Digital Signature with Hashing and XML Signature Patterns

Keiko Hashizume, Eduardo B. Fernandez, and Shihong Huang
Dept. of Computer Science and Engineering,
Florida Atlantic University
Boca Raton, FL 33431, USA,
ahashizu@fau.edu, ed@cse.fau.edu, shihong@cse.fau.edu

Abstract

Data security has become one of the most important concerns for organizations. Information is a valuable asset and needs to be protected. One important countermeasure against attackers is the use of digital signatures. Digital signatures provides message authentication and may also provide message integrity. We present here two patterns: XML Signature and Digital Signature with Hashing patterns. The XML Signature pattern, a specialization of the Digital Signature with Hashing, is used to secure XML messages.

1. Introduction

Data security has become one of the most important concerns among us, especially for organizations that have valuable information. An important security risk is that information can be modified during its transmission by somebody trying to make us believe something to his convenience. How do we prove that a message came from a specific user? Digital signatures use public-key cryptography to provide message authentication by proving that a message was sent indeed from a specific sender [dig, Sta06]. The sender encrypts the message using his private key to sign it. In this case, the signature has at least the same length as the message. This works but it wastes bandwidth and time. Thus, we need to reduce the length to the message before signing it. This can be done producing a digest through hashing. When the receiver gets the signed message, he verifies the signature by decrypting it using the sender's public key, thus proving that the message was encrypted by the sender. Also, digital signatures can provide message integrity by verifying whether a message was modified during its transmission. Digital signatures can also protect the integrity and verify the origin of a digital document, e.g. a certificate, or of programs. Digital signatures provide also *non-repudiation*, the sender cannot deny having sent the message he signed. In several countries, including the U.S., digital signatures have legal validity.

An emerging use of web services that exchanges XML messages also can be target of attacks. Some security standards have been developed to apply mechanisms that reduce security risks, one of these is. XML Signature. This standard is a joint effort between the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (ITEF). XML Signature defines how to digitally sign an entire XML message, part of an XML message, or an external object. XML Signature also includes hashing, but the pattern name follows the name of the standard. Because XML documents can have the same contents but in different layouts, we need to convert the documents into a canonical form before we apply digital signatures. Note that XML

Signature solves the same problem as the Digital Signature with Hashing pattern but in a more specialized context.

We present here two patterns: XML Signature and Digital Signature with Hashing patterns. The XML Signature pattern, a specialization of the Digital Signature with Hashing, is used to secure XML messages. We assume the reader is a designer intending to use message authentication in her design or a user intending to sign documents; we also assume both types of users have a basic knowledge of cryptography and UML. We provide a solution with sufficient detail so as it can be used as a guideline for design of signature systems and for users of signed documents.

Section 2 presents the Digital Signature with Hashing pattern, and Section 3 presents the XML Signature pattern.

2. Digital Signature with Hashing

2.1. Intent

Digital Signature with Hashing allows a principal to prove that a message was originated from it. It also provides message integrity by indicating whether a message was altered during transmission.

2.2. Example

Alice in the Sales department wants to send a product order to Bob in the production department. The product order does not contain sensitive data such as credit card number, so it is not important to keep it secret. However, Bob wants to be certain that the message was created by Alice so he can charge the order to her account. Also, because this order includes the quantity of items to be produced, an unauthorized modification to the order will make Bob manufacture the wrong quantity of items. Eve is a disgruntled employee who can intercept the messages and may want to do this kind of modification to hurt the company.

2.3. Context

People or systems often need to exchange documents or messages through insecure networks and need to prove their origin and integrity. Stored legal documents need to be kept without modification and indicating their origin. Software sent by a vendor through the Internet is required to prove its origin.

We assume that those exchanging documents have access to a public key system where a principal possesses a key pair: a private key that is secretly kept by the principal and a public key that is in a publicly-accessible repository. We assume that there is a mechanism for the generation of these key pairs and for the distribution of public keys; that is, a public key infrastructure (PKI).

2.4. Problem

In many applications we need to verify the origin of a message (message authentication). Since an impostor may assume the identity of a principal, how do we verify that a message came from a particular principal? Also, messages that travel through insecure channels can be captured and modified by attackers. How do we know that the message/document that we are receiving has not been modified?

The solution for these problems is affected by the following **forces**:

- For legal or business reasons we need to be able to verify who sent a particular message. Otherwise, we may not be sure of its origin and the sender may deny having sent it (repudiation).
- Messages may be altered during transmission, so we need to verify that the data is in its original form when it reaches its destination.
- The length of the signed message should not be significantly larger than the original message; otherwise we would waste time and bandwidth.
- Producing a signed message should not require a large computational power or take a long time.

2.5. Solution

Apply properties of public key cryptographic algorithms to messages in order to create a signature that will be unique for each sender [Sta06]. The message is first compressed (hashed) to a smaller size (digest), and then it is encrypted using the sender's private key. When the signed message arrives at its target, the receiver verifies the signature using the sender's public key to decrypt the message, if it produces a readable message, it could only have been sent by this sender. The receiver then generates the hashed digest of the received message and compares it to the received hashed digest: if it matches the message has not been altered.

This approach uses public key cryptography where one key is used for encryption and the other key for decryption. To produce a digital signature (SIG), we encrypt (E) the hash value of a message (H(M)) using the sender's private key (PrK): $SIG = E_{PrK} (H(M))$

We recover the hash value of the message (H(M)) by applying decryption function D to the signature (SIG) using the sender's public key (PuK). If this produces a legible message, we can be confident that the sender created the message because she is the only one who has access to her private key. Finally, we calculate the hash value of the message as $H(M) = D_{PuK}(SIG)$ .If this value is the same as the message digest obtained when the signature was decrypted, then we know that the message has not been modified.

It is clear that the sender and receiver should agree to use the same encryption and hashing algorithms.

Structure

Figure 1 describes the class diagram for the Digital Signature Pattern.

A **Principal** may be a process, a user, or an organization that is responsible for sending or receiving messages. This **Principal** may have the roles of **Sender** or **Receiver**. A Sender may send a plain Message and/or a SignedMessage to a receiver.

The **KeyPair** entity contains two keys: public and private, that belong to a **Principal**. The public key is registered and accessed through a repository, and the private key is kept secret by its owner. In a Public Key system, one key is normally used for encryption, while the other is used for decryption. **PublicKeyRepository** is a repository that contains public keys that can be available to anyone. The PublicKeyRepository may be located in the same local network as the principal or in an external network.

The **Signer** creates the **SignedMessage** that includes the **Signature** for a specific message. On the other side, the **Verifier** checks that the **Signature** within the SignedMessage corresponds to that message. The Signer and Verifier use the **DigestAlgorithm** and **SignatureAlgorithm** to create and verify a signature respectively. The DigestAlgorithm is a hash function that condenses a message to a fixed length called a *hash value* or message digest. The SignatureAlgorithm encrypts and decrypts messages using public/private key pairs.

