How to Break XML Encryption – Automatically *

Dennis Kupser, Christian Mainka, Jörg Schwenk, Juraj Somorovsky
Horst Görtz Institute for IT Security
Ruhr University Bochum
dennis.kupser@gmx.de, christian.mainka@rub.de
joerg.schwenk@rub.de, juraj.somorovsky@rub.de

Abstract

In the recent years, XML Encryption became a target of several new attacks [19, 18, 17]. These attacks belong to the family of adaptive chosen-ciphertext attacks, and allow an adversary to decrypt symmetric and asymmetric XML ciphertexts, without knowing the secret keys. In order to protect XML Encryption implementations, the World Wide Web Consortium (W3C) published an updated version of the standard.

Unfortunately, most of the current XML Encryption implementations do not support the newest XML Encryption specification and offer different XML Security configurations to protect confidentiality of the exchanged messages. Resulting from the attack complexity, evaluation of the security configuration correctness becomes tedious and error prone. Validation of the applied countermeasures can typically be made with numerous XML messages provoking incorrect behavior by decrypting XML content. Up to now, this validation was only manually possible.

In this paper, we systematically analyze the chosen-ciphertext attacks on XML Encryption and design an algorithm to perform a vulnerability scan on arbitrary encrypted XML messages. The algorithm can automatically detect a vulnerability and exploit it to retrieve the plaintext of a message protected by XML Encryption. To assess practicability of our approach, we implemented an open source attack plugin for Web Service attacking tool called WS-Attacker. With the plugin, we discovered new security problems in four out of five analyzed Web Service implementations, including IBM Datapower or Apache CXF.

*This is the full version of the paper with same title that was published at the 9th USENIX Workshop on Offensive Technologies (WOOT'15): https://www.usenix.org/conference/woot15/workshop-program/presentation/kupser

1 Introduction

The W3C standard XML Encryption ensures confidentiality of XML data, directly on the message level. It is used in security-critical scenarios like business and governmental applications, banking systems or healthcare services. Given the importance of the scenarios XML Encryption is deployed, its security becomes a crucial point.

XML Encryption is mainly used with two encryption algorithms: AES-CBC and RSA-PKCS#1 v1.5.[1] These two standards recently became targets of attacks in many practical scenarios ranging from IPSec [8, 9] and TLS [2] to web applications and Captchas [29]. In 2011, it was shown that the XML Encryption standard is also vulnerable to attacks affecting confidentiality of symmetric ciphertexts [19]. One year later, further attacks affecting public key encryption in XML Encryption were described [18]. The attacks belong to the family of adaptive chosen-ciphertext attacks. They are applicable when the attacker is able to modify an inspected ciphertext (i.e., the ciphertext is not authenticated), send it to the server for processing, and observe the server's response. Based on this response, the attacker can decide whether the decrypted request was *valid* or *invalid*. To distinguish valid from invalid requests, he can use side channels, for example, by observing response error message or measuring response times.

In order to protect the servers against these attacks the newest XML Encryption specification proposes to use encryption schemes that are not vulnerable to adaptive chosen-ciphertext attacks: AES-GCM and RSA-OAEP. However, these schemes are not widely deployed in today's XML Security frameworks and different measures have to be applied to vulnerable servers.

Typically, XML Encryption is deployed together with

[1] In addition, the PKCS#1 standard contains version 2.1, also called RSA-OAEP. In our paper, with PKCS#1 we refer to version 1.5, unless defined otherwise.

XML Signatures, which can be used to protect data integrity and authenticity. Nevertheless, in many cases, the XML Signature protection can be circumvented using XML Signature Wrapping and XML Encryption Wrapping techniques [33]. The idea behind these techniques is very simple: the attacker moves the signed or encrypted data to a different document part so that the encrypted data becomes unprotected. However, the complexity of the XML structure and XML processing makes it difficult to prevent from these attacks, which is underlined by a large body of research [32, 25, 33, 31, 24, 27]. This allows the attacker to force the server to decrypt unprotected elements, and thus practically execute the chosen-ciphertext attacks.

Contribution. In this paper, we first summarize possible countermeasures against the attacks on XML Encryption. We present problems connected with various configurations XML Encryption is deployed with, and how to circumvent these countermeasures. We present a systematic methodology on verifying interfaces using XML Encryption. Based on this methodology, we implement an *automatic plugin* for the WS-Attacker Web Service penetration testing framework [26] that allows one to automatically analyze Web Services interfaces and execute attacks on XML Encryption.

We use our new plugin to analyze different Web Services frameworks and their application of XML Encryption. One could think that widely used Web service frameworks and commercially used XML Security Gateways are aware of the threat to XML Encryption. However, our evaluation shows that it is possible to attack frameworks like *Apache CXF*,[2] *IBM Datapower*[3] (if not configured correctly) and *Axway Gateway*[4]. All these frameworks implemented several methods to protect Web Services from the attacks. The protection mechanisms by Apache CXF could be successfully circumvented using XML Encryption and XML Signature Wrapping techniques. Axway Gateway and IBM Datapower offer several security configurations. However, only a few of them could be successfully applied to prevent the attacks.

Our paper once again shows that usage of insecure cryptographic algorithms (AES-CBC, RSA-PKCS#1) in complex scenarios can lead to sustainable and severe consequences (e.g., backward compatibility attacks [17]), which can be used to expose confidential data even if specific countermeasures are applied. We thus encourage protocol and standard designers to use *provably secure cryptography* and prevent future specification vulnerabilities.

Even though our library is currently embedded in the WS-Attacker framework, the implemented algorithms are of general importance and can be used to analyze further XML Security standards (e.g., SAML) as well.[5]

Responsible Disclosure. We communicated our findings to the Web Services developers. Vulnerabilities in Apache CXF are summarized under CVE-2015-0226 and CVE-2015-0227. Security best practices resulting from our discussions with IBM Datapower developers are addressed in their Flash alert [1]. Problems reported to the Axway security team are still under investigation.

2 Foundations

This section describes the foundations needed for automatically attacking XML Encryption in SOAP-based Web Services.

In the following, we assume the reader is familiar with basic concepts behind symmetric and asymmetric cryptography. Details behind the concrete cryptographic algorithms (RSA-PKCS#1 [23], AES-CBC [10], AES-GCM [11]) are not needed to understand this paper. We stress again that with RSA-PKCS#1, we refer to version 1.5, unless defined otherwise.

2.1 SOAP-based Web Services

The SOAP standard describes the message exchange with a Web Service [15]. A basic SOAP message consists of an `Envelope` element with two child elements named `Header` and `Body`. The SOAP `Header` element can contain meta information, for example, timestamps, signatures or encryption details. The SOAP `Body` element stores the payload that is processed by the Web Service operation.

Listing 1 depicts a SOAP message example.

```
<soapenv:Envelope>
  <soapenv:Header/>
  <soapenv:Body>
    <addUser>
      <name>Bob</name>
    </addUser>
  </soapenv:Body>
</soapenv:Envelope>
```

Listing 1: Exemplary SOAP message invoking an `addUser` operation on the Web Service.

[2] `http://cxf.apache.org`

[3] `http://www-03.ibm.com/software/products/en/datapower-gateway`

[4] `http://www.axway.com`

[5] During the development of our plugin, we strictly separated the code that is generally applicable to XML attacks, and the code that is only used to apply the attacks on Web Services.

