The State of XML Digital Signatures

Henrich C. Pöhls
Institute of IT-Security and Security Law
University of Passau, Germany
hp@sec.uni-passau.de
+498515093217
me, myself and this talk

- M.Sc. Information Security from Royal Holloway
- Diplom Informatik from University of Hamburg
- currently PhD student at University of Passau

ReSCUe IT:

- General: IT supported robust & secure Supply Chains
- Our Goal: Legally compliant & manageable integrity and authenticity statements for the data
Authenticate not just the message, but everything that is used to determine the meaning of the message.

Ferguson and Schneier
Authenticate not just the message, but everything that is used to determine the meaning of the message.

Verifying a Digital Signature on the input (XML message)

Today's most common approach is to let application process the input (XML message).
XML Digital Signature Basics

- Overview
- Detail: Syntax and Processing Steps

XML Digital Signature Complexity and Pitfalls

- Problem: Semantic Gap between Vrfy/Sign and App
- Example: XML Wrapping Attacks
- Solution: Ensure Application Control

Sanitizable Signatures within XML

- Chameleon Hashes
- Sanitizable Signatures within the standard

XML Digital Signature Syntax and Processing 2.0

- currrent W3C DRAFT
Keep in mind two questions:

Why do we sign and verify?

What do we want to verify?
Digital Signature in ISO/IEC 9798-1

A cryptographic transformation of a data unit that allows a recipient of the data unit to prove the origin and integrity of the data unit and protect the sender and the recipient of the data unit against forgery by third parties, and the sender against forgery by the recipient.
Digital Signature in XML

1. XML representation (Syntax) of information related to the Signature, like:
   - Signature Value
   - Algorithms used in the processes
   - Reference to data that was signed

2. two Processes
   - Signature Generation
   - Signature Verification
#1 Overview of the Process

```xml
<xml>
  <data>
    <important id="1">aaa</important>
    <other id="2">bbb</other>
  </data>
</xml>

1<sup>st</sup> selection sign important data → id="1"
```
Overview of the Process

<xml>
  <data>
    <important id="1">aaa</important>
    <other id="2">bbb</other>
  </data>
</xml>

1st selection

2nd transform

sign important data ➔ id="1"

i.e. C14N (canonicalization)
#1 Overview of the Process

```xml
<xml>
  <data>
    <important id="1">aaa</important>
    <other id="2">bbb</other>
  </data>
</xml>

1\textsuperscript{st} selection \quad sign important data $\rightarrow$ id="1"

2\textsuperscript{nd} transform \quad i.e. C14N (canonicalization)

3\textsuperscript{rd} digest (hash) \quad SHA1 is "zcj4...hN4yk="
#1 Overview of the Process

```xml
<xml>
  <data>
    <important id="1">aaa</important>
    <other id="2">bbb</other>
  </data>
</xml>
```

1<sup>st</sup> selection  sign important data → id="1"
2<sup>nd</sup> transform  i.e. C14N (canonicalization)
3<sup>rd</sup> digest (hash)  SHA1 is "zcj4...hN4yk="

... repeat step 1-3 for each data part to sign ...
#1 Overview of the Process

