Symmetric Encryption and XML Encryption Patterns

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

Abstract

Most of the time information handled by organizations has been collected and processed by computers and transmitted across networks to other computers. How can we protect this information from unauthorized access? Encryption provides confidentiality by protecting sensitive information from being read by intruders. In this paper, we present two patterns: a Symmetric Encryption pattern that describes a basic type of algorithms and XML Encryption that describes how to apply symmetric and asymmetric encryption to XML messages.

Keywords: cryptography, security patterns, symmetric encryption, XML security

1. Introduction

Data security has become one of the most important concerns for governments, financial institutions, hospitals, and private businesses. An important security risk is that information can be captured and read during its transmission. How do we protect this information from being read by intruders? Encryption provides message confidentiality by transforming readable data (plain text) into an unreadable format (cipher text) that can be understood only by the intended receiver after a process called decryption The inverse function that the encrypted information readable again. There are two types of encryption: symmetric and asymmetric encryption. In symmetric encryption a common key is used for both encryption and decryption. In asymmetric encryption a public/private key pair is used for encryption/decryption; the sender encrypts the information using the receiver's public key, while the receiver uses his private key to decrypt the ciphered text.

The encrypted messages may be intercepted and be the object of attacks, including illegal reading, modification, and replay. An emerging use of web services that exchanges XML messages also can be target of attacks. Some security standards have been developed to apply correctly encryption functions and thus reduce security risks. XML Encryption is one of the basic standards in securing web services. XML Encryption defines how to encrypt/decrypt an entire XML message, part of an XML message, or an external object, and how to represent the encrypted content and information such as encryption algorithm and key in XML format.

Section 2 presents the Symmetric Encryption Pattern, and Section 3 presents the XML Encryption pattern. We assume the reader is a designer intending to use message secrecy in her design and has 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.

2. Symmetric Encryption

2.1. Intent

Encryption protects message confidentiality by making a message unreadable to those that do not have access to the key. Symmetric encryption uses the same key for encryption and decryption.

2.2 Example

Alice, in the Purchasing department regularly sends purchase orders to Bob in a distribution office. The purchase order contains sensitive data such as credit card numbers and other company information, so it is important to keep it secret. Eve can intercept her messages and may try to read them to get the confidential information.

2.3 Context

Applications that exchange sensitive information over insecure channels.

2.4 Problem

Applications that communicate with external applications interchange sensitive data that may be read by unauthorized users while they are in transit. How do we protect messages from being read by intruders?

The solution for this problem is affected by the following forces:

- *Confidentiality*--Messages may be captured while they are in transit, so we need to prevent unauthorized users from reading them by hiding the information of the message. Hiding the information also makes replaying of messages by an attacker harder to perform.
- *Reception*--The hidden information should be revealed conveniently to the receiver.
- *Protocol*--We need to apply the solution properly or it will not be able to stand attacks (there are several ways to attack a method to hide information.
- *Performance*--The time to hide and recover the message should be reasonable.

2.2. Solution

- so we need to prevent unauthorized users from reading them by hiding the information of the message using a symmetric cryptographic encryption.

Transform a message in such a way that only can be understood by the intended receiver after applying the reverse transformation using a valid key. The transformation process at the sender's end is called Encryption, while the reverse transformation process at the receiver's end is called Decryption.

The sender applies an encryption function (E) to the message (M) using a key (k); the output is the cipher text (C).

$$C = E_k (M)$$

When the cipher text (C) is delivered, the receiver applies a decryption function (D) to the cipher text using the same key (k) and recovers the message, i.e.

$$M = D_k (C)$$

Structure

Figure 1 describes the class diagram for the Symmetric Encryption Pattern.