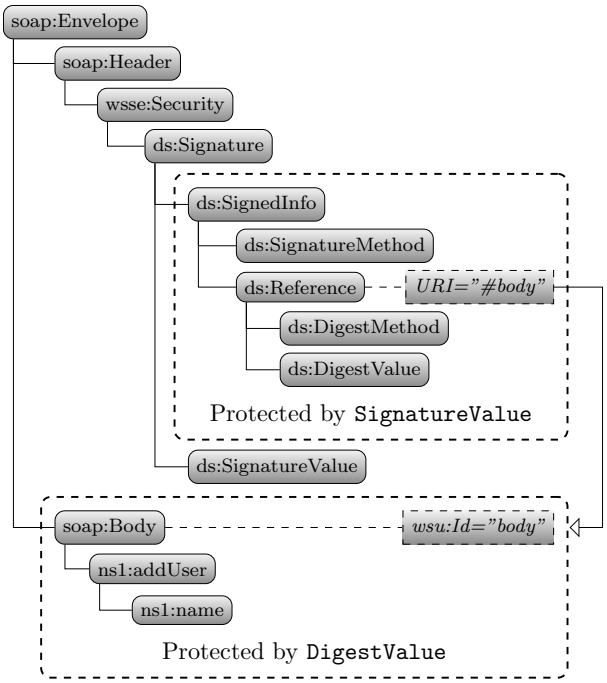

Figure 1: Simplified signed SOAP message example.

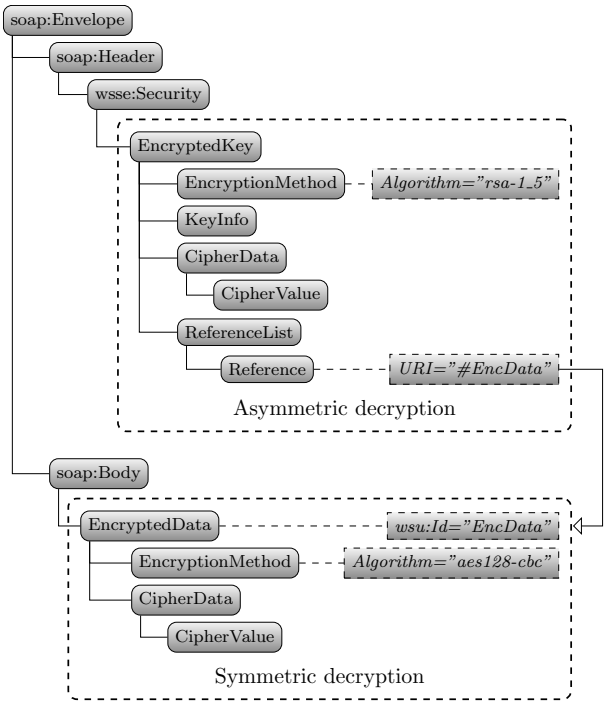

Figure 2: Simplified encrypted SOAP message example.

2.2 XML Signature

XML Signature is a W3C recommendation that defines a syntax for using digital signatures in XML messages [16]. It is used for ensuring integrity and authenticity of XML message fragments or even the whole XML messages.

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.

For embedding an XML Signature into a SOAP message, the `Signature` element is placed as a child of a WS Security header as shown in Figure 1.

2.3 XML Encryption

XML Encryption is a W3C recommendation that defines structures for ensuring confidentiality on the XML message level. Similarly to XML Signature, it is possible to encrypt whole XML documents or only 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 2 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 encrypted using the session key k.

A SOAP-based Web Service processes such an XML document as follows. It locates the `EncryptionMethod` and `KeyInfo` elements within the `EncryptedKey` element to retrieve the used algorithm and asymmetric decryption key. The server then decrypts the content of the `CipherValue` element using RSA-PKCS#1 [23]. 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. It 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 put back into the XML document tree. The server can then process the plain SOAP message and respond to the client.

If an error occurs during one of the decryption steps or during the parsing process, the server typically responds with an error message to the client.

2.4 WS-Attacker

WS-Attacker is a modular framework for Web Services penetration testing [26]. It is free, open source, and available on GitHub.[6] WS-Attacker uses a plugin architecture to execute XML-specific attacks on Web Services automatically. In its current version, WS-Attacker supports the following attacks: (1.) SOAPAction Spoofing [26], (2.) WS-Addressing Spoofing [26], (3.) XML Denial-of-Service Attacks [13], (4.) and XML Signature Wrapping [5].

3 Attacks on XML Encryption

The analyzed attacks on XML Encryption belong to the family of adaptive chosen-ciphertext attacks. In the following, we give a brief description of an adaptive chosen-ciphertext attack scenario and present basic ideas behind these attacks. Afterwards, possible countermeasures and their problems are summarized.

3.1 Adaptive Chosen-Ciphertext Attacks

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. To this end, he iteratively issues new ciphertexts $C', C'', \ldots$ that are somehow related to the original ciphertext C. He sends the ciphertexts to a receiver, and observes its responses. The receiver responses leak specific information about the validity of the decrypted message. With each response the attacker learns some plaintext information. He repeats these steps until he decrypts C. See Figure 3 for the description of this scenario.

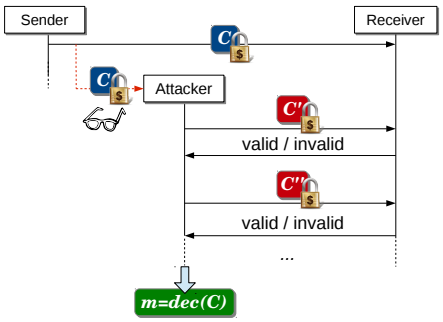

Figure 3: Adaptive chosen-ciphertext attack scenario: the attacker uses the receiver as an oracle which responds whether the message was *valid* or *invalid*.

Two major examples of these attacks are Vaudenay's attack on CBC-based symmetric encryption [35] and Bleichenbacher's attack on RSA-PKCS#1-based public-key encryption [23, 4]. Cryptographic details behind these attacks are not relevant to our paper. It is just necessary to know that the attacks against these cryptographic algorithms are applicable whenever an oracle is given that 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. Thus, the 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 [19]: The attack on symmetric CBC-ciphertexts generalizes the idea behind Vaudenay's padding oracle attacks [35]. The attacker exploits the behavior of XML servers, which need to parse XML messages after they are decrypted. In case the message cannot be parsed, the server responds with a failure, which gives the attacker a hint on message validity. This enables 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 [18]: The attack on asymmetric ciphertexts completely breaks confidentiality of the exchanged symmetric keys encrypted with the RSA-PKCS#1 [23] padding scheme. The gained symmetric key enables the attacker to decrypt the symmetric ciphertext in the XML message. The attacker can determine validity of the modified RSA-PKCS#1 ciphertext by an invalid server response, which is triggered when, for example, the RSA-PKCS#1 ciphertext decrypts to a symmetric key of an invalid length.

3.2 XML Signature as a Countermeasure

The attacks on XML Encryption are only applicable if:

(1.) The server supports RSA-PKCS#1 or Cipher Block Chaining (CBC) mode of operation.

(2.) The attacker can force the server to process modified ciphertexts and receive responses based on the message validity.

The first aspect can be solved by deploying ciphers secure against adaptive chosen-ciphertext attacks. XML Encryption supports RSA-OAEP and AES-GCM [12]. However, these two ciphers are not well-integrated in common Web Service frameworks.[7] This forces the developers to use RSA-PKCS#1 and CBC [17].

