

Mag. iur. Dr. techn. Michael Sonntag

XML signing and encryption

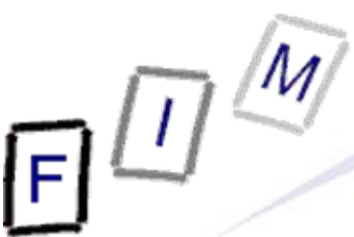
With Java examples

Institute for Information Processing and
Microprocessor Technology (FIM)
Johannes Kepler University Linz, Austria

E-Mail: sonntag@fim.uni-linz.ac.at
<http://www.fim.uni-linz.ac.at/staff/sonntag.htm>



- XML security consists of two independent parts:
 - XML Signature: Allows signing XML documents (non-repudiation)
 - XML Encryption: Allows encrypting XML documents (secrecy)
- Both trivially accomplished by existing technologies/standards
 - But only for the **complete** file!
 - » This prevents e.g. writing the signature into the XML file itself!
 - » Locating parts is no longer possible in encrypted files
 - » Tags are also encrypted ⇒ known plaintext attack possible
 - » No schema validation while encrypted
 - Solution: Standards for encrypting/signing **parts** of XML files
- Problem: XML content may differ binary but be logically the same
 - E.g. linefeeds, blanks, entity style/replacement, CDATA sections,...
 - Solution: Canonical XML
 - » Specific "formatting" which always produces the same binary result



XML Canonicalization

- Produce a unique physical representation of an XML fragment
 - Not foolproof: Even more strict is "Exclusive XML Canonicalization"
 - Works not really well for parts which are not well-formed
- Unifies:
 - Character set: Always UTF-8 in NFC (=Normalization Form C)
 - Linebreaks: Always #x0A
 - Attribute values: Normalized, double quotes, default attr. added
 - Content text: CDATA, entities, special characters, ...
 - Superfluous elements: XML declaration, DTD, unneeded NS
 - Extraneous whitespace: Within tags, outside of document element
 - Ordering: Attributes within a tag, namespace declarations
- Limitations:
 - Base URIs, notations, external unparsed entity references, attribute types in DTD



- A signature consists of
 - The actual signature value (Base64 encoded)
 - Signature information:
 - » Canonicalization, signature, digest method
 - » What was actually signed: URI/XPath, ...; additional transformations
 - Information on the key to use for verification
 - » E.g. certificate (X.509, PGP, ...), key name, ...
 - Object information: What is actually signed
 - Additional properties: E.g. timestamp
- Three kinds of signatures exist
 - Enveloping: Signed data contained within the *Object* information
 - Enveloped: An ancestor of the signature is signed
 - » The signature itself must be excluded from digesting, obviously!
 - Detached: External content (identified by *URI* or *Transform*)



XML Signature Transformations

- Describes how to obtain the data object to be digested
 - Ordered list: Result of first is input for second, ...
- Each transform consists of an algorithm and appr. attributes
- Examples:
 - Two enveloped signatures required: Each signature must exclude itself, but it must **also** exclude the other one!
- Enveloped transform: Equivalent to the following XPath transform
 - `<XPath xmlns:dsig="&dsig;">`
`count(ancestor-or-self::dsig:Signature | here()/ancestor::dsig:Signature[1]) >`
`count(ancestor-or-self::dsig:Signature)</XPath>`
 - » If the direct parent signature is in the set of all outer signatures, this element is excluded from signing



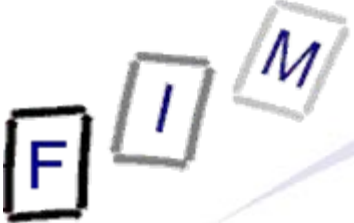
- Algorithms are identified by URIs
 - Some of them must be implemented (not used!), some are optional
- Digest: SHA1
- Encoding: Base64
- MAC: HMAC-SHA1
 - MAC=Message Authentication Code (=crypt. hash algorithm)
- Signature: DSAwithSHA1, RSAwithSHA1
- Canonicalization: Canonical XML 1.0 omitting comment/**with comments**, Canonical XML 1.1 omitting comment/**with comments**
- Transform: Enveloped signature, XPath, XSLT

Required Recommended Optional



- An API specification was created in JSR 105
 - We use the version from Apache (“Santuario” → Java and C++)
 - » Requires external library for the actual signature, digest, ... algorithms!
 - Here: BouncyCastle (Free implementation)
- Example:
 - Please note that the verification checks only the cryptographic quality of the signature!
 - » I.e. verification will succeed for **ANY** signature with **ANY** key!
 - Real application should check whether the certificate is something trusted/expected or use their own certificates
 - The resulting signed document is no longer valid!
 - » The signature (or any other extensions) is not specified in the schema