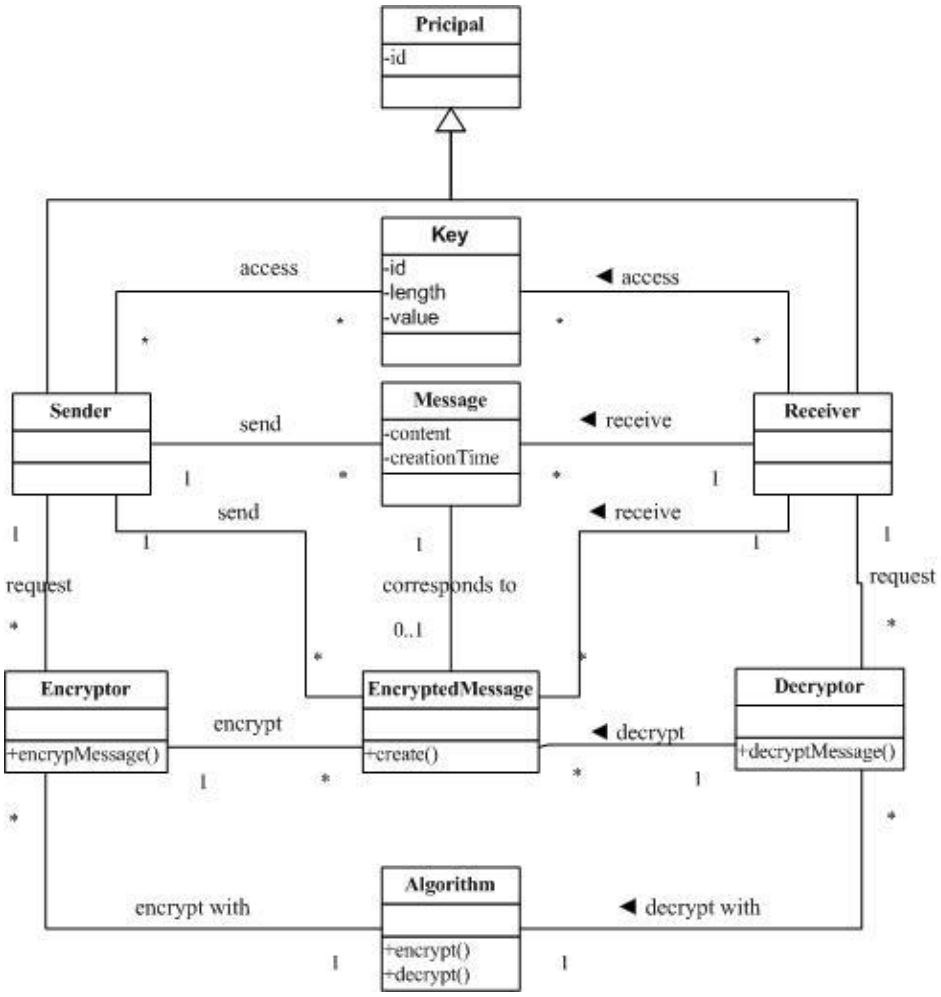

Fig1: Class Diagram for Symmetric Encryption Pattern

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 Message and/or a **EncryptedMessage** to a receiver with which it shares a secret **Key**.
The **Encryptor** creates the EncryptedMessage that contain the cipher text using the shared key, while the **Decryptor** deciphers the encrypted data into its original form using the same key. Both the Encryptor and Decriptor use the same **Algorithm** to encipher and decipher a message.

Dynamics

We describe the dynamic aspects of the Encryption Pattern using sequence diagrams for the following use cases: encrypt a message and decrypt a message.

Encrypt a message (Figure 2):

Summary: A Sender wants to encrypt a message
Actors: A Sender
Precondition: Both sender and receiver have a shared key and access to a repository of algorithms. The message has already been created by the sender.
Description:

a) A Sender sends the message, the shared key, and the algorithm identifier to the Encryptor.
b) The Encryptor ciphers the message using the algorithm specified by the sender.
c) The Encryptor creates the EncryptedMessage that includes the cipher text.

Postcondition: The message has been encrypted and sent to the sender.