[6] https://github.com/RUB-NDS/WS-Attacker

[7] For example, only one out of five frameworks analyzed in Section 5 implements AES-GCM: Apache CXF.

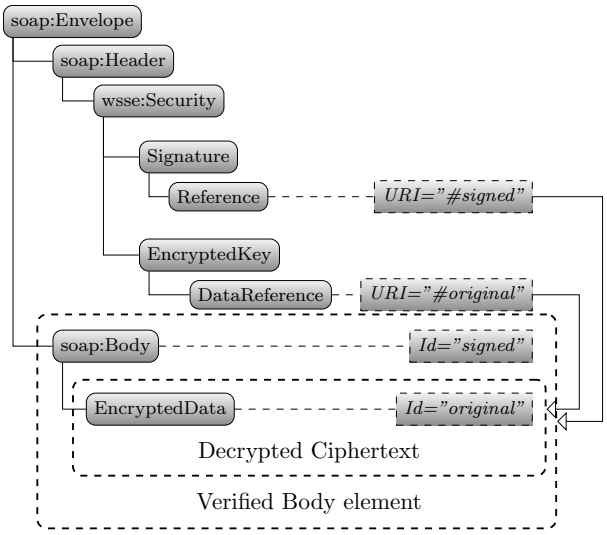

Figure 4: Encrypted SOAP message protected by an XML Signature.

The second point can be solved by protecting authenticity of the exchanged ciphertexts with XML Signatures. However, this countermeasure brings several problems [33, 30], which are briefly discussed in the following. For this purpose, please consider Figure 4, which depicts an encrypted and signed SOAP message.

3.2.1 XML Signature Wrapping (XSW)

The XML Signature Wrapping attack was first presented in 2005 [27]. The basic idea behind this attack is to move signed elements in a different part of the XML tree and force the processing logic to evaluate newly defined elements.

An XML Signature Wrapping attack example applied on the message shown in Figure 4 is depicted in Figure 5. In this message, the attacker first moves the original `Body` element to the SOAP `Header`. Afterwards, he defines a new `Body` element, and forces the `EncryptedKey` `DataReference` to point to the `EncryptedData` element within the newly defined SOAP `Body`. 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 decrypts the newly defined `EncryptedData` element with `Id="oracle"`, since this element is referenced in `EncryptedKey`.

This allows the attacker to insert arbitrary content into the `EncryptedData` element and execute the attack

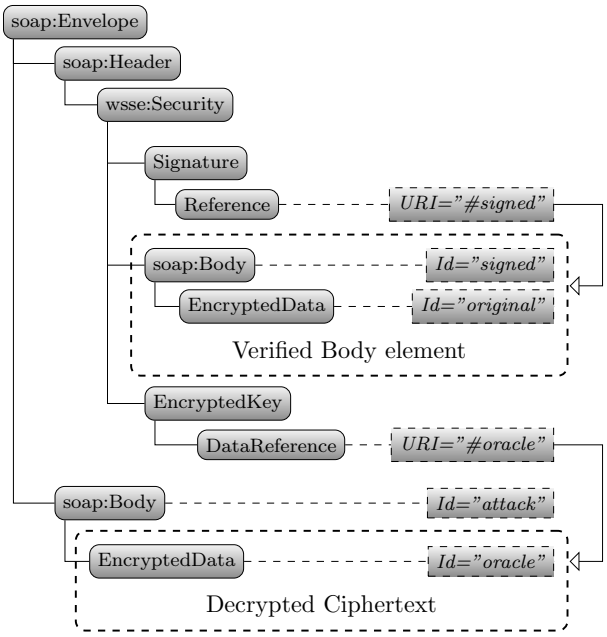

Figure 5: XML Signature Wrapping attack applied on an encrypted and signed message shown in Figure 4.

on symmetric cipher. Note that applying the XSW attack technique requires to find a valid position to move the originally signed element [31, 26]. Therefore, the attacker has to send several messages until the message is accepted.

3.2.2 XML Encryption Wrapping (XEW)

The XML Encryption Wrapping attack follows a similar principle as XML Signature Wrapping [33, 30] and enforces the decryption logic to decrypt unauthenticated XML contents. The attacker achieves this by defining new `EncryptedData` in the SOAP Header element, see Figure 6.

As can be seen in the figure, the attacker does not move the original SOAP Body element with its content. This enables the Web Service to verify and decrypt the original SOAP Body. However, the Web Service additionally decrypts also a newly defined `EncryptedData` element with `Id="oracle"`, since the `EncryptedKey` element contains a `DataReference` with `URI="#oracle"`.

There are few variations of this attack. It is for example also possible to define a completely new `EncryptedKey` element with a `DataReference` `URI="#oracle"`. This is applicable to servers processing only one `EncryptedData` for each `EncryptedKey` element.

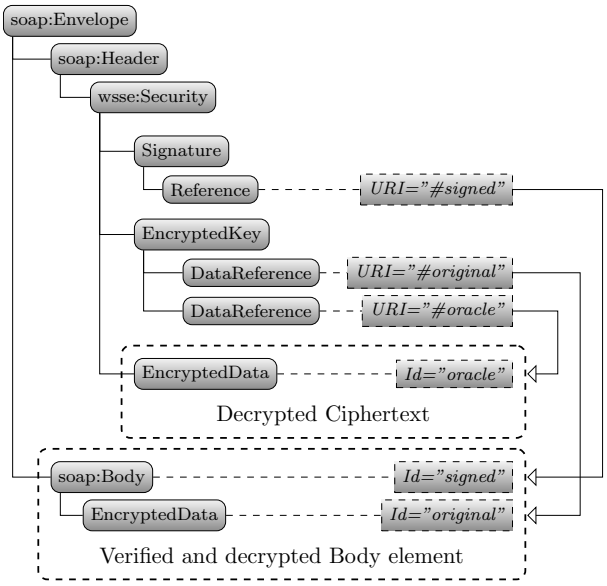

Figure 6: XML Encryption Wrapping attack applied on a signed and encrypted message forces the recipient to process unverified `EncryptedData`.

3.2.3 Protecting `EncryptedKey` Element

`EncryptedKey` element is typically not protected by XML Signatures in Web Services scenarios, as shown in Figure 4. However, by modifying the `EncryptedKey` content the symmetric key changes, which leads to a failure in symmetric data decryption. If the server responds with unified error messages, the attacker is not able to distinguish whether an error results from invalid `EncryptedKey` or invalid `EncryptedData` decryption.

Jager et al. have shown several ways to distinguish the source of decryption failure [18]. One of them is to provoke direct messages by defining a new `EncryptedKey` element without any `DataReference`. This results in decryption of a symmetric key, however this symmetric key is *not* used further for symmetric data decryption. Thus, the server responds with a failure if and only if the `EncryptedKey` is invalid. This allows an attacker to distinguish valid from invalid asymmetric ciphertexts.

A valid countermeasure against the attacks on PKCS#1 ciphertexts is to generate a random symmetric key every time the decryption fails, and use this key for further processing steps [4]. This prevents from distinguishing valid from invalid PKCS#1 ciphertexts in protocols like TLS. However, Jager et al. have shown that this countermeasure does not apply to XML Encryption [18]: the attacker can use validity of CBC ciphertexts as a side channel to distinguish valid from invalid PKCS#1 ciphertexts. This attack results in several millions of server queries and becomes impractical. See [18] for more details.

4 Automatic XML Encryption Attack

We have implemented the described attacks on XML Encryption as a plugin for WS-Attacker. This section gives a high-level overview on our implementation and highlights some noteworthy facts and problems we faced during our design and implementation phases.