Dynamics

We describe the dynamic aspects of the Digital Signature Pattern using sequence diagrams for the use cases sign a message and verify a signature.

Sign a message (Figure 2)*:*

Summary: A Sender wants to sign a message before sending it
Actors: A Sender
Precondition: A Sender has a public/private pair key
Description:

a) A Sender sends the message and its private key to the signer.
b) The Signer calculates the hash value of the message (digest) and returns it to the Signer.
c) The Signer encrypts the hash value using the sender's private key with the Signature Algorithm. The output of this calculation is the digital signature value.
d) The Signer creates the Signature object that contains the digital signature value.
e) The Signer creates the SignedMessage that contains the original message and the Signature.

Postcondition: A SignedMessage object has been created.

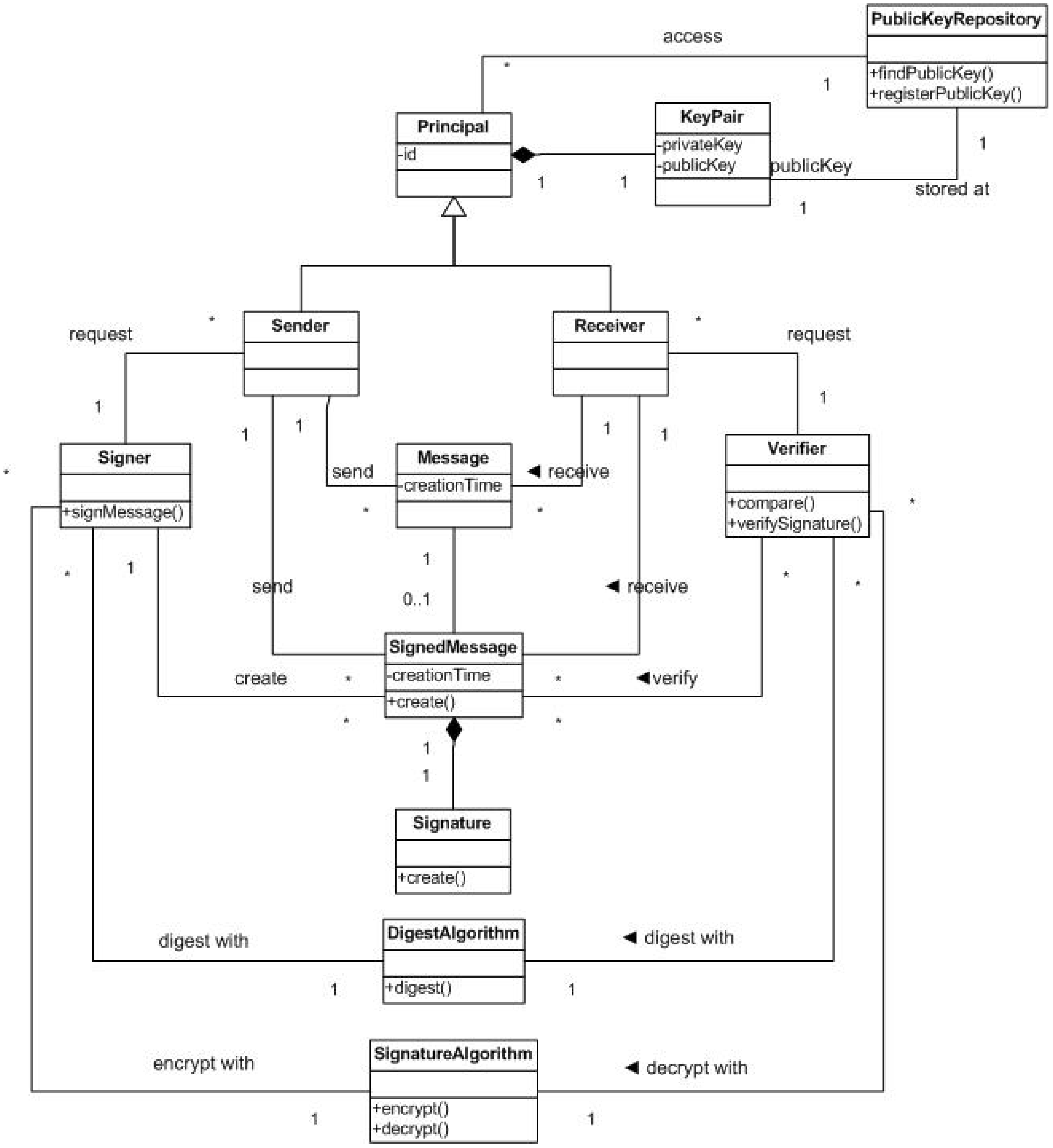

Figure 1: Class Diagram for Digital Signature Pattern

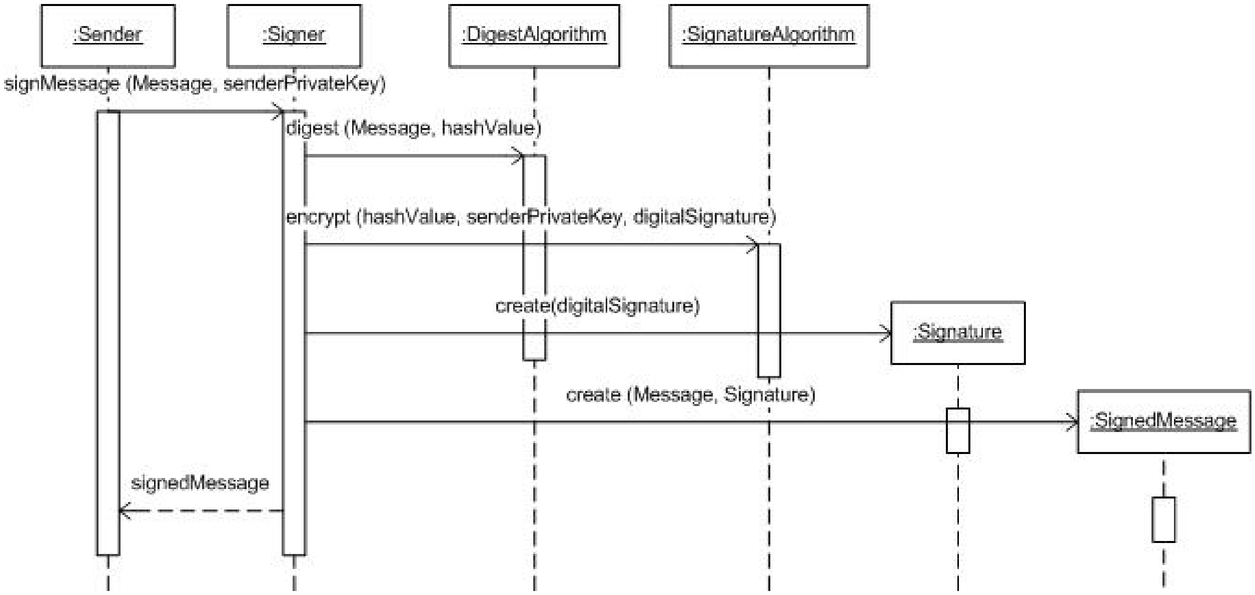
:Sender
:Signer
:DigestAlgorithm
:SignatureAlgorithm
signMessage (Message, senderPrivateKey)
digest (Message, hashValue)
encrypt (hashValue, senderPrivateKey, digitalSignature)
create(digitalSignature)
:Signature
create (Message, Signature)
:SignedMessage
signedMessage

