type="saml:AssertionType"/>
<complexType name="AssertionType">
  <sequence>
    ...
    <choice minOccurs="0" maxOccurs="unbounded">
      ...
      <element ref="saml:AttributeStatement"/>
    </choice>
  </sequence>
    ...
</complexType>

<element name="AttributeStatement" type="saml:AttributeStatementType"/>
<complexType name="AttributeStatementType">
  <complexContent>
    <extension base="saml:StatementAbstractType">
      <choice maxOccurs="unbounded">
        <element ref="saml:Attribute"/>
        ...
      </choice>
    </extension>
  </complexContent>
</complexType>

<element name="Attribute" type="saml:AttributeType"/>
<complexType name="AttributeType">
  <sequence>
    <element ref="saml:AttributeValue"
     minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="Name" type="string" use="required"/>
  ...
</complexType>

<element name="AttributeValue" type="anyType" nillable="true"/>
<element name="EncryptedAttribute" type="saml:EncryptedElementType"/>
```

Fig. 8. (Stripped) XSD of a SAML Assertion - Source: *OASIS* (`http://docs.oasis-open.org/security/saml/v2.0/saml-schema-assertion-2.0.xsd`)