4.1 About Attack Complexity

Before we describe how to break XML Encryption automatically, we need to spot on the complexity of the attack and its prerequisites. The root of the complexity is founded in different XML Security components, for example, timestamps, signed, as well as encrypted elements. To be more precise, an XML document can contain XML Signatures that do not protect encrypted elements but are used to prevent replay attacks. If the *to be decrypted XML document* contains a *nonce* or a *timestamp* that is signed, XSW must be applied to this document part. There can also be XML Signatures that protect encrypted elements as shown in Figure 4. To be able to run the XML Encryption attack, XSW or XEW must be applied on this document part.

Regarding Figure 5, we already presented *one* possible XSW vector. This is however only *one* vector. XSW is a very complex attack on its own and there can exist a large number of possible vector adaptations. Each of these vectors has to be sent to the Web Service in order to find a working solution, which enlarges the attack complexity by the number of possible XSW vectors. We refer to [32, 5] for more details.

Let us consider a typical scenario where a SOAP message includes an encrypted SOAP Body. The message contains one `EncryptedKey` and one `EncryptedData` element. The `EncryptedData` element is protected by an XML Signature, together with a `Timestamp` element.[8] We assume that the XML Signature uses ID-based referencing mechanism, which was described in Section 2.2.[9] This assumption allows us to implicate that the XSW and XEW attacks have always the same number of attack vectors (=n). This is because both attacks in general use the same wrapping positions.

If we want to attack the `EncryptedData` element in this scenario, we first need to circumvent the XML Signature that protects the `Timestamp`. We assume n pos-

[8] It is also possible that the message contains more encrypted elements. For simplicity, we omit this in our analysis.

[9] In addition, XML Signature specification allows one to use a more complex XPath-based referencing, which is omitted in our analysis, but is implemented in our plugin.

sible XSW vectors for this. We then need to circumvent the XML Signature that protects the `EncryptedData` element, which results in further n possibilities. If the second XSW fails, we can try to use the XML Encryption Wrapping on the `EncryptedData` element. We can again assume n possibilities for this. In total, we have to send $3 \cdot n$ messages to a Web Service for detecting whether we can construct an XML decryption oracle from a Web Service and execute the XML Encryption attack. The concrete number of n scales with the document's total element number – typical values are $250 - 5,000$ [5], thus we have to send up to $15,000$ messages.

If the XML Signature countermeasures could be successfully circumvented, we have to send differently formatted ciphertexts to the server. We then have to map the real server responses to responses produced by an oracle (valid and invalid). Once the mapping is provided, the attack can be executed. The complexity of the XML Encryption attacks was analyzed in [19, 18]. The number of attack queries depends on the encryption scheme the attack is targeting. The attack on symmetric encryption scheme (AES-CBC) takes about 14 server queries per decrypted plaintext byte. The attack on RSA-PKCS#1 ciphertexts allows the attacker to directly decrypt the symmetric key, and thus is independent of the plaintext length. However, it needs to issue from 20,000 to several millions of server queries, depending on the given side channel (see [18] for more details).

4.2 Attack Workflow

Figure 7 depicts the whole attack workflow. It is structured into three phases: (1.) *detection phase*, (2.) *avoid phase*, (3.) *attack phase*.

Detection Phase. The *encrypted XML document* is the input for the whole process. In the detection phase, the document is analyzed *offline* and security elements are identified. This includes the identification of signatures, encrypted document parts, as well as timestamps. The results are stored in the *knowledge pool*, so that other components can access them.

Avoid Phase. The avoid phase is *online*. Its goal is to avoid the protection of the input document so that it is possible to: (1.) send several messages to the Web Service (circumvent replay protection) and (2.) manipulate the encrypted part that is going to be decrypted (circumvent its authenticity).

To fulfill these goals, the knowledge pool is first asked whether the document contains a signed timestamp. In this case, XSW is performed. More precisely, different XSW vectors are created in order to update the timestamp and sent to the Web Service. If no XSW is possible, the attack is aborted.

In the following step, the knowledge pool is asked whether the document contains an encrypted element that is protected by a signature. If the encrypted element is protected, further XSW and XEW steps follow. If either the XSW or the XEW step is successful, the attacker is able to modify the encrypted document part, and execute an *identify oracle step*. Otherwise, the attack is aborted and cannot be applied.

Finally, the last step in this phase identifies the oracle to perform the attack. Depending on the attacked XML part (asymmetric or symmetric), XML messages are prepared in order to provoke an error behavior in the Web Service processing (e.g., invalid RSA-PKCS#1 padding or unparsable XML character). The generated messages are then sent to the Web Service. At the end, the attacker needs to provide a mapping between the response and the oracle answer 1 and 0. This mapping is saved in the knowledge pool.

Attack Phase. In the attack phase, the Web Service is used as an oracle to execute an attack on symmetric [19] or asymmetric [18] encryption scheme. During the attack execution, adapted XML ciphertexts are created and sent to the Web Service. The received responses are evaluated using the configured knowledge pool and transformed to a 1 or 0 oracle response.

4.3 Integration into WS-Attacker

According to the fully automatic approach of WS-Attacker to penetrate Web Services, we developed a WS-Attacker plugin for XML Encryption attacks. Our plugin is open source as well and is distributed with WS-Attacker on GitHub.

As shown in Figure 8, the new plugin can be configured with different attack parameters for attacking XML Encryption. After the *detection phase* we automatically get an overview of the encrypted elements, their relations and countermeasures. The collected information has effect on the further configuration of the *avoid phase* and *attack phase*. The first step is to choose an encrypted element from the list *Elements*. Then we have to choose which element we would like to attack (*isAttackPayload*). In our example, this could be the `EncryptedData` element or the `EncryptedKey` element. Furthermore, we have the possibility to fine-tune the configuration in order to reduce the complexity of the attack, and thus reduce the total number of messages sent to the Web Service:

- *Oracle Type*: This setting allows one to define a spe-

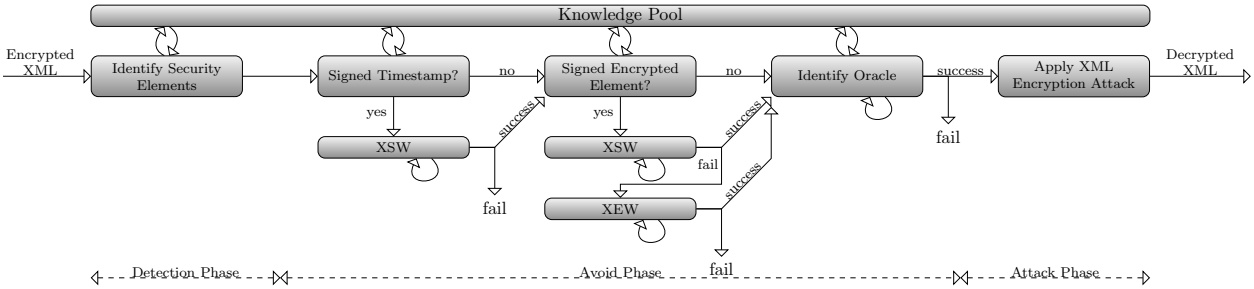

Figure 7: The attack workflow consists of three phases: *Detection* phase analyses the encrypted XML message, *Avoid* phase circumvents XML Signature protection, and *Attack* phase executes the attack.