Figure 2: Sequence Diagram for signing a message

Verify a Signature (Figure 3):

Summary: A receiver wants to verify that the signature corresponds to the received message.
Actors: A Receiver
Precondition: None
Description:

a) A Receiver retrieves the sender's public key from the repository.
b) A Receiver sends the signed message and the sender's public key to the verifier.
c) The verifier decrypts the signature using the sender's public key with the Signature Algorithm.
d) The verifier calculates the digest value of the message.
e) The verifier compares the outputs from step c) and d).
f) The verifier sends an acknowledgement to the receiver that the signature is valid.

Alternate Flows:

- The outputs from step c) and d) are not the same. Then, the verifier sends an acknowledgement to the receiver that the signature failed.

Postcondition: The signature has been verified.

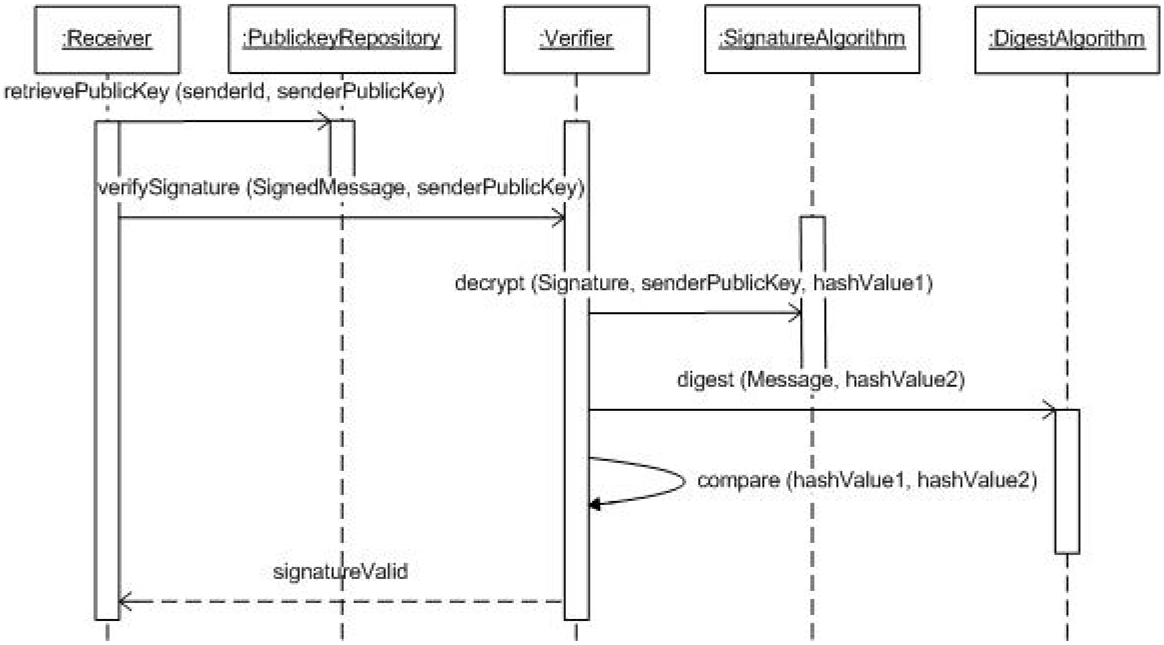
:Receiver
:PublickeyRepository
:Verifier
:SignatureAlgorithm
:DigestAlgorithm
retrievePublicKey (senderId, senderPublicKey)
verifySignature (SignedMessage, senderPublicKey)
decrypt (Signature, senderPublicKey, hashValue1)
digest (Message, hashValue2)
compare (hashValue1, hashValue2)
signatureValid

Figure 3: Sequence Diagram for verifying a signature

2.6. Implementation

- Use the Strategy Pattern [Gam94] to select different hashing and signature algorithms. The most widely used hashing algorithms are MD5 and SHA1. Those and others are discussed in [Sta06].
- A good hashing algorithm produces digests that are very unlikely produced by other meaningful messages, meaning that it is very hard for an attacker to create an altered message with the same hash value. The message digest should be encrypted after being signed to avoid man-in-the-middle attacks, where a person who captures a message could reconstruct its hash value.
- Two popular digital signature algorithms are RSA [RSA], and Digital Signature Algorithm (DSA) [Fed00, Sta06].
- The designer should choose strong and proven algorithms to prevent attackers from breaking them. The cryptographic protocol aspects, e.g. key generation, are as important as the algorithms used.
- The sender and receiver should have a way to agree on the hash and encryption algorithms used for a specific set of messages. XML documents indicate which algorithms they use and pre-agreements are not necessary.
- Access to the sender's public key should be available from a public directory or from certificates presented by the signer.
- Digital signatures can be implemented in different applications such as in email communication, distribution of documents over the Internet, or web services. For example, one can sign email's contents or any other document's content such as PDF. In both cases, the signature is appended to the email or document. When digital signatures are applied in web services, they are also embedded within XML messages. However,

these signatures are treated as XML elements, and they have additional features such as signing parts of a message or external resources which can be XML or any other data type.

- When certificates are used to provide the sender's public key, there must be a convenient way to verify that the certificate is still valid [SOA01].
- There should be a way to authenticate the signer software [dig]. An attacker who gains control of a user's computer could replace the signing software with his own software.

2.7. Known Uses

Digital Signatures have been widely used in different products.

- Adobe Reader and Acrobat [Ado05] have an extended security feature that allows users to digitally sign PDF documents.
- CoSign [Arx] digitally signs different types of documents, files, forms, and other electronic transactions.
- GNuPG [Gnu] digitally signs e-mail messages.
- The Java Cryptographic Architecture [Sun] includes APIs for digital signature.
- Microsoft .Net [Mic07] includes APIs for asymmetric cryptography such as digital signature.
- XML Signature [W3C08] is one of the foundation web services security standards that defines the structure and process of digital signatures in XML messages.