```xml
<xml>
  <data>
    <important id="1">aaa</important>
    <other id="2">bbb</other>
  </data>
</xml>
```

1<sup>st</sup> selection: sign important data → id="1"

2<sup>nd</sup> transform: i.e. C14N (canonicalization)

3<sup>rd</sup> digest (hash): SHA1 is “zcj4...hN4yk=“

4<sup>th</sup> build `<SignedInfo>`: ....
<SignedInfo>
  <CanonicalizationMethod Algorithm="REC-xml-c14n-20010315"/>
  <SignatureMethod Algorithm="xmldsig#rsa-sha1"/>
  <Reference URI="#xpointer(id('1'))">
    <Transforms>
      <Transform Algorithm="xmldsig#enveloped-signature"/>
    </Transforms>
    <DigestMethod Algorithm="xmldsig#sha1"/>
    <DigestValue>zcj4...hN4yk=</DigestValue>
  </Reference>
</SignedInfo>
Overview of the Process

1\textsuperscript{st} selection  
   sign important data \(\rightarrow\) id="1"

2\textsuperscript{nd} transform  
i.e. C14N (canonicalization)

3\textsuperscript{rd} digest (hash)  
generate \texttt{<DigestValue>}

4\textsuperscript{th} build \texttt{<SignedInfo>}  
   selection \texttt{<Reference>} and their \texttt{<DigestValue>} \& other signature related info i.e. digest and signature algorithms
Overview of the Process

1\textsuperscript{st} selection | sign important data → id="1"
2\textsuperscript{nd} transform | i.e. C14N (canonicalization)
3\textsuperscript{rd} digest (hash) | generate <DigestValue>
4\textsuperscript{th} build <SignedInfo> | selection <Reference> and their <DigestValue> & other signature related info i.e. digest and signature algorithms
5\textsuperscript{th} transform <SignedInfo> | C14N of <SignedInfo>
Overview of the Process

1st selection  
2nd transform  
3rd digest (hash)  
4th build <SignedInfo>  
5th transform <SignedInfo>  
6th sign <SignedInfo>

- sign important data → id="1"
- i.e. C14N (canonicalization)
- generate <DigestValue>
- selection <Reference> and their <DigestValue> & other signature related info i.e. digest and signature algorithms
- C14N of <SignedInfo>
- generate <SignatureValue>
#1  Overview of the Process

1st selection  sign important data → id="1"
2nd transform  i.e. C14N (canonicalization)
3rd digest (hash)  generate `<DigestValue>`
4th build `<SignedInfo>`  selection `<Reference>` and their `<DigestValue>` & other signature related info i.e. digest and signature algorithms
5th transform `<SignedInfo>`  C14N of `<SignedInfo>`
6th sign `<SignedInfo>`  generate `<SignatureValue>`
7th build `<Signature>`  build the `<Signature>` XML
#1 Abbreviated <Signature>