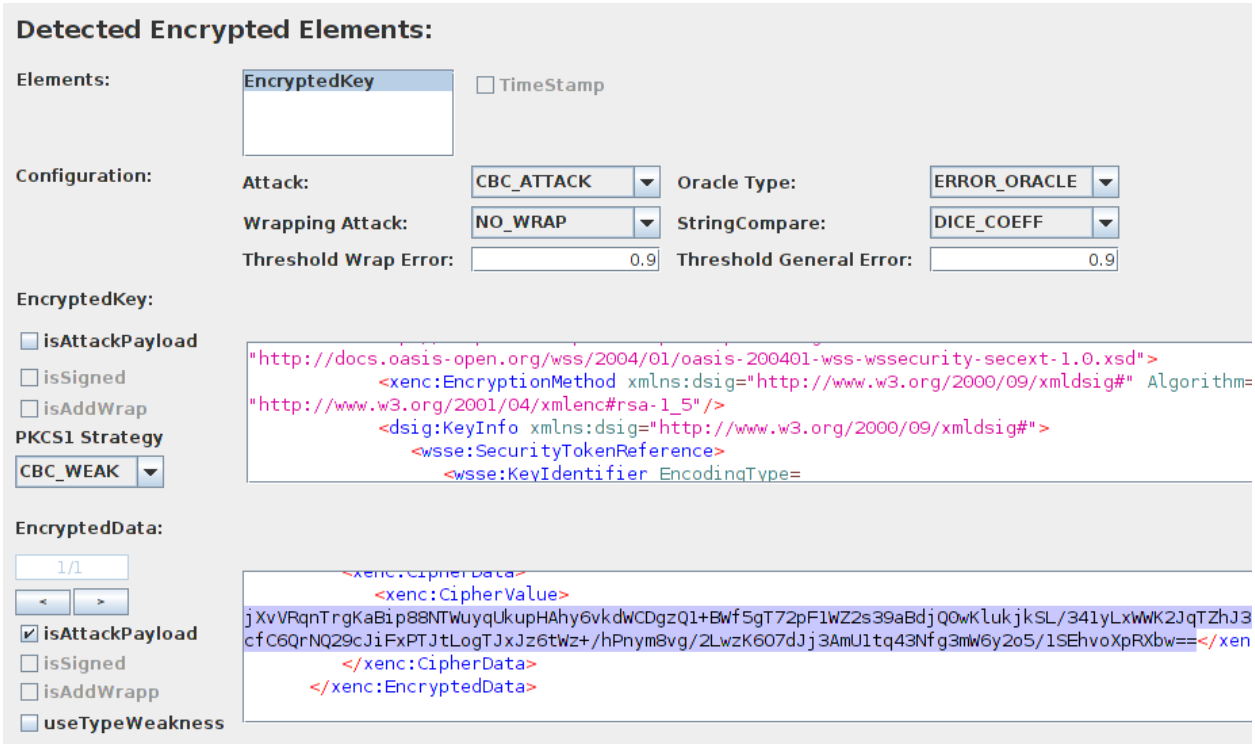

Figure 8: Our WS-Attacker XML Encryption plugin automatically detects encrypted elements and offers a user to configure oracle and attack properties.

cific oracle type, for example an error message or a timing oracle, or to test all known oracle types (which increase the duration of the attack workflow). Currently, timing oracle is not yet implemented.

- *Wrapping Attack*: Setting to use only XSW or XEW attack in order to prefer one specific type. Otherwise, both wrapping attack types are used.
- *StringCompare and Threshold Errors*: Different server responses can be mapped to the same oracle response. This is because real server responses can include message specific data like nonces or timestamps. In order to omit such comparison problems, the algorithm uses different string comparison methods (e.g., Levensthein or Dice coefficient) [34]. During the attack execution, the comparison methods are used to compare the actual server response with the ones saved in the knowledge pool to get the 1/0 oracle mapping. In addition, the setting allows one to choose from the implemented similarity metrics and define specific similarity thresholds. A similarity threshold defines a similarity value that must be achieved to map a server response to a 1/0 oracle response from the knowledge pool.

- *PKCS#1 Strategy*: As discussed in Section 3.2.3, there are different strategies to provoke error messages while applying an attack on `EncryptedKey`. One of them is a *NoKeyRef* strategy. This strategy defines a new `EncryptedKey` element that is not used further by any `EncryptedData`. Furthermore, the setting allows one to choose a *CbcWeak* strategy, which exploits a combination of weaknesses in RSA-PKCS#1 and AES-CBC, more details can be found in [18].

5 Practical Evaluation

We used our implemented WS-Attacker plugin for automatically attacking different XML Encryption implementations. We first analyzed default server configurations with XML Signature and XML Encryption. After we found a successful attack, we further investigated the server behavior and the possibilities for extended countermeasures. The summary of our results is reported in Table 1 and provides information on the number of server queries and the applied attack type (XSW / XEW / NoKeyRef / Direct). Direct attack type indicates that there is no attack strategy needed and the attack works directly (without XSW / XEW / NoKeyRef).

Please note that attacks on PKCS#1 ciphertexts are always applicable when the attacks on CBC ciphertexts are possible, as discussed in Section 3.2.3. However, the attacks become impractical, since the attacker needs to issue several millions of server queries. Thus, we do not consider them in our practical evaluation.

5.1 Apache Axis2

Web Service security standards in Apache Axis2 are provided by the Apache Rampart library. For testing purposes, we used the delivered Apache Rampart samples 5 and 6, which apply a configuration with XML Encryption and XML Signature.

5.1.1 PKCS#1 Attack

The attack on PKCS#1 ciphertexts was applicable only to an older Apache Axis2 1.6.0 version, and needed about 55,000 server queries to decrypt a symmetric key. The current version (1.6.2) was not vulnerable to the attacks. This is because the underlying libraries of Axis2 1.6.2 generate a random symmetric key in case the PKCS#1 decryption fails. This prevents from successful attacks, as discussed in Section 3.2.3.

5.1.2 CBC Attack

Both configurations could be attacked using XEW. We are not aware of any configuration that would protect against these attacks in the current Apache Axis2 version.

5.2 Apache CXF

For our tests we used a sample delivered by one of the Apache CXF developers. The example Web Service applies XML Signature and XML Encryption.

5.2.1 PKCS#1 Attack

The PKCS#1 attack could be applied thanks to an XSW attack combined with a *NoKeyRef* strategy. This means the `EncryptedKey` contained no reference to `EncryptedData`. In case of an incorrect `EncryptedKey`, a random symmetric key was generated in order to prevent further side channels [18]. The algorithm looked for the first `EncryptedData` structure referenced by the `EncryptedKey` and generated a random symmetric key for this `EncryptedData`. Since there was no `EncryptedData` referenced in our attack message, the server attempted to generate a random key for a default AES-128 algorithm. However, the server incorrectly generated a key of a 128-byte length (instead of 128 bits), which led to an internal exception and a different server response.

We reported this problem to the developers, who analyzed this incorrect behavior. The problem was fixed in versions 1.6.17 and 2.0.2 of the underlying WSS4J library.

5.2.2 CBC Attack

The default configuration could be attacked using XSW and XEW attacks.

In addition, we tested the server for further countermeasures. Apache CXF allows one to apply a configuration attribute `requireSignedEncryptedDataElements = "true"`, which ensures that the authenticity of the encrypted content is verified prior to decryption. With our new attack plugin, we found out that this countermeasure could be circumvented using an XSW attack.