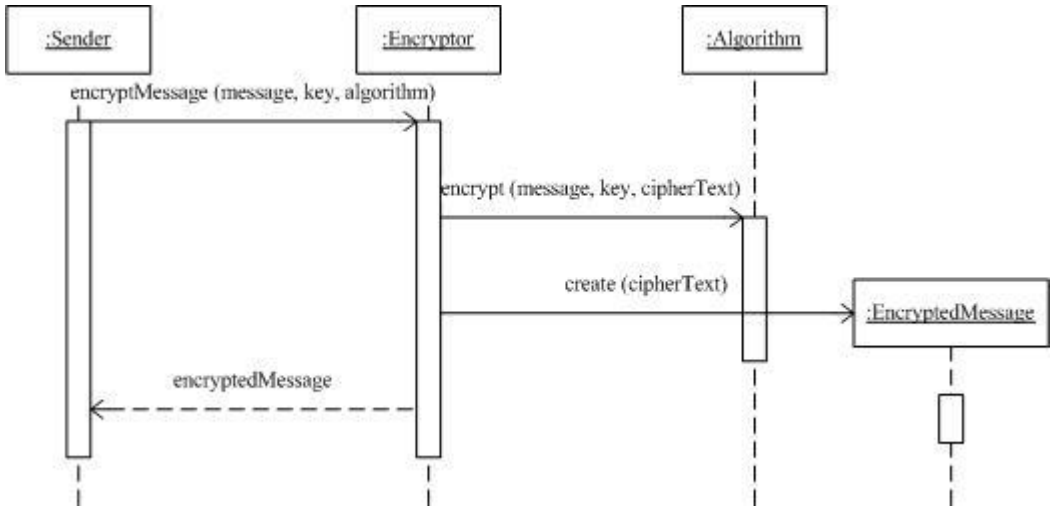

Figure 2: Sequence Diagram for Encrypting a Message

Decrypt an Encrypted Message (Figure 3):

Summary: A receiver wants to decrypt an encrypted message from a sender.

Actors: A Receiver
Precondition: Both the sender and receiver have a shared key and access to a repository of algorithms.
Description:
a) A Receiver sends the encrypted message and the shared key to the decryptor.
b) The Decryptor deciphers the encrypted message using the shared key.
c) The Decryptor creates the Message that contains the plain text obtained from the previous step.
d) The Decryptor sends the plain Message to the receiver.
Alternate Flows:
- If the key used in step b) is not the same as the one used for encryption, the decryption process fails.

Postcondition: The encrypted message has been deciphered and delivered to the Receiver.

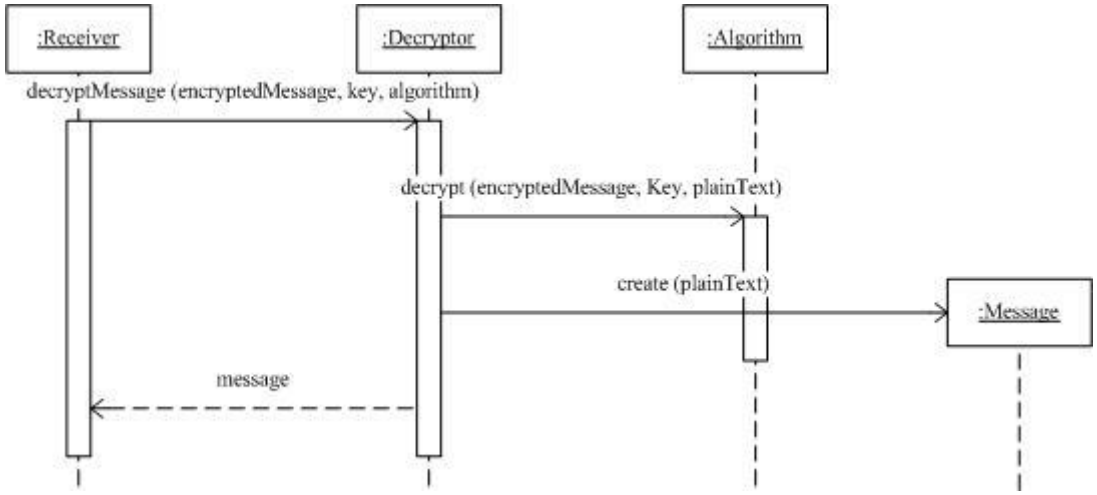

Figure 3: Sequence Diagram for Decrypting an Encrypted Message

2.3. Implementation

- Use the Strategy Pattern [Gam94] to select different encryption algorithms.
- The designer should choose well-known algorithms such as AES (Advanced Encryption Standard) [Fed01] and DES (Data Encryption Standard) [Fed99]. Books such as [] describe their features and criteria for selection.
- Encryption can be implemented in different applications such as in email communication, distribution of documents over the Internet, or web services. In these applications, we are able to encrypt the entire document. However, in web services we can encrypt parts of a message.
- Both the sender and the receiver have to previously agree what cryptographic algorithm they support.
- A good key generator is very important. It should generate keys that are as random as possible or an attacker who captures some messages could be able to deduce the key..