2.8. Consequences

This pattern presents the following advantages:

- Because a principal's private key is used to sign the message, the signature can be validated using its public key, which proves that the sender created and sent the message.
- When a signature is validated using a principal's public key, the sender cannot deny that he created and sent the message. If a message is signed using another private key that does not belong to the sender, the validity of the signature fails.
- If the proper precautions are followed (See 2.6), any change in the original message will produce a digest value that will be different (with a very high probability) from the value obtained after decrypting the signature using the sender's public key.
- A message is compressed into a fixed length string using the hash algorithm before it is signed. As a result, the process of signing is faster, and the signed message is much shorter.
- The available algorithms that can be used for digital signatures do not require very large amounts of computational power and do not take large amounts of time.

The pattern also has some (possible) liabilities:

- We need a well established Public Key Infrastructure that can provide reliable public keys. Certificates issued by some certification authority are the most common way to obtain this [Sta06].
- Both the sender and the receiver have to previously agree what signature and hashing algorithms they support. This is not necessary in XML documents because they are self-describing.
- Cryptographic algorithms create some overhead (time, memory, computational power), which can be reduced but not eliminated.
- The required storage and computational power may not be available, e.g. in mobile devices.
- Users must implement properly the signature protocol.
- There may be attacks against specific algorithms or implementations [dig]. These are difficult to use against careful implementations.
- This solution only allows one signer for the whole message. A variant or specialization, such as the XML Signature pattern, allows multiple signers.
- Digital signatures do not provide message authentication and replay attacks are possible [SOA01]. Nonces or time stamps could prevent this type of attacks.

2.9. Example Resolved

Alice and Bob agree on the use of a digital signature algorithm, and Bob has access to Alice's public key. Alice can then send a signed message to Bob. When the message is received by Bob, he verifies whether the signature is valid using Alice's public key and the agreed signature algorithm. If the signature is valid, Bob can be confident that the message was created by Alice. If the hash value is correct Bob also knows that Eve has not been able to modify the message.

2.10. Related Patterns

- Encryption/Decryption using public key cryptography [Bra00]
- Generation and Distribution of public keys [Leh02]
- Certificates [Mor06] are issued by a Certificate Authority (CA) that digitally signs them using its private key. A certificate carries a user's public key and allows anyone who has access to the CA's public key to verify that the certificate was signed by the CA.
- Strategy Pattern [Gam94], defines how to separate the implementation of related algorithms from the selection of one of them.

3. XML Signature

3.1. Intent

XML Signature allows a principal to prove that a message was originated from it. It also provides message integrity by defining whether a message was altered during transmission. The XML Signature standard [W3C08] describes the syntax and the process of generating and validating digital signatures for authenticating XML documents. XML Signature also provides message integrity. It requires canonicalization before hashing and signing.

3.2. Example

Alice in the Sales department wants to send product orders to Bob in the production department. The product orders are XML documents and do not contain sensitive data such as credit card number, so it is not important to keep them secret. Each order must be signed by Alice's supervisor Susie to indicate approval. Bob wants to be certain that the message was created by Alice so he can charge the order to her account and also needs to know that the orders are approved. Because the orders include the quantity of items to be produced, an unauthorized modification to an order will make Bob manufacture the wrong quantity of items. Eve can intercept the messages and may want to do this kind of modification.

3.3. Context

Users of web services send and receive SOAP messages through insecure networks such as the Internet and need to prove their origin and integrity. During their transmission these messages can be subject to a variety of attacks.

We assume that a principal possesses a key pair: a private key that is secretly kept by the principal and a public key that is in a publicly-accessible repository. We assume that there is a mechanism for the generation of these key pairs and for the distribution of public keys.

3.4. Problem

In many applications we need to verify the origin of a message (message authentication). Since an impostor may assume the identity of a principal, how do we verify that a message came from a particular principal? Also, messages that travel through insecure channels can be captured and modified by attackers. How do we know that the message/document that we are receiving has not been modified?

- For legal or business reasons we need to be able to verify who sent a particular message. Otherwise, we may not be sure of its origin and the sender may deny having sent it (repudiation). We assume the sender has signed the message to prove she is its author.
- Messages may be altered during transmission, so we need to verify that the data is in its original form when it reaches its destination.
- The length of the signed message should not be significantly larger than the original message; otherwise we would waste time and bandwidth.
- Producing a signed message should not require a large computational power or take a long time.
- We need to express a digital signature in a standardized XML format, so interoperability can be ensured between applications.
- There may be situations where we want to ensure proper origin or integrity in specific parts of a message. For example, an XML message can travel through many

intermediaries that add or subtract information, so if we sign the entire message, the signature would have no meaning. Thus, we should be able to sign portions of a message.

3.5. Solution

Apply cryptographic algorithms to messages in order to create a signature that will be unique for each message. First, the data to be signed may need to be transformed before applying any digest algorithm. The series of XML elements (that includes other subelements) is canonicalized before applying a signature algorithm. Canonicalization is a type of transform algorithm that converts data into a standard format, to remove differences due to layout formatting. This process is required because XML is a flexible language where a document can be represented in different ways that are semantically equal. Thus, after calculating the canonical form, both the sender and the receiver will sign and verify the same XML data respectively. After applying a canonicalization algorithm, the result value is digested and then encrypted using the sender's private key. Finally, the signature, in XML form, is embedded in the message.

In the other side, the receiver verifies the signature appended in the signed message. The verification process has two parts: reference verification and signature verification. In the reference verification, the verifier recalculates the digest value of the original data. This value is compared with the digest value included in the signature. If there is any mismatch, the verification fails. In the signature verification, the verifier calculates the canonical form of the signed XML element, and then applies the digest algorithm. This digest value is compared against the decrypted value of the signature. The decryption is done using the sender's public key.

There are three types of XML Signature: enveloped, enveloping and detached signature. In an enveloped signature, the signature is a child element of the signed data. For example, when you sign the entire XML message, the signature is embedded within the message. An enveloping signature is a signature where the signed data is a child of the signature. You can sign elements of a signature such as the Object or KeyInfo element. A detached signature is calculated over external network resources or over elements within the message. In the latter case, the signature is neither an enveloped nor an enveloping signature.

Structure

Figure 4 describes the structure of the XML Signature Pattern. Note that the upper part of this figure is almost the same as Figure 1. The main difference is that Figure 4 ads more details about the structure of the elements of the message so that signature can be applied more finely.

A **Principal** may be a process, a system, a user, or an organization that sends and receives **XMLMessages** and/or **SignedXmlMessages**. This principal may have the roles of **Sender** and **Receiver**.