We again reported this vulnerability to the Apache CXF developers. The XSW problem was then fixed in Apache CXF versions 1.6.17 and 2.0.2 of the underlying WSS4J library so that configuration attribute `requireSignedEncryptedDataElements = "true"` can now be used securely.

Framework	PKCS#1 Attack		CBC Attack		Countermeasures
	Type	Total Queries	Type	Queries / Byte	Applicable
Apache Axis2 1.6.2	–		XEW	14	no
Apache CXF 2.7.10	XSW+NoKeyRef	46,000	XSW/XEW	14	yes[a]
Axway Gateway 7.3.1	Direct	20,000	XSW/XEW	23[b]	yes[c]
IBM Datapower XI50	–		XEW	23[b]	yes[d]
Microsoft WCF	–		–		yes

Table 1: Evaluation results report attack application possibilities on the investigated XML security frameworks, including the number of requests needed to decrypt a ciphertext.

[a] After the framework was patched against the issues we reported.

[b] The different number of attack queries resulted from a different XML parsing technique applied in the gateway. For this reason, we needed to extend the original attack algorithm.

[c] With specific XPath expressions and unifying error messages.

[d] With specific XPath expressions.

5.3 Axway Gateway

For deployment of XML Signature and XML Encryption, Axway Gateway provides several configurations. These configurations allow one to enforce which elements have to be verified and which elements have to be decrypted. We first applied the default configuration, which defines that arbitrary elements have to be decrypted and signed. Afterwards, we analyzed possible countermeasures.

5.3.1 PKCS#1 Attack

It was possible to apply a direct attack using differences in error messages, see Figure 9. We found out that the server responded with a unified SOAP error message in a case we sent an invalid `EncryptedKey`. On the other hand, an `EncryptedKey` with a correctly formatted PKCS#1 message led to a simple `HTTP Error` message. This was because the server decrypted a symmetric key, which was of an invalid length so it could not be used to decrypt `EncryptedData`, or the decrypted symmetric key had a valid length but `EncryptedData` was decrypted to an unparsable content. This allowed us to distinguish valid from invalid messages and apply a Bleichenbacher attack directly.

5.3.2 CBC Attack

As mentioned above, the server responds with different error messages in cases where `EncryptedData` decryption fails. In order to modify ciphertexts in `EncryptedData` elements, XSW or XEW attacks were necessary. This allowed us to distinguish error messages and apply an attack against the symmetric encryption scheme.

As can be seen in the table, the attack needs about 23 queries to decrypt one byte. This number differs from the original paper [19] and results from a different XML parsing approach used in the gateway. More precisely, the parser accepts decrypted content if and only if the content contains at least one valid character (in comparison to an empty string, which is accepted by the parsers analyzed in the original paper). For this reason, we needed extend the original algorithm to handle this stricter XML parsing property, which resulted in a higher number of attack requests. Its description is behind the scope of this article.

5.3.3 Countermeasures

Axway Gateway offers several XPath expressions [6] to define concrete positions of signed and encrypted elements. However, most of these default expressions are insecure and allow us to apply XSW or XEW attacks.

In order to defend the CBC attack, it is possible to deploy the following secure configuration and define (see Figure 10):

- *What must be signed?*
  `/soap:Envelope/soap:Body` to ensure that all the `Body` elements are signed.
- *Nodes to decrypt?*
  `/soap:Envelope/soap:Body/enc:EncryptedData` to ensure that only `EncryptedData` elements inside of the (signed) `Body` element is decrypted. Others are ignored.

This is however not a solution for the PKCS#1 attack, since the attacker is still able to modify `EncryptedKey` elements. In order to protect from this attack, the user has to additionally unify the outgoing error messages. Another countermeasure would be to generate random symmetric keys in case the PKCS#1 decryption fails, as proposed in [18] and deployed by other analyzed frameworks.

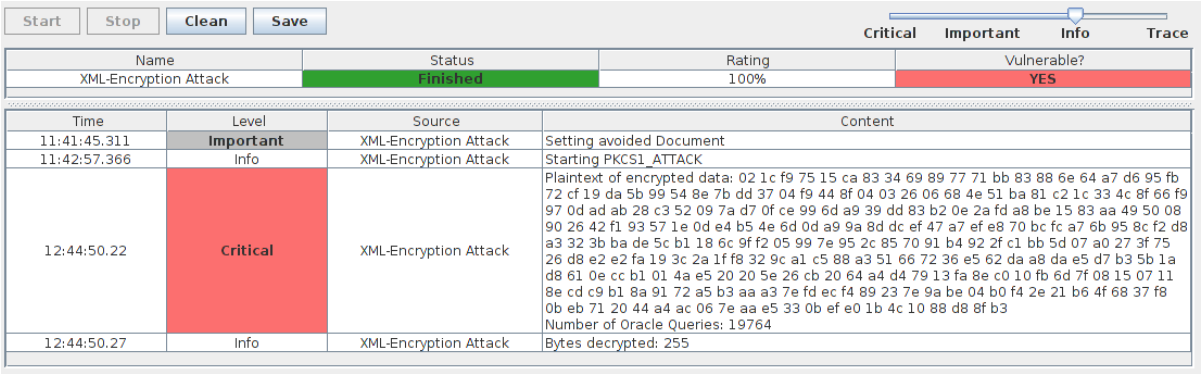

Figure 9: WS-Attacker shows the decrypted plaintext after the successful attack on the Axway Gateway.

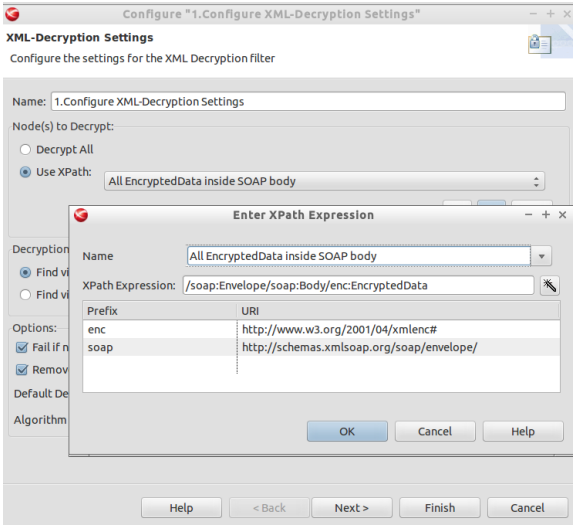

Figure 10: Countermeasures applicable in the Axway Gateway.

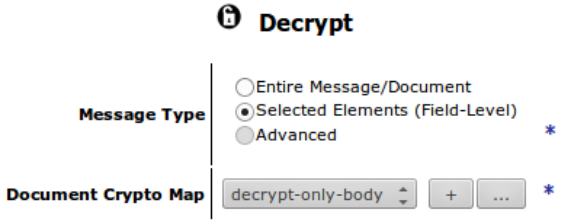

Figure 11: In order to restrict decryption of `EncryptedData` elements, *Selected Elements* configuration has to be used.

5.4 IBM Datapower

We tested IBM Datapower XI50 with the Firmware XI50.6.0.0.2. Similarly to the Axway Gateway, IBM Datapower offers several configurations combining XML Encryption with other security mechanisms. We first used the default configuration with XML Signature and XML Encryption for SOAP messages, which was vulnerable to the attack on CBC ciphertexts. Afterwards, we analyzed possible countermeasures together with IBM developers.