- A long encryption key should be used (at least 64 bits). Only brute force is known to work against the DES and AES for example; using a short key would let the attacker generate all possible keys.

2.4. Known Uses

Symmetric Encryption has been widely used in different products.

- GNuPG [Gnu] is free software that secures data from eavesdroppers.
- OpenSSL [Ope] is an open source toolkit that encrypts and decrypts files.
- Java Cryptographic Extension [Sun] provides a framework and implementations for encryption.
- The .NET framework [Mica] provides several classes to perform encryption and decryption using symmetric algorithms.
- XML Encryption [W3C02] is one of the foundation web services security standards that defines the structure and process of encryption for XML messages.
- Pretty Good Privacy (PGP), a set of programs used mostly for e-mail security, includes methods for symmetric encryption and decryption [PGP].

2.5. Consequences

This pattern presents the following advantages:

- Only receivers who possess the shared key can decrypt a message transforming it into a readable form. A captured message is unreadable to the attacker. This makes attacks bsed on replaying a message very hard.
- The strength of a cryptosystem is based on the secrecy of a long key []. The cryptographic algorithms are known to the public, so the key should be kept protected from unauthorized users.
- It is possible to select from several encryption algorithms the one suitable for the application needs.
- There exist encryption algorithms that take a reasonable time to encrypt messages.

The pattern also has some (possible) liabilities:

- This pattern assumes that the shared key was distributed in a secure way. This may not be easy for large groups of nodes exchanging messages.
- Cryptography operations are computationally intensive and may affect the performance of the application.
- Encryption does not provide data integrity. The encrypted data can be modified by an attacker, and the receiver would decrypt the cipher text to something else other than the original text.
- Encryption does not prevent a replay attack because an encrypted message can be captured and resent without being decrypted. It is recommended to use another security mechanism such as Timestamps or Nonce.

2.6 Example resolved

Alice, in the Purchasing department encrypts the purchase orders she sends to Bob. The purchase's order sensitive data is now unreadable to Eve. Eve can try to apply to it all possible keys but if the algorithm has been well implemented, she cannot read the confidential information.

2.7 Related Patterns

- Information Secrecy Pattern [Bra98], supports the encryption/decryption of data. This pattern describes encryption in more general terms. It does not distinguish between asymmetric and symmetric encryption.
- Strategy Pattern [Gam94], defines how to separate the implementation of related algorithms from the selection of one of them.

3. XML Encryption Pattern

3.1. Intent

XML Encryption standard [W3C02] describes the syntax to represent XML encrypted data and the process of encryption and decryption. XML Encryption provides message confidentiality by hiding sensitive information in such way that can be understood only by intended recipients.

3.2 Example

Alice, in the Purchasing department regularly sends purchase orders in the form of XML documents to Bob, who works in a distribution office. The purchase order contains sensitive data such as credit card numbers and other company information, so it is important to keep it secret. In the receiving end, different people will handle different parts of the order. Eve can intercept these orders and may try to read them to get the confidential information.

3.3 Context

Users of web services send and receive SOAP messages through insecure networks such as the Internet.

3.2. Problem

Applications that communicate with external applications or users interchange sensitive data that may be read by unauthorized people while the messages with this data are in transit.

The solution for this problem is affected by the following forces:

- Messages may be captured while they are in transit, so we need to prevent unauthorized users from reading them by hiding the information of the message using encryption.

- We need to express encrypted elements in a standardized XML format to allow encrypted data to be nested within an XML message. Otherwise, different applications cannot interoperate.
- Different parts of a message may be intended for different recipients, and not all the information contained within a message should be available to all the recipients. Thus, recipients should be able to read only those parts of the message that are intended for them.
- For flexibility reasons, both symmetric and asymmetric encryption algorithms should be supported.
- If a secret key is embedded in the message, it should be protected. Otherwise, an attacker could read some messages.