```xml
<Signature>
  <SignedInfo> ...
    <Reference URI="#xpointer(id('1'))">
      <Transforms> ...
        </Transforms>
    ...
      <DigestValue>zcj4...hN4yk=
        </DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>IBvYNuSHdvfP...u+xk=
    </SignatureValue>
</Signature>
```
#1 XML Digital Signature Basics
- Overview
- Detail: Syntax and Processing Steps

#2 XML Digital Signature Complexity and Pitfalls
- Problem: Semantic Gap between Vrfy/Sign and App
- Example: XML Wrapping Attacks
- Solution: Ensure Application Control

#3 Sanitizable Signatures within XML
- Chameleon Hashes
- Sanitizable Signatures within the standard

#4 XML Digital Signature Syntax and Processing 2.0
- current W3C DRAFT
1. Die Bahnsteigposter dürfen nicht eigenmächtig geöffnet werden
2. Den Anweisungen des Bedienungspersonals ist Folge zu leisten.
3. Eigenmächtiges Einsteigen ist verboten.
4. Ski und Stöcke in Fahrtrichtung halten.
5. Schaukeln verboten.
6. Überqueren des Bahnsteiges ist verboten.
Semantic Gap: Message/Transport Layer vs. Application Layer

Different Layers:

- **Application “Layer”**
  - App. extracts data from message
  - App. logic works on data

- **Message / Transport “Layer”**
  - Msg. contains input data for application
Semantic Gap: Message/Transport Layer vs. Application Layer

Different Layers:

- **Application “Layer”**
  - App. extracts data from message
  - App. logic works on data

- **Message / Transport “Layer”**
  - Msg. contains input data for application

Signature
Sign / Verify
Black-Box

configure

result

Henrich C. Pöhls
#2 Obvious Solution: Apply XML msg. security

SOAP
- WS-Security (Tokens ... )
- XML Signature (and Encryption)

XML Security
- well defined XML schema
- rigorous schema validation
- validity check of signing public-key
- enforce strict security policies
Obvious Solution: Apply XML msg. security

SOAP
- WS-Security (Tokens … )
- XML Signature (and Encryption)

XML Security
- well defined XML schema
- rigorous schema validation
- validity check of signing public-key
- enforce strict security policies

... but ...
XML Signature Wrapping

graphic taken from: Meiko Jensen - Ruhr Universität Bochum - http://www.nds.rub.de/chair/people/meiko-jensen/
XML Signature Wrapping

document root

Envelope

Header
  Shipping
  Wrapper
  Security

Body
  Manifest
    Id="newMf1"
    CId="c1"
  (Modified Content)

(Content)

graphic taken from:  Meiko Jensen - Ruhr Universität Bochum - http://www.nds.rub.de/chair/people/meiko-jensen/
#2 Loss of Context → XML Signature Wrapping

- Header
- Wrapper
- Security

- Manifest
- (Content)

- Manifest
- (Modified Content)
  - Id="newMf1"
  - CId="c1"

- IDRef: URI='#mf1'

Processed by application logic

Document root

Envelope

Shipping

Protected by signature

Graphic from: Meiko Jensen - Ruhr Universität Bochum - http://www.nds.rub.de/chair/people/meiko-jensen/
Loss of Context → XML Signature Wrapping

graphic from: Meiko Jensen  
Ruhr Universität Bochum

graphic taken from: Meiko Jensen - Ruhr Universität Bochum - http://www.nds.rub.de/chair/people/meiko-jensen/
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
    <soap:Body Id="1">
      <nds:return_hash> <!--Optional:--> </nds:return_hash>
      <nds:name>fffdf g</nds:name>
    </soap:Body>
  </soap:Header>
  <soap:Body>
    <nds:return_hash> <!--Optional:--> </nds:return_hash>
    <nds:name>evilHomer</nds:name>
  </soap:Body>
</soap:Header>
  <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
  <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
  <Reference URI="#xpointer(id('1'))"><Transforms>
    <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
    <DigestMethod

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:nds="http://www.example.com/namespaces">
  <soap:Header/>
  <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
    <soap:Body Id="1">
      <nds:return_hash> <!--Optional:--> </nds:return_hash>
      <nds:name>?fff d g</nds:name>
    </soap:Body>
  </soap:Header>
  <soap:Body>
    <nds:return_hash> <!--Optional:--> </nds:return_hash>
    <nds:name>evilHomer</nds:name>
  </soap:Body>
</soap:Header>
</soap:Envelope>

  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <Reference URI="#xpointer(id(1))"/>
  </SignedInfo>
  <Transforms>
    <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
    <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha1"/>
  </Transforms>
</Signature>
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:nds="">
    <soap:Header/>
    <soap:Body>
        <nds:return_hash> <!--Optional:-->
            <nds:name>fffdf g</nds:name>
        </nds:return_hash>
    </soap:Body>
</soap:Header>
<soap:Body>
    <nds:return_hash> <!--Optional:-->
        <nds:name>evilHomer</nds:name>
    </nds:return_hash>
</soap:Body>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#"> <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <Reference URI="#xpointer(id('1'))"> <Transforms>
        <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
        <DigestMethod

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:nds="http://ceo.net主权名称服务器"
<soap:Header>
<nds:return_hash> <!--Optional:-->
<nds:name>?fffd g</nds:name>
</nds:return_hash>
</nds:return_hash>
</soap:Body>
</soap:Header>
<soap:Body>
<nds:return_hash> <!--Optional:--> 
<nds:name>evilHomer</nds:name>
</nds:return_hash>
</soap:Body>
</soap:Body>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#"><SignedInfo>
<CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
<Reference URI="#xpointer(id('1'))"><Transforms>
<Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
</Transform>
</Reference>
</SignedInfo>
<SignatureValue>...
</Signature>
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:nds="urn:nds:schema">
  <soap:Header/>
  <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
    <nds:return_hash> <!--Optional:-->
      <nds:name>?fffdf g</nds:name>
    </nds:return_hash>
  </wsse:Security>
  <soap:Body>
    <nds:return_hash> <!--Optional:-->
      <nds:name>evilHomer</nds:name>
    </nds:return_hash>
  </soap:Body>
</soap:Envelope>

<Signature xmlns="http://www.w3.org/2000/09/xmldsig#" xmlns:nds="urn:nds:schema">
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <Reference URI="#xpointer(id('1'))"/>
  </SignedInfo>
</Signature>
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-assertions-1.0-assert.xsd">
<soap:Header>
<soap:Body Id="1">
<nds:return_hash> <!--Optional:--> </nds:return_hash>
<nds:name>fffd g</nds:name>
</soap:Body>
</soap:Header>
</soap:Body>
</soap:Envelope>

SOAP message example is from:

Meiko Jensen
Ruhr Universität Bochum
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:nds="http://www.example.com/nds"
<soap:Header>
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
<soap:Body Id="1">
<nds:return_hash> <!--Optional:--> </nds:return_hash>
<nds:name>?fffdb g</nds:name>
</nds:return_hash>
</soap:Body>
</soap:Header>
<soap:Body>
<nds:return_hash> <!--Optional:--> </nds:return_hash>
<nds:name>evilHomer</nds:name>
</nds:return_hash>
</soap:Body>
</soap:Header>
<SignedInfo>
<CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
<Reference URI="#xpointer(id('1'))"></Transforms>
<Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
</Reference>
</SignedInfo>
</Signature>
Not covered by Signature
Covered by Signature

XML file is corrupt
Signature couldn't be found

Internals from JAVA verify process:
javax.xml.crypto.dsig.XMLSignature
Marshal Exception
Nullpointer while unmarshaling
Not covered by Signature
Covered by Signature
XML file is corrupt
Signature couldn't be found
Internals from JAVA verify process:
javax.xml.crypto.dsig.XMLSignature
Marshal Exception
Nullpointer while unmarshaling
Application logic extracts:
```
<soap:Header>
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
</soap:Header>
<soap:Body Id="1">
<nds:return_hash> <!--Optional:--> </nds:return_hash>
<nds:name>?fffdf g</nds:name>
</soap:Body>
</soap:Body>
<soap:Body>
<nds:return_hash> <!--Optional:--> </nds:return_hash>
<nds:name>evilHomer</nds:name>
</nds:return_hash>
</soap:Body>
</soap:Envelope>
```

BitFlip Test on:
```
<soap:Header>
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
</soap:Header>
<soap:Body Id="1">
<nds:return_hash> <!--Optional:--> </nds:return_hash>
<nds:name>evilHomer</nds:name>
</soap:Body>
</soap:Envelope>
```
Signed Word Document “internal” XML files

Not covered by Signature
Covered by Signature
Signed Word Document “internal” XML files

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
<Relationship Id="rId8" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/theme" Target="theme/theme1.xml"/>
<Relationship Id="rId3" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/styles" Target="styles.xml"/>
<Relationship Id="rId7" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/fontTable" Target="fontTable.xml"/>
<Relationship Id="rId2" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/numbering" Target="numbering.xml"/>
<Relationship Id="rId1" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/customXml" Target="/..\customXml\item1.xml"/>
<Relationship Id="rId5" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/webSettings" Target="webSettings.xml"/>
<Relationship Id="rId4" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/settings" Target="settings.xml"/></Relationships>
Signed Word Document: document.rels.xml

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
Covered by Signature
XML file is corrupt

Institute of IT-Security and Security Law

Henrich C. Pöhls

OWASP 44
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
xxmlns:dcterms="http://purl.org/dc/terms/"
xxmlns:dcmitype="http://purl.org/dc/dcmitype/"
Signed Word Document: docProbs/core.xml

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
xmlns:dc:creator="matthias"
xmlns:dc:creator><cp:lastModifiedBy="matthias"
</cp:lastModifiedBy><cp:revision>3</cp:revision><dcterms:created
xsi:type="dcterms:W3CDTF">2010-10-05T08:57:00Z</dcterms:created>><dcterms:modified
xsi:type="dcterms:W3CDTF">2010-10-13T14:53:00Z</dcterms:modified></cp:coreProperties>
```