5.4.1 PKCS#1 Attack

We were not able to apply the attack on PKCS#1 ciphertexts. We analyzed the Datapower server logs and found out that Datapower generates a random symmetric key every time the PKCS#1 decryption fails. This makes the PKCS#1 attacks impractical, see Section 3.2.3.

5.4.2 CBC Attack

By default, IBM Datapower decrypts *all* the `EncryptedData` elements in the document. If the decryption of an `EncryptedData` element fails, the server just responds with the original encrypted content. Otherwise, the server proceeds with the decrypted message and its response differs. This allowed us to apply attacks on CBC ciphertexts. To overcome the XML Signature protection, we used the XEW technique.

As can be seen in Table 1, we needed about 23 server queries to decrypt one plaintext byte. This is because IBM Datapower uses a parsing mechanism that is similar to the one used by Axway Gateway.

5.4.3 Countermeasures

We discussed several countermeasures with IBM developers. It turned out that it is possible to restrict positions of `EncryptedData` elements that are going to be decrypted. In order to achieve this, the server administrator has to choose *Selected Elements (Field-Level)* in the decryption configuration, see Figure 11. Afterwards, he has to define an XPath for the element that is going to

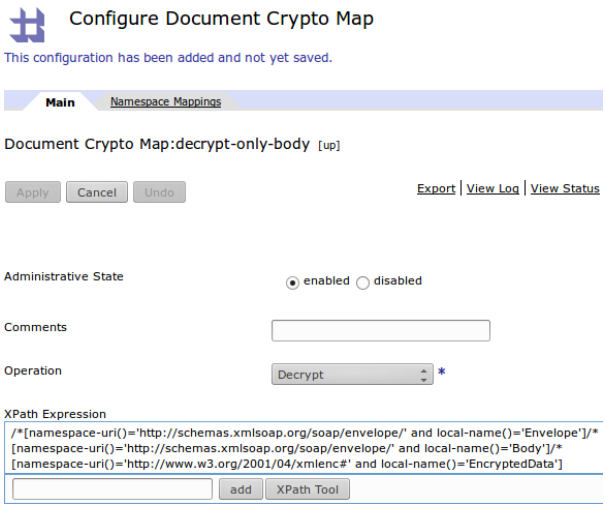

Figure 12: Specific positions for `EncryptedData` elements, which are going to be decrypted, can be restricted using XPath.

decrypted, see Figure 12. In addition, proper XPath expressions have to be defined for XML Signature validation step in order to protect authenticity of the encrypted data.

5.5 Microsoft WCF

Microsoft WCF was not vulnerable to the investigated attacks.

This framework allows a developer to define three different protection levels: `EncryptAndSign`, `Sign`, `Unprotected`. For our tests we used the `EncryptAndSign` profile, which applies a very strict XML processing:

- There is no possibility of including an additional `EncryptedKey` or `EncryptedData` element, to enforce decryption.
- Signatures are strictly verified only on specified fields. There is no possibility to apply an XSW attack.
- The error messages do not reveal any confidential data relevant to our attacks.

The tests on WCF were complex due to the fact that the generated messages include timestamps and messageIds so the messages could be used only once.

Thereby, Microsoft WCF provides a very good example on how to handle WS-Security: the configuration is secure by default, without a need of complex developer steps.

6 Related Work

Research related to this paper can be divided into three parts. (1.) research on the analysis of Web Services security in general, (2.) specific investigation in the field of XML Encryption, and (3.) further adaptive chosen-ciphertext attacks.

Security of Web Services. The security of Web Services was analyzed manually and without tool support in many researches [21, 20, 22]. In 2005, McIntosh and Austel found the first XML Signature Wrapping Attack [27]. This attack concept was later adopted on Amazon's Web Services [14, 31], but without any automatism or tool support. In 2012, WS-Attacker was developed as a first tool supporting fully-automatic Web Service specific attacks [26], and was then extended by a plugin for Denial-of-Service [13] and XML Signature Wrapping attacks [5].

XML Encryption. This paper is based on the attacks on symmetric and asymmetric encryption schemes in XML Encryption [19, 18]. These works cover cryptographic background behind the attacks and explain how to apply them in simple scenarios where XML Signatures are used to protect message authenticity. A complete analysis of countermeasures needed to be applied against these attacks was published in [33]. This work builds one important foundation to design our fully-automatic approach for the WS-Attacker plugin.

As a response to the attacks, W3C working group included an AES-GCM algorithm into the newest XML Encryption 1.1 specification and recommends to use RSA-OAEP. However, an analysis of Jager et al. revealed that there are still possibilities for backwards compatibility attacks, even if these secure cryptographic algorithms are supported [17]. The only prerequisite for the attacks is that the server also supports RSA-PKCS#1 v1.5 and CBC along with the secure algorithms. Backwards compatibility attacks are not covered by our plugin.

Adaptive Chosen-Ciphertext Attacks. In 1998, Bleichenbacher published an attack on RSA-PKCS#1 encryption scheme [4]. He described the basic algorithm behind the attack and how it can be applied to the SSL protocol if certain oracle is given. In [3] Bardou et al. improved Bleichenbacher's attack, and applied it to PKCS#11-based environments, e.g. Hardware Security Modules. A variant of Bleichenbacher's attack was explored in [7] in the context of EMV signatures (where the same RSA key pair may be used for both signature and encryption functions). In 2014, Meyer et al.

showed that is still possible to apply Bleichenbacher's attack against real TLS servers [28]. To adopt the above attacks, the authors used different error messages and timing side channels. Very recently, Zhang et al. showed that specific cross-tenant side channels allow for application of performant Bleichenbacher attacks in PaaS environments [36]. In their attack scenario, the attacker exploits a specific property of container-based tenant isolation, which allows him to share a CPU with his victim and apply flush-reload attacks.

In 2002, Vaudenay presented an attack on the Cipher Block Chaining (CBC) mode of operation [35]. The attack is possible due to a CBC property called malleability which allows an attacker to flip specific ciphertext bits without knowing the secret key. Strict structure of the CBC padding is used to construct an oracle which responds with 1 or 0 according to the padding validity. Thus, the attack is called a padding oracle attack. The padding oracle attack was later used to attack further standards with improved techniques, e.g. IPSec [8, 9], CAPTCHAs and the .NET framework [29], or TLS [2].

7 Conclusion

A deep knowledge of XML Security standards and applied cryptography is necessary to understand the attacks on XML Encryption. For developers and security testers, who use XML Encryption, it is thus hard to decide whether a Web Service is vulnerable or not.

Our paper presented a methodology to verify the existence of vulnerabilities in XML Encryption interfaces and automatically exploit possible attacks. As a result, we created an open source plugin for the penetration testing framework WS-Attacker. We hope it will enable even developers with basic XML and cryptographic skills to easily verify their implementations.

In a future work, our code could be extended to XML scenarios beyond Web Services, for example, to SAML, or even to JSON Web Encryption.

Acknowledgements

We would like to thank Colm O hEigeartaigh (Apache CXF), Krithika Prakash (IBM), and Philipp Schöne (Axway) for their cooperation. We also thank our anonymous reviewers for their helpful comments.

The research was supported by the *German Ministry of research and Education (BMBF)* as part of the VERTRAG research project.

References

[1] Configure IBM DataPower Gateways effectively to prevent XML Encryption attacks, July 2015. http://www-01.ibm.com/support/docview.wss?uid=swg21962335.

[2] Nadhem AlFardan and Kenneth G. Paterson. Plaintext-recovery attacks against Datagram TLS. In *Network and Distributed System Security Symposium (NDSS)*, February 2012.