Both an **XMLMessage** and a **SignedXMLMessage** are composed by XML elements, but this is only shown in the SignedXMLMessage. Each **XMLElement** may be a **SingleElement** that does not have any children or be a **Composite** element which is composed by other XML elements.

The **XMLSigner** and the **XMLVerifier** create and verify a **Signature,** respectively. They can select signature and digest algorithms. A **Signature** element is an XML element that has two required children: **SignedInfo** and **SignatureValue** and two optional children: **KeyInfo** and **Object**.

The **SignedInfo** element is the one that is actually signed. It contains one or more **Reference** elements, the canonicalization algorithm identifier, and the signature algorithm identifier. The Canonicalization algorithm is used to convert the SignedInfo element into a standard form before it is signed or verified. The Signature algorithm includes also a digest algorithm that is applied after calculating the canonical form of the Signed Info in both process creation and verification of XML signatures.

Each **Reference** element includes a Uniform Resource Identifier (URI), a hash value (DigestValue), the digest algorithm identifier (DigestMethod)**,** and an optional list of **Transform** elements. The URI is a pointer that identifies the data to be signed. It can point to an element inside an XML message, an element inside the Signature element such as Object or KeyInfo, or resources located in the Internet. The DigestValue contains a hash value after applying the digest algorithm to the data pointed by its URI. If the Transform element exists, it includes an ordered list of transform algorithms that are applied to the data before being digested.

The **SignatureValue** element includes the value of the digital signature.

If the **KeyInfo** is present, it indicates the information about the sender's public key that will be used to verify the signature. This flexible element may contain certificates, key names, and other public keys forms. Additional information about this element can be found in [W3C08].

The optional **Object** element may contain **SignatureProperties** and/or a **Manifest**. The **SignatureProperty** identifies properties of the signature itself such as the date/time when the signature was created. The **Manifest** element includes one or more Reference elements same as the **Reference** element within the SignedInfo. They are semantically equal; however, each Reference in the SignedInfo has to be validated in order to consider a valid signature. On the other hand, the list of Reference elements within the Manifest is validated.

The sender and receiver must use the same hash, signature, and canonicalization algorithms. XML documents are self-descriptive and indicate this information so the sender only needs to find the corresponding algorithms.

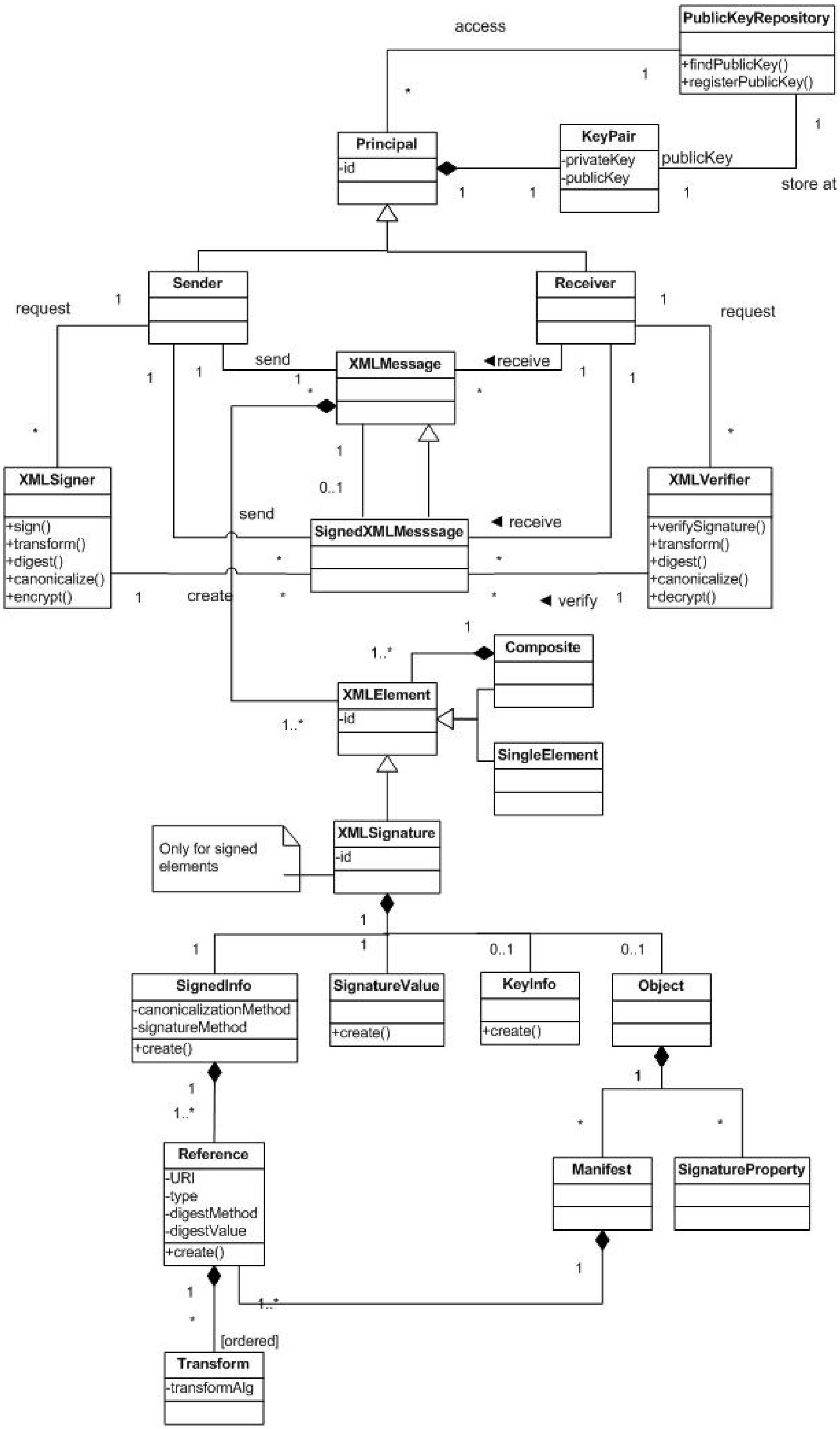

Figure 4: Class Diagram for XML Signature Pattern

Dynamics

We describe the dynamic aspects of the XML Signature Pattern using sequence diagrams for the use cases sign different XML elements of an XML message and verify an XML signature with multiple references.