Not covered by Signature

Covered by Signature

XML file is corrupt
Make sure you get the important data signed!

Make sure you know what data was verified!

Make sure you rely only on verified data!
the application

Make sure you get the important data signed!

Make sure you know what data was verified!

Make sure you rely only on verified data!
Make sure you get the important data signed!

Make sure you know what was verified!

Make sure you rely only on verified data!
Make sure you get the important data signed!

Make sure you know what data was verified!

Make sure you rely only on verified data!
Make sure you get the important data signed!

Make sure you know what data was verified!

Make sure you rely only on verified data!
Make sure you get the important data signed!

Make sure you know what was verified!

Make sure you rely only on verified data!
# XML Digital Signature Basics
- Overview
- Detail: Syntax and Processing Steps

# XML Digital Signature Complexity and Pitfalls
- Problem: Semantic Gap between Vrfy/Sign and App
- Example: XML Wrapping Attacks
- Solution: Ensure Application Control

# Sanitizable Signatures within XML
- Chameleon Hashes
- Sanitizable Signatures within the standard

# XML Digital Signature Syntax and Processing 2.0
- current W3C DRAFT


#3 Integrity protection by digital signatures

Hash-and-Sign process: $\text{SIGN} ( h(m)) = \text{sig}$

cryptographic hash function $h()$

- ensures that no bit of $m$ is allowed to change
- computationally infeasible to find a collision
Loosen Integrity protection ...