[3] Romain Bardou, Riccardo Focardi, Yusuke Kawamoto, Lorenzo Simionato, Graham Steel, and Joe-Kai Tsay. Efficient padding oracle attacks on cryptographic hardware. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 608–625. Springer, August 2012.

[4] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In Hugo Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 1–12. Springer, August 1998.

[5] Christian Mainka. Automatic Penetration Test Tool for Detection of XML Signature Wrapping Attacks in Web Services, May 2012. Master thesis supervised by Jörg Schwenk and Juraj Somorovsky. http://nds.ruhr-uni-bochum.de/media/nds/arbeiten/2012/07/24/ws-attacker-ma.pdf.

[6] James Clark and Steven DeRose. XML path language (XPath) version 1.0. W3C recommendation, W3C, November 1999. http://www.w3.org/TR/1999/REC-xpath-19991116.

[7] Jean Paul Degabriele, Anja Lehmann, Kenneth G. Paterson, Nigel P. Smart, and Mario Strefler. On the joint security of encryption and signature in EMV. In Orr Dunkelman, editor, *Topics in Cryptology – CT-RSA 2012*, volume 7178 of *Lecture Notes in Computer Science*, pages 116–135. Springer, February / March 2012.

[8] Jean Paul Degabriele and Kenneth G. Paterson. Attacking the IPsec standards in encryption-only configurations. In *IEEE Symposium on Security and Privacy*, pages 335–349, 2007.

[9] Jean Paul Degabriele and Kenneth G. Paterson. On the (in)security of IPsec in MAC-then-encrypt configurations. In *ACM Conference on Computer and Communications Security*, pages 493–504, 2010.

[10] Morris Dworkin. Recommendation for block cipher modes of operation: Methods and techniques, December 2001.

[11] Morris Dworkin. Recommendation for block cipher modes of operation: Galois/counter mode (GCM) and GMAC. In *NIST Special Publication 800-38D*, November 2007.

[12] Donald Eastlake, Joseph Reagle, Frederick Hirsch, Thomas Roessler, Takeshi Imamura, Blair Dillaway, Ed Simon, Kelvin Yiu, and Magnus Nyström. XML Encryption Syntax and Processing 1.1. *W3C Candidate Recommendation*, 2012. `http://www.w3.org/TR/2012/WD-xmlenc-core1-20121018`.

[13] Andreas Falkenberg, Christian Mainka, Juraj Somorovsky, and Jörg Schwenk. A New Approach towards DoS Penetration Testing on Web Services. In *IEEE 20th International Conference on Web Services (ICWS)*, pages 491–498. IEEE, 2013.

[14] Nils Gruschka and Luigi Lo Iacono. Vulnerable Cloud: SOAP Message Security Validation Revisited. In *ICWS '09: Proceedings of the IEEE International Conference on Web Services*, Los Angeles, USA, 2009. IEEE.

[15] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik F. Nielsen, Anish Karmarkar, and Yves Lafon. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). Technical report, April 2007.

[16] Frederick Hirsch, David Solo, Joseph Reagle, Donald Eastlake, and Thomas Roessler. XML signature syntax and processing (second edition). W3C recommendation, W3C, June 2008.

[17] Tibor Jager, Kenneth G. Paterson, and Juraj Somorovsky. One Bad Apple: Backwards Compatibility Attacks on State-of-the-Art Cryptography. In *Network and Distributed System Security Symposium (NDSS)*, 2013.

[18] Tibor Jager, Sebastian Schinzel, and Juraj Somorovsky. Bleichenbacher's Attack Strikes again: Breaking PKCS#1 v1.5 in XML Encryption. In *ESORICS*, pages 752–769, 2012.

[19] Tibor Jager and Juraj Somorovsky. How To Break XML Encryption. In *The 18th ACM Conference on Computer and Communications Security (CCS)*, October 2011.

[20] Meiko Jensen, Nils Gruschka, and Ralph Herkenhöner. A survey of attacks on web services. *Computer Science-Research and Development*, 24(4):185–197, 2009.

[21] Meiko Jensen, Nils Gruschka, Ralph Herkenhoner, and Norbert Luttenberger. Soa and web services: New technologies, new standards-new attacks. In *Web Services, 2007. ECOWS'07. Fifth European Conference on*, pages 35–44. IEEE, 2007.

[22] Meiko Jensen, Jörg Schwenk, Nils Gruschka, and Luigi Lo Iacono. On technical security issues in cloud computing. In *Cloud Computing, 2009. CLOUD'09. IEEE International Conference on*, pages 109–116. IEEE, 2009.

[23] B. Kaliski. PKCS #1: RSA Encryption Version 1.5. RFC 2313 (Informational), March 1998. Obsoleted by RFC 2437.

[24] Christian Mainka, Meiko Jensen, Luigi Lo Iacono, and Jörg Schwenk. XSpRES-Robust and Effective XML Signatures for Web Services. In *CLOSER*, pages 187–197, 2012.

[25] Christian Mainka, Vladislav Mladenov, Florian Feldmann, Julian Krautwald, and Jörg 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*, CCSW '14, pages 93–104, New York, NY, USA, 2014. ACM.

[26] Christian Mainka, Juraj Somorovsky, and Jörg Schwenk. Penetration Testing Tool for Web Services Security. In *SERVICES Workshop on Security and Privacy Engineering*, June 2012.

[27] Michael McIntosh and Paula Austel. XML signature element wrapping attacks and countermeasures. In *SWS '05: Proceedings of the 2005 Workshop on Secure Web Services*, pages 20–27, New York, NY, USA, 2005. ACM Press.

[28] Christopher Meyer, Juraj Somorovsky, Eugen Weiss, Jörg Schwenk, Sebastian Schinzel, and Erik Tews. Revisiting SSL/TLS implementations: New bleichenbacher side channels and attacks. In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, pages 733–748, 2014.

[29] Juliano Rizzo and Thai Duong. Practical padding oracle attacks. In *Proceedings of the 4th USENIX conference on Offensive technologies*, WOOT'10, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association.

[30] Juraj Somorovsky. On the Insecurity of XML Security (Doctoral dissertation), July 2013. Ruhr University Bochum, Germany. `https://www.nds.rub.de/research/publications/xmlinsecurity`.

[31] Juraj Somorovsky, Mario Heiderich, Meiko Jensen, Jörg Schwenk, Nils Gruschka, and Luigi Lo Iacono. All Your Clouds are Belong to us – Security Analysis of Cloud Management Interfaces. In *The ACM Cloud Computing Security Workshop (CCSW)*, October 2011.

[32] Juraj Somorovsky, Andreas Mayer, Jörg Schwenk, Marco Kampmann, and Meiko Jensen. On Breaking SAML: Be Whoever You Want to Be. In *21st USENIX Security Symposium*, Bellevue, WA, August 2012.

[33] Juraj Somorovsky and Jörg Schwenk. Technical Analysis of Countermeasures against Attack on XML Encryption – or – Just Another Motivation for Authenticated Encryption. In *SERVICES Workshop on Security and Privacy Engineering*, June 2012.

[34] UK Sheffield University. SimMetrics.

[35] Serge Vaudenay. Security flaws induced by CBC padding - applications to SSL, IPSEC, WTLS ... In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 534–546. Springer, April / May 2002.

[36] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Cross-Tenant Side-Channel Attacks in PaaS Clouds. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 990–1003, New York, NY, USA, 2014. ACM.