3.3. Solution

Transform a message using some algorithm so that it can only be understood by legitimate receivers that possess a valid key.
First, the data has to be serialized before encryption. The serialization process will convert the data into octets. Then, this serialized data is encrypted using the chosen algorithm and the encryption key. The cipher data and the information of the encryption (algorithm, key, and other properties) are represented in XML format.
XML Encryption supports both types of encryption: symmetric and asymmetric. The symmetric encryption algorithm uses a common key for both encryption and decryption. On the other hand, the asymmetric encryption algorithm uses a key pair (public key and private key). The sender encrypts a message using the receiver's public key, and the receiver uses its private key to decrypt the encrypted message. Thus, in both types of encryption, only recipients who possess the shared key or the private key that matches the public key used in the encryption process can read the encrypted message after decryption.

Structure

Figure 4 describes the structure of the XML Encryption Pattern.
A **Principal** may be a process, a system, a user, or an organization that sends and receives **XMLMessages** and/or **EncryptedXMLMessages**. This principal may have the roles of **Sender** and **Receiver**.

Both an **XMLMessage** and a **EncryptedMLMessage** are composed of XML elements. Each XMLElement may have many children, and each child also can be composed by other XML elements, and so on. The **Encryptor** and the **Decryptor** encipher a message and decipher an encrypted message respectively.

The **EncryptedData** contains other subelements such as the encryption method, key information, cipher value, and encryption properties. The **EncryptionMethod** is an optional element that specifies the algorithm used to encrypt the data. If this element is not specified, the receiver must know the encryption algorithm. The **KeyInfo** (optional) contains the same key information as the one describes in the XML Signature standard [W3C08]. However, this standard defines two other subelements: **EncryptedKey** and **ReferenceList**. The EncryptedKey

contains similar elements as the EncryptedData; however, they are not shown in the class diagram. The EncryptedKey includes an optional ReferenceList element that points to data or keys encrypted using this key. The **CipherData** is a mandatory element that stores either the cipher value or a pointer (cipher reference) where the encrypted data is located. The **EncryptionProperties** element holds information such as the time that the encryption was performed or the serial number of the hardware used for this process.

Dynamics:
We describe the dynamic aspects of the XML Encryption Pattern using sequence diagrams for the following use cases: "encrypt XML elements" and "decrypt an encrypted XML message".

Encrypt XML elements (Figure 5):

Summary: A sender wants to encrypt different elements of an XML message using a shared key.
Actors: A sender
Precondition: Both sender and receiver have a shared key and a list of encryption algorithms.
Description:

a) A sender requests to the encryptor to encrypt a list of XML elements. This list is represented with an asterisk (*) in the sequence diagram.
b) The encryptor creates the EncryptedXMLMessage.
c) The encryptor encrypts the XML Element using the shared key and the encryption method provided by the sender and produces an encrypted value.
d) The encryptor creates the EncryptionData element including the EncryptionMethod that holds the encryption algorithm used to encrypt the data, the KeyInfo that contains information about the key, and the CipherData obtained from step c)
e) The encryptor replaces the XML element with the encrypted data.
f) Repeat steps c) to e) for each XML element to encrypt.
g) The encryptor sends the EncrypteXMLMessage to the sender.

Alternate Flows: none
Postcondition: The encrypted XML message has been created.