Hash-and-Sign process: \( \text{SIGN} \left( \text{ch}(m) \right) = \text{sig} \)

chameleon hash function \( \text{ch}() \)

- someone who knows the trapdoor can find a collision
- enables that someone to change \( m \) without invalidating signature
- otherwise, computationally infeasible to find collision
#3 Sanitizable Signatures

a) Original

b) Blackening

c) Pseudonymization

d) DeIdentification
Sanitizable Signatures


Sanitizable Signatures within XML Signature Standard

Chameleon-Hash by Krawczyk et al.
Chameleon-Hash by Zheng et al
Chameleon-Hash by Chen et al.
Chameleon-Hash by Ateniese et al.
Scheme by Miyazaki et al.

- Implementation (the first of some schemes) in JAVA
- XML Digital Signature is flexible enough to allow it
- First Performance results show they are usable
<Signature xmlns=http://www.w3.org/2000/09/xmldsig#>
  <SignedInfo>
    <CanonicalizationMethod Algorithm=http://www.w3.org/2001/10/xml-exc-c14n# />
    <SignatureMethod Algorithm=http://www.w3.org/2000/09/xmldsig#rsa-sha1 />
    <Reference URI=#xpointer(id('8492340'))>
      <Transforms>
        <Transform Algorithm=http://www.w3.org/2000/09/xmldsig#enveloped-signature/>
      </Transforms>
      <DigestMethod Algorithm=http://www.w3.org/2000/09/xmldsig#sha1 />
      <DigestValue>ysP4CnBuIqMs9v7HEmKXRhtt+60=</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>Pjvvhb5NuIcrRoySJ1hUb/N...8MUEU01YCA==</SignatureValue>
  <KeyInfo><KeyValue><RSAKeyValue>
    <Modulus>gzma/Xbgfdn....rkMfjQ==</Modulus> <Exponent>AQAB</Exponent>
  </RSAKeyValue></KeyValue></KeyInfo>
</Signature>
<Signature xmlns=http://www.w3.org/2000/09/xmldsig#>
  <SignedInfo>
    <CanonicalizationMethod Algorithm=http://www.w3.org/2001/10/xml-exc-c14n# />
    <SignatureMethod Algorithm=http://www.w3.org/2000/09/xmldsig#rsa-sha1 />
    <Reference URI=#xpointer(id('8492340'))>
      <Transforms>
        <Transform Algorithm=http://www.w3.org/2000/09/xmldsig#enveloped-signature/>
      </Transforms>
      <DigestMethod Algorithm="http://www.example.org/xmldsig-more#chamhashid">
        <ChamIdKeyValue><x>eDF2+AXRZ...a3B81U="/x>
          <y>AZux4whBS...51pIPNf</y>
          <receiver>UGV0ZXI="/receiver>
        </ChamIdKeyValue>
      </DigestMethod>
      <DigestValue>XX6kzLw+SZnV76wcD4/y7k1iIKaTKoVkm5ZosDqMqTY="/DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>KGXP6GsG7Pi...4CRdsbxFIG0a+vA====</SignatureValue>
  <KeyInfo><KeyValue><RSAKeyValue>
    <Modulus>gzma/Xbgfdn....rkMfjQ==</Modulus> <Exponent>AQAB</Exponent>
  </RSAKeyValue></KeyValue>
</KeyInfo>
</Signature>
Sanitizable Signatures within XML Signature Standard

Figure 6.21: Generation

Figure 6.22: Validation
XML Digital Signature is very flexible!

Even non-standard signature schemes like sanitizable signatures fit-in!

Sanitizable signatures will allow you to specify who is authorized to change what!
1. XML Digital Signature Basics
   - Overview
   - Detail: Syntax and Processing Steps

2. XML Digital Signature Complexity and Pitfalls
   - Problem: Semantic Gap between Vrfy/Sign and App
   - Example: XML Wrapping Attacks
   - Solution: Ensure Application Control

3. Sanitizable Signatures within XML
   - Chameleons Hashes
   - Sanitizable Signatures within the standard

4. XML Digital Signature Syntax and Processing 2.0
   - current W3C DRAFT

Institute of IT-Security
and Security Law

Henrich C. Pöhls

OWASP
<Signature Id="MyFirstSignature" xmlns="xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod Algorithm="xml-c14n2"/>
    <SignatureMethod Algorithm="xmldsig-more#rsa-sha256"/>
    <Reference>
      <Transforms>
        <Transform Algorithm="xmldsig2#transform">
          <dsig2:Selection type="xmldsig2#xml" xmlns:dsig2="xmldsig2#" URI="http://www.w3.org/TR/2000/REC-xhtml1-20000126">
            </Selection>
        </Transform>
        <CanonicalizationMethod Algorithm="xml-c14n2"/>
      </Transforms>
      <DigestMethod Algorithm="sha256"/>
      <DigestValue>dGhpcyBpcyBub3QgYS...</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>dDFMsAD52sDFAad...</SignatureValue>
</Signature>
Changes to `<Reference>`