Sign an XML message (Figure 5):

Summary: A sender wants to sign specified XML elements of an XML message.
Actors: A sender
Precondition: A sender has a private/public key pair.
Description:

g) A sender requests the signer to sign different XML elements of a message.
h) The signer calculates the digest value over the XML element.
i) The signer creates the <Reference> element including the digest value and using the digest algorithm.
j) Repeat steps b) and c) for each XML element to be signed.
k) The signer creates the <SignedInfo> that includes the Reference elements, the canonicalization algorithm identifier, and the signature algorithm identifier.
l) The signer applies the canonicalization algorithm to the <SignedInfo> element.
m) The signer signs the output from step f). First, it applies the digest algorithm, and then it encrypts the digest using the sender's public key. The output is the signature value.
n) The signer creates the <SignatureValue> element that includes the signature value.
o) The signer created the <KeyInfo> element that holds the sender's public key that will be used to verify the signature.
p) The signer creates the <Signature> element that includes the <SignedInfo>, the <SignatureValue>, and the <KeyInfo> elements.
q) The signer creates the SignedXMLMessage that includes the Signature and the XMLMessage.

Alternate Flows: None
Postcondition: The specified elements of the document have been signed

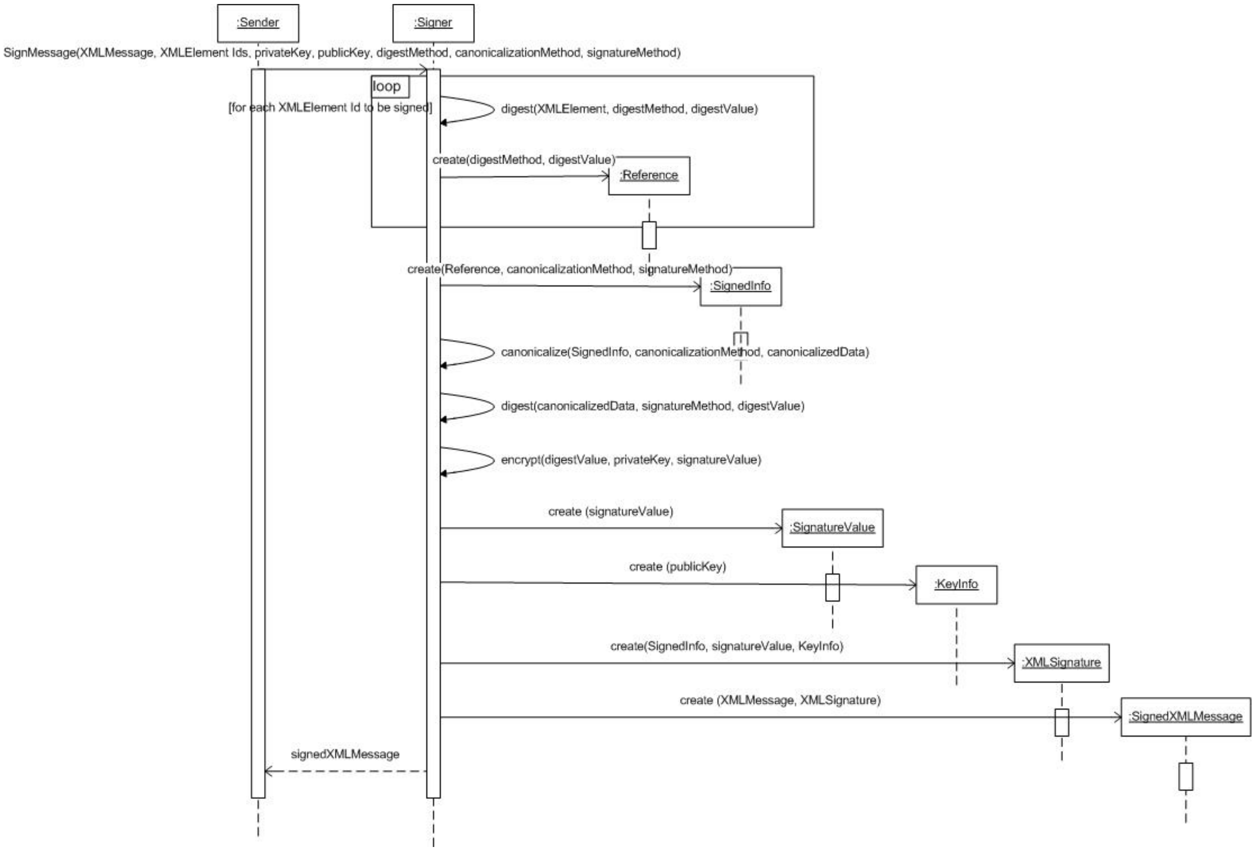
:Sender
:Signer
SignMessage(XMLMessage, XMLElement Ids, privateKey, publicKey, digestMethod, canonicalizationMethod, signatureMethod)
loop
[for each XMLElement Id to be signed]
digest(XMLElement, digestMethod, digestValue)
create(digestMethod, digestValue)
:Reference
create(Reference, canonicalizationMethod, signatureMethod)
:SignedInfo
canonicalize(SignedInfo, canonicalizationMethod, canonicalizedData)
digest(canonicalizedData, signatureMethod, digestValue)
encrypt(digestValue, privateKey, signatureValue)
create (signatureValue)
:SignatureValue
create (publicKey)
:KeyInfo
create(SignedInfo, signatureValue, KeyInfo)
:XMLSignature
create (XMLMessage, XMLSignature)
:SignedXMLMessage
signedXMLMessage

Figure 5: Sequence Diagram for signing an XML message

Verify an XML signature with multiple references (Figure 6):

Summary: A receiver wants to verify the signature of a received document.
Actors: A Receiver
Precondition: None
Description:

a) A receiver requests to verify the signature that is included in the SignedXMLMessage.
b) The verifier obtains the signature elements such as the <SignedInfo> which includes the <Reference> elements, the <SignatureValue>, and the <KeyInfo> elements.
c) The verifier calculates the digest value over the XML element that is pointed (URI) in the <Reference> element using the digest algorithm specified in the <Reference> element as well.
d) The verifier compares the output from step c) against the digest value specified in the Reference element.
e) Repeat step c) and d) for each <Reference> included in the <SignedInfo> element.
f) The verifier canonicalizes the <SignedInfo> element using the canonicalization method specified in the <SignedInfo>.
g) The verifier digests the output from step f) using the digest algorithm specified in the Signature Algorithm.
h) The verifier decrypts the signature value using the sender's public key (<KeyInfo>).
i) The verifier compares the outputs from step f) and h).

j) The verifier sends an acknowledgement to the receiver that the signature is valid.

Alternate Flows:

- If the values compared in step d) are not the same, then the signature is invalid.
- If the outputs in the step i) are not the same, then the validation fails.

Postcondition: The signature is validated.