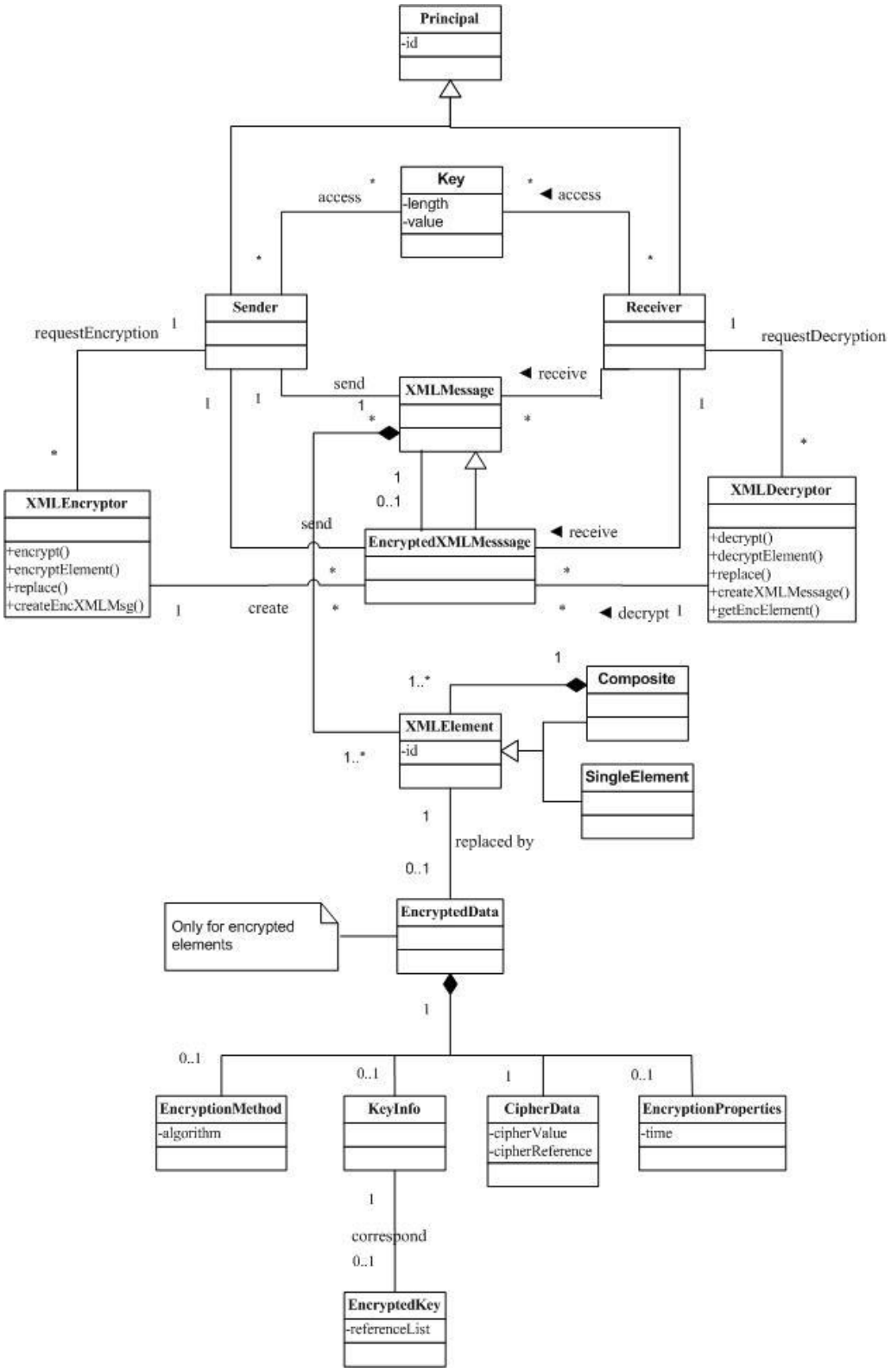

Fig4: Class Diagram for XML Encryption Pattern

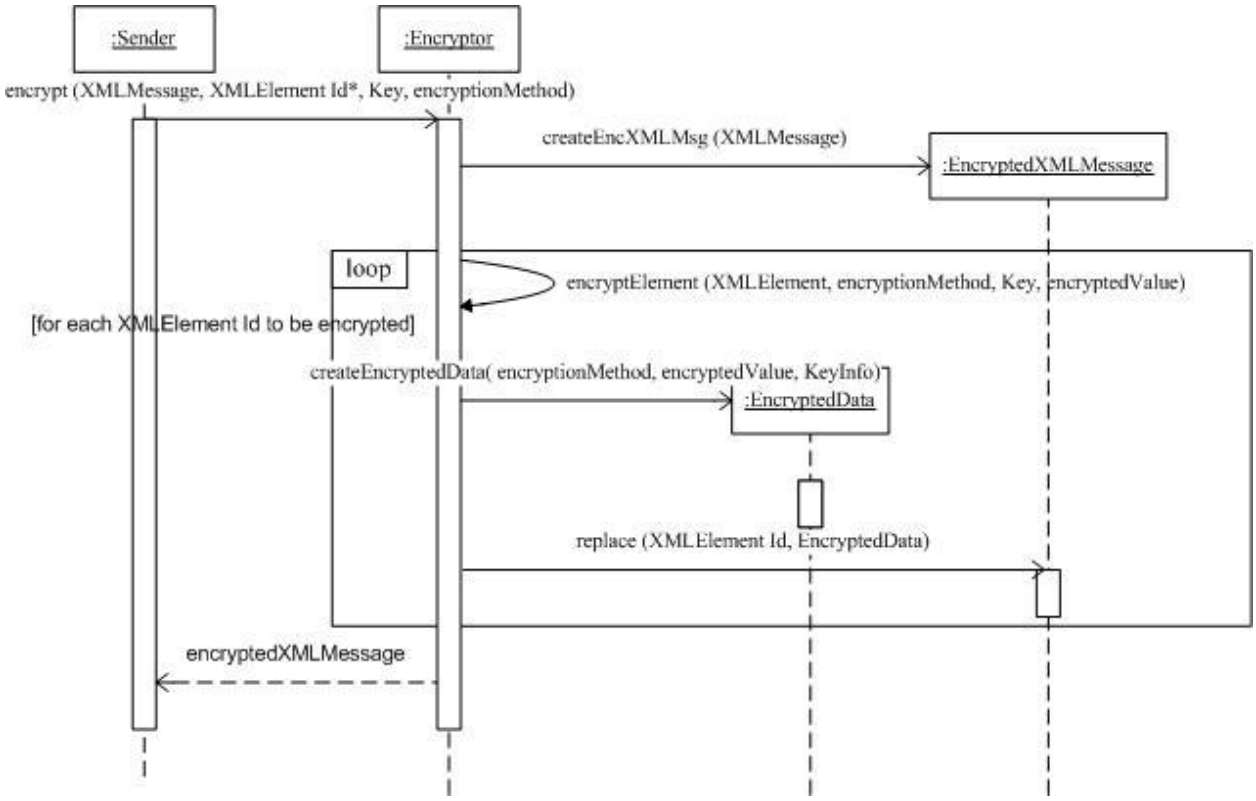

Figure 5: Sequence Diagram for encrypting XML Elements

Decrypt an Encrypted XML Message (Figure 6):

Summary: A receiver wants to decrypt an encrypted XML message.
Actors: A Receiver
Precondition: Both sender and receiver have a shared key and a list of encryption algorithms
Description:

a) A receiver requests to the verifier to decrypt an encrypted XML message.
b) The decryptor creates the XMLMessage that contains a copy of the EncryptedXMLMessage.
c) The decryptor obtains the elements within the EncryptedData element such as the EncryptionMethod, KeyInfo, and the cipherValue.
d) The encryptor decrypts the cipher value using the encryption method and the shared key.
e) The encryptor replaces the encrypted data with the plain text obtained from the previous step.
f) Repeat steps c) to e) for each XML element to decrypt.
g) The decryptor sends the decrypted XMLMessage to the receiver.

Alternate Flows:
If the key used in step d) is not the same as the one used in the encryption, then the decryption process fails.
Postcondition: The message has been decrypted.