[Sign.java](#), [Verify.java](#)
[Order.xml](#), [Order_signed.xml](#)



- Encrypted can be:

- The whole XML document
- A single XML element
- XML element content: several (sub-)elements
- XML element content: character data

```
<elem><sub/>Text</elem>  
<elem><sub/>Text</elem>  
<elem><sub/>Text</elem>
```

- Encrypted data can again be encrypted without problem

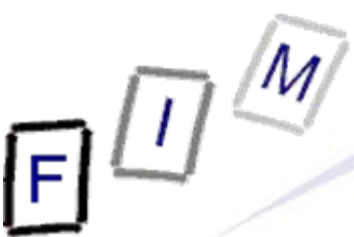
- Encrypted data is represented by the following information

- Encryption method: The algorithm used
- Key information: How to find the key for decryption or the key itself
 - » Symmetric encryption: The key itself (encrypted!)
 - » Asymmetric encryption: The public key used
 - » General: Name or pointer to the key to be used
- The enciphered data: Value or pointer to it
- Additional properties



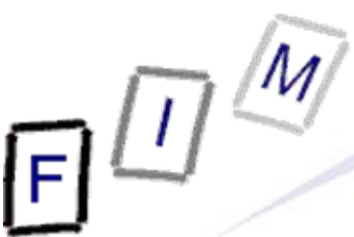
- Algorithms are identified by URIs
 - Some of them must be implemented (not used!), some are optional
- Block encryption: TripleDES, AES-128, AES-256, AES-192
- Stream encryption: None specified!
- Key transport: RSA-v1.5, RSA-OAEP
- Key agreement: Diffie-Hellman
- Symmetric key wrap: TripleDES, AES-128, AES-256, AES-192
- Message digest: SHA1, SHA256, SHA512, RIPEMD-160
- Message authentication: XML digital signature
- Canonicalization: (Exclusive) canonical; with(-out) comments
- Encoding: Base64
 - The encoded result is for almost all algorithms binary data!

Required Recommended Optional



- Currently there exists no API specification
 - One was under development by Sun in JSR 106 (Withdrawn 2010)
 - Implementation available from Apache (Java and C++; “Santuario”)
 - » Requires external library for the actual encryption, ... algorithms!
- Very simple to use
 - But take care of the problems (see next slide!)
 - » E.g. the encrypted order has some new namespace declarations!
- The real problem is often somewhere else: Key management!
 - Where to (**securely!**) store encryption/signature keys?
 - How to identify the key to use (certificates, public registries, ...)?

[Encrypt.java](#), [Decrypt.java](#)
[Order.xml](#), [Order_encrypted.xml](#), [Order_decrypted.xml](#)



- When namespaces are used, these may be inherited by the element which is to be encrypted
 - Or explicitly removed by specifying ' `xmlns:ns=""` '
- When this is encrypted and later decrypted and put into a different context, the result might be invalid!
 - With empty namespace even in the same context
 - » On canonicalization this might be stripped away, so after decryption the default namespace is inherited instead of removed!
- `xml:base`, `xml:lang`, `xml:space` attributes: May cause problems
 - These are also inherited!

The application must take care to specify these things explicitly or know exactly into which context to put the result of decryption!



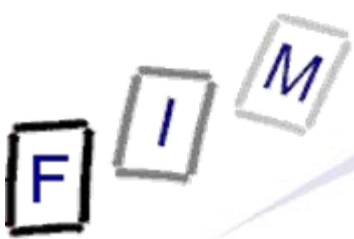
XML Signature + Encryption

- Both do not specify new algorithms
 - These must be acquired separately (patent problems, ...)!
- Combining both can lead to problems
 - Signing encrypted data: How to know what is really signed?
 - » Should be avoided; task of the application!
 - Encrypting signed data: How to know whether signature verification should be done before decryption or afterwards?
 - » If complete structure is encrypted \Rightarrow no problem
 - » When only subparts are encrypted, this gets important!
 - » Example: Signing the payment information and later on encrypting the creditcard number, but leaving the name in cleartext
 - » There exists a separate specification for this!
 - Introduces "exception" elements to the transformation

F I M

Questions?

Please ask immediately!



- W3C XML Security Working Group
<http://www.w3.org/2008/xmlsec/>
- XML Signature
<http://www.w3.org/Signature/>
- XML Encryption
<http://www.w3.org/Encryption/2001/>
- XML Canonicalization
<http://www.w3.org/TR/2001/REC-xml-c14n-20010315>
- Exclusive XML Canonicalization
<http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718>
- Apache Santuario
<http://santuario.apache.org/download.html>