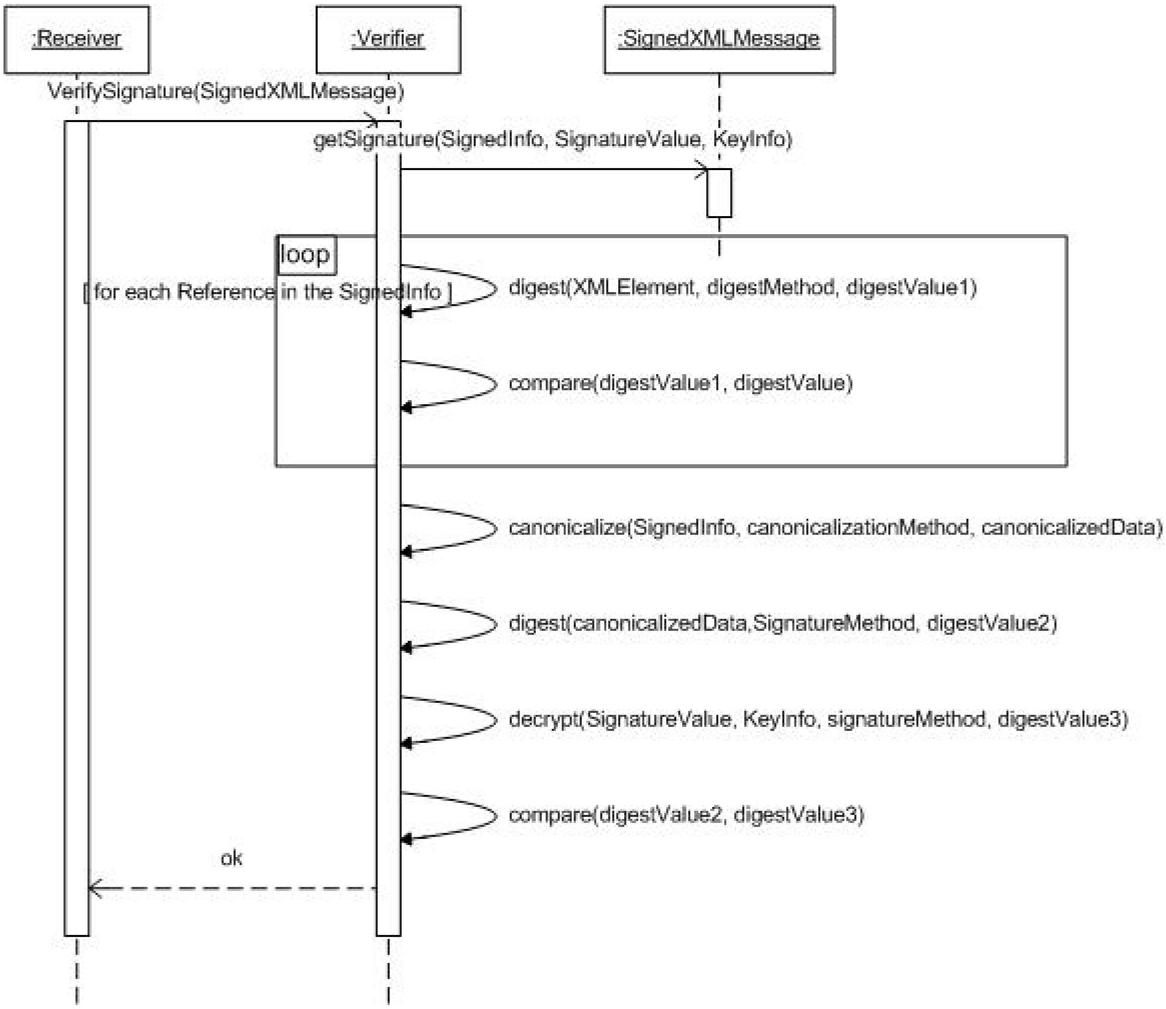
:Receiver
:Verifier
:SignedXMLMessage
VerifySignature(SignedXMLMessage)
getSignature(SignedInfo, SignatureValue, KeyInfo)
loop
[for each Reference in the SignedInfo]
digest(XMLElement, digestMethod, digestValue1)
compare(digestValue1, digestValue)
canonicalize(SignedInfo, canonicalizationMethod, canonicalizedData)
digest(canonicalizedData,SignatureMethod, digestValue2)
decrypt(SignatureValue, KeyInfo, signatureMethod, digestValue3)
compare(digestValue2, digestValue3)
ok

Figure 6: Sequence Diagram for verifying an XML signature

3.6. Implementation

- Identifiers of algorithms used to create a signature are attached along with the signature, so they also should be protected from being modified by attackers.
- XML documents may be parsed by different processors, and also XML allows some flexibility without changing the semantic of the message. Thus, we need to convert the data to be signed to a standard format.
- All the signers of a given document should have the same level of trust to avoid misleading the receivers about the trust of the whole message. Allowing untrusted signers might give them a better chance to attack the message.

- Use the Strategy Pattern [Gam94] to select different hashing and signature algorithms. The most widely used hashing algorithms are MD5 and SHA1. Two popular digital signature algorithms are RSA [RSA] and Digital Signature Algorithm (DSA) [Fed00].
- If needed the data to be signed needs to be transformed using transformation algorithms before producing a digest. For instance, if the object to be signed is an image, it needs to be converted into text.
- It is recommendable the use of certificates issued by an Certification Authority that are trusted by the sender and the receiver.

3.7. Known Uses

Several vendors have developed tools that support XML Signature.

- IBM - DataPower XML Security Gateway XS40 [IBM] parses, filters, validates schema, decrypts, verifies signatures, signs, and encrypts XML message flows.
- Xtradyne – Xtradyne's WS-DBC [Xtr]. The Web Services Domain Boundary Controller is an XML firewall that provides protection against malformed messages and malicious content, XML encryption, XML signature, and authentication, authorization, and audit.
- Forum Systems - Forum Sentry SOA Gateway [For] conforms to XML Digital Signature, XML Encryption, WS-Trust, WS-Policy and other standards.
- Microsoft .NET [Mic] includes API that support the creation and verification of XML digital signatures.
- Java XML Digital Signature API [Mul07] allows to generate and validate XML signatures

3.8. Consequences

This pattern presents the following advantages:

- A principal's private key is used to sign the message. The signature is validated using its public key, which proves that the sender created and sent the message.
- When a signature is validated using a principal's public key, the sender cannot deny that he created and sent the message. If a message is signed using another private key that does not belong to the sender, the validity of the signature fails.
- Any change in the original message will produce a digest value that will be different from the value obtained after decrypting the signature using the sender's public key.
- Before applying any signature algorithm, the data is compressed to a short fixed-length string. In XML Signature, digest algorithms are used two times; one is used to digest data to be signed indirectly, and the other digest algorithm is used to digest the canonical form of the SignedInfo element.
- Any change in the data that was indirectly signed will produce another digest that will invalidate the signature.
- The available algorithms that can be used for digital signatures do not require very large amounts of computational power and do not take large amounts of time.
- We can sign different parts of a message with different signatures. This allows a set of principals to write portions of a document and sign them.