No more transforms: each reference object has

- Selection element to choose data to be signed
- One C14N element to convert the data

No more URI attribute in `<Reference>  →  Selection`
dsig2:Verification new optional element contains information to help in signature Verification

<dsig2:DigestDataLength>
  ▸ integer that specifies the number of bytes that were digested in this reference.
  ▸ debug digest verification failures
  ▸ indicate intentional signing of 0 bytes
  ▸ bypass digest calculation if length of the byte array containing the canonicalized bytes doesn't match the DigestDataLength found
dsig2:Verification new optional element contains information to help in signature Verification

<dsig2:DigestDataLength>
<dsig2:PositionAssertion>
  ▶ enable ID-based referencing to be more resistant to signature wrapping attacks
  ▶ contains an XPath expression
  ▶ ID-based referenced content must be within this Xpath position
dsig2:Verification new optional element contains information to help in signature Verification

<dzig2:DigestDataLength>
<dzig2:PositionAssertion>
<dzig2:IDAttributes>
  ▸ lists the signer`s ID attributes that the signer has used
  ▸ optional hint for the verifier which attributes should have „Idness“-properties
Make sure your application knows how its signature scheme works !!!
BACKUP - BitFlip
What BitFlip does ...

- detects absence of integrity protection ("white spots")
- works independently of signature verification process ("black-box")
- implemented on application level
  - application controlled
  - use same "parser logic" to select flipping data
- absence can be detected by a single "flip"
  - overhead of one additional signature verification
BitFlip: Conclusion

- Allows Applications to test if Signature Verification Process covers the data the application logic extracted
- Independent of Verification Process (black-box)
  - Full Verification not necessary if no black-box
- Tool to evaluate the Verification Process
  - detect errors during application design
  - testing the layers below before application roll-out
  - re-run tests after changes to the policy or the verification process
BitFlip: Observing the Signature Verification Outcome on Application Induced Errors
BitFlip: Observing the Signature Verification Outcome on Application Induced Errors

Signature Verification

VERIFIED

Henrich C. Pöhls
BitFlip: Observing the Signature Verification Outcome on Application Induced Errors

BitFlip: controlled change of single character
BitFlip: Observing the Signature Verification Outcome on Application Induced Errors

BitFlip: controlled change of single character
BitFlip: Observing the Signature Verification Outcome on Application Induced Errors

BitFlip: controlled change of single character
BitFlip: Observing the Signature Verification Outcome on Application Induced Errors

BitFlip: controlled change of single character
BitFlip: Observing the Signature Verification Outcome on Application Induced Errors

Result: character not covered
BitFlip: Observing the Signature Verification Outcome on Application Induced Errors

BitFlip: controlled change of single character

Signature Verification

VERIFIED ✔

Signature Verification

✔ FAILURE
BitFlip: Observing the Signature Verification Outcome on Application Induced Errors

**Result:** character covered
BitFlip: BlackBox' ing MS Word Signatures
BitFlip: BlackBox’ ing MS Word Signatures

Finalize HTML result file by closing HTML-Tags

NO

i>0?

YES

i = i - 1

NO

k>0?

YES

Perform Bitflip on k-th Character in file A

Rebuild Word document W with file A and other ZipEntries

NO

Open i-th ZipEntry

Create XML-File A and copy content from i-th ZipEntry to A

k = #Characters in file A

END

Word (Blackbox)

Open signed Word document W

Start VBA-Macro „verifySignature“

Let macro verify Signature

Return Result

Which verification result?
BitFlip: BlackBox'ing MS Word Signatures

**Word (Blackbox)**

1. Open signed Word document \(W\)
2. Start VBA-Macro „verifySignature“
3. Let macro verify Signature
4. Return Result

**Decision Point:** Which verification result?
- **VALID:** Mark character \(k\) as red
- **INVALID:** Mark character \(k\) as green
- **CORRUPT:** Mark character \(k\) as black

**Next Steps:**
- Write marked character \(k\) to HTML result file
- \(k = k - 1\)