- An XML signature is an XML element that is embedded in the message. The XML signature is composed of several XML elements that include information such as the value of the signature, the key that will be used to verify the signature, and algorithms used to compute the signature. This standard format helps XML parsers to better understand signature elements during the validation process.
- This pattern supports also message authentication code (MAC). Both signatures and MACs are syntactically identical. The difference between them is that signatures use public key cryptography while MAC uses a shared common key.
- The data being signed is pointed by its URI (Uniform Resource Identifier), so elements within XML messages and external network resources can be located using their identifiers.
- The SignedInfo is the element that is actually signed. It includes the references that point the data being signed along with their digest values, and algorithms identifiers. Thus, the XML signature also protects the algorithm identifiers from modification.
- XML Signature uses canonicalization algorithms to ensure that different representations of XML are transformed into a standard format before applying any signature algorithm.
- XML documents are self-describing and the sender and receiver don't need to agree in advance on the algorithms to be used.

The pattern also has some (possible) liabilities:

- We need a well established Public Key Infrastructure that can provide reliable public keys. Certificates issued by some certification authority are the most common way to obtain this [Sta06]. There is a public key standard for XML that should be used.
- Users must implement properly the signature protocol.
- There may be attacks against specific algorithms or implementations [dig]. These are difficult to use against careful implementations.
- Signing and verifying XML messages may create a significant overhead.
- The pattern does not describe the complete standard. For example, details of transforms and key values have been left out for simplicity [W3C08].

3.9 Example resolved

Alice and Susie sign each product order sent to Bob. Bob has access to Alice's and Susie's public keys. When the message is received by Bob, he verifies whether the signatures are valid using Alice's and Susie's public keys and the signature algorithm specified in the order. If the signature are valid, Bob can be confident that the message was created by Alice and approved by Susie. If the hash value is correct Bob also knows that Eve has not been able to modify the message.

3.10 Related Patterns

- This pattern is a specialization of the Digital Signature with Hashing Pattern.
- WS-Security Pattern [Has09] is a standard for securing XML messages using XML signature, XML Encryption, and security tokens.

The following specifications are related to XML Signature, but they have not been expressed as patterns.

- The XML Key Management Specification (XKMS) [W3C01] specifies the distribution and registration of public keys, which works together with the XML Signature.
- WS-SecurityPolicy [OAS07] standard describes how to express security policies such as what algorithms are supported by a web service or what parts of an incoming message need to be signed or encrypted.

4 Conclusions

We presented two patterns: Digital Signature with Hashing and XML Signature, the latter a specialization of the first one for a more specific context. Since the XML pattern solves the same problem it repeats the general aspects of the Digital Signature pattern but repeating this information allows the XML pattern to be used alone. We showed these two patterns together to make clearer the logic behind XML Signature, a rather complex pattern. Future work will include completing our development of other web services security patterns such as XML Encryption and WS-Security [Has08].

Acknowledgements

We thank our shepherd Amir Raveh, for his careful comments that improved the quality of this paper. The workshop participants at EuroPLoP 2009 also provided valuable comments. This work was supported by a grant from DISA, administered by Pragmatics, Inc. Our security research group provided useful comments.

References

[Ado] Adobe System Incorporated, Digital Signatures, http://www.adobe.com/security/digsig.html

[Arx] Arx, Digital Signature Solution (Standard Electronic Signatures), http://www.arx.com/products/cosign-digital-signatures.php

[Bra00] A. Braga, C. Rubira, and R. Dahab, "Tropyc: A pattern language for cryptographic object-oriented software", Chapter 16 in *Pattern Languages of Program Design 4* (N. Harrison, B. Foote, and H. Rohnert, Eds.). Also in *Procs. of PLoP'98*, http://jerry.cs.uiuc.edu/~plop/plop98/final_submissions/

[dig] Digital signature, http://en.wikipedia.org/wiki/Digital_signature

[Fed00] Federal Information Processing Standard, "Digital Signature Standard," 27 January 2000, http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf

[For] Forum Systems, Sentry: Messaging, Identity, and Security, http://www.forumsys.com/products/soagateway.php

[Gam94] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional, 1994

[Gnu] GnuPG, The GNU Privacy Guard, http://www.gnupg.org/

[Has09] K. Hashizume and E. B. Fernandez, "A Pattern for WS-Security", submitted for publication.

[IBM] IBM, WebSphere DatatPower XML Security Gateway XS40, http://www-01.ibm.com/software/integration/datapower/xs40/

[Leh02] S. Lehtonen and J. Parssinen. "A Pattern Language for Key Management," EuroPlop 2002. http://www.hillside.net/patterns/EuroPLoP2002/papers.html

[Mor06] P. Morrison and E.B.Fernandez, "The Credential pattern", Procs. of the Conference on Pattern Languages of Programs, PLoP 2006, Portland, OR, October 2006, http://hillside.net/plop/2006/

[Mic07] Microsoft Corporation, .NET Framework Class Library, November 2007, http://msdn.microsoft.com/en-us/library/ms229335.aspx

[Mul07] S. Mullan, Programming with the Java XML Digital Signature API, Sun Microsystems March 2007, http://java.sun.com/developer/technicalArticles/xml/dig_signature_api/

[OAS06] OASIS, Web Services Security: SOAP Message Security 1.1 (WS-Security 2004), 1 February 2006, http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

[OAS07] OASIS, W-S SecurityPolicy 1.2, 1 July 2007, http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.2/ws-securitypolicy.pdf

[RSA] RSA Security, PKCS #1: RSA Cryptography Standard, http://www.rsa.com/rsalabs/node.asp?id=2125

[SOA01] W3C, SOAP Security extensions: Digital Signature, W3C NOTE 06, February 2001, http://www.w3.org/TR/SOAP-dsig/

[Sta06] W. Stallings, *Cryptography and network security* (4th Ed.), Pearson Prentice Hall, 2006.

[Sun] Sun Microsystems Inc., Java SE Security, http://java.sun.com/javase/technologies/security/

[W3C01] W3C, XML Key Management Specification, March 2001 http://www.w3.org/TR/xkms/

[W3C08] W3C, XML Signature Syntax and Processing (Second Edition), 10 June 2008, http://www.w3.org/TR/xmldsig-core

[Xtr] Xtradyne, Xtradyne's WS-DBC - the XML/SOAP Firewall for Enterprises, http://www.xtradyne.de/products/ws-dbc/ws-dbc.htm