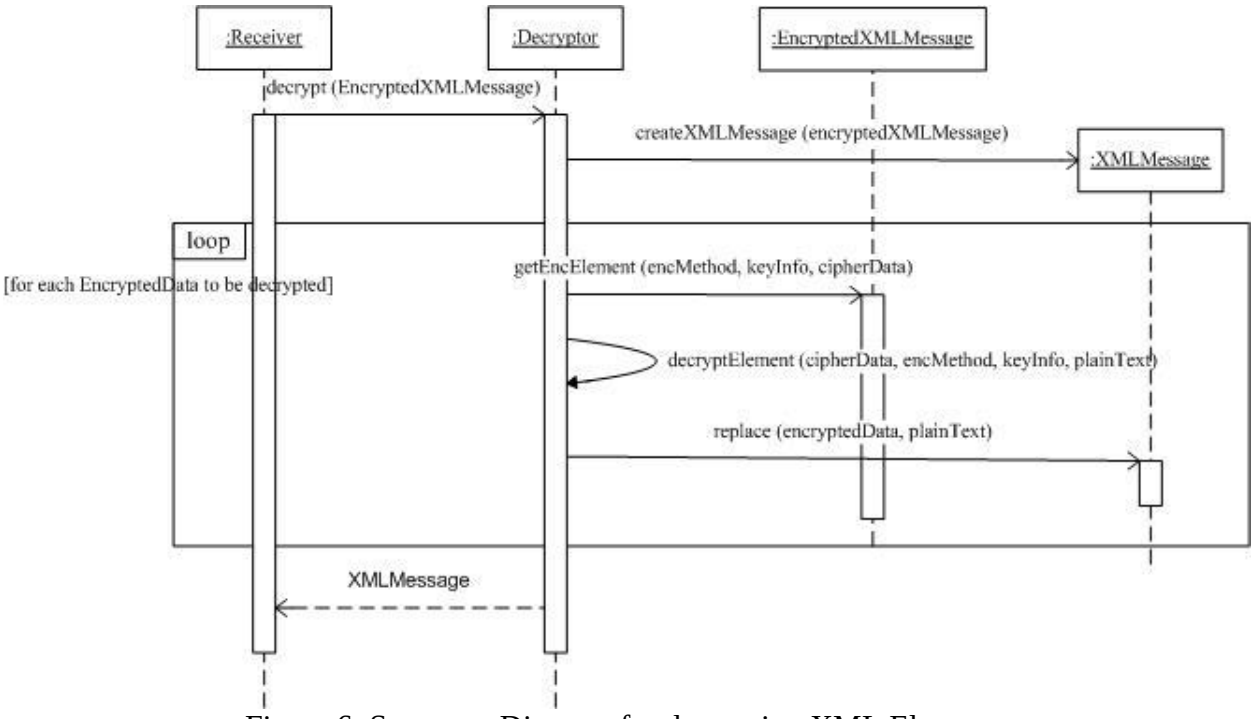

Figure 6: Sequence Diagram for decrypting XML Elements

3.4. Implementation

- The designer should choose strong encryption algorithm to prevent attackers from breaking them such as Advanced Encryption Standard (AES) and DES (Data Encryption Standard) for symmetric encryption, and RSA (Rivest, Shamir, and Adleman) for asymmetric encryption.
- Asymmetric encryption or public-key encryption is more computationally intensive than symmetric encryption. However, symmetric encryption requires that both sender and receiver share a common key. A better practice will be to use the asymmetric encryption in combination with the symmetric encryption. Use symmetric encryption for the data and asymmetric encryption for secure key distribution.
- XML Encryption supports both symmetric and asymmetric encryption. This provides application flexibility; for example, a session uses symmetric encryption and key distribution uses asymmetric encryption.

3.5. Known Uses

Several vendors have developed tools that support XML Encryption:

- Xtradyne's WebService Domain Boundary Controller (WS-DBC) [Xtr]. The WS-DBC is an XML firewall that provides protection against malformed messages and malicious content, XML encryption, XML signature, and authentication, authorization, and audit.
- IBM - DataPower XML Security Gateway XS40 [IBM] parses, filters, validates schema, decrypts, verifies signatures, signs, and encrypts XML message flows.

- 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 APIs that support the encryption and decryption of XML data.

3.6. Consequences

This pattern presents the following advantages:

- Only users that know the key can decrypt and read the message. Each recipient can only decrypt parts of a message that are intended for him but is unable to decrypt the rest.
- The EncryptedData is an XML element that replaces the data to be encrypted. The EncryptedData as well as the EncryptedKey are composed by other subelements such as encryption method, key information, and cipher value.
- The entire XML message or only some parts can be encrypted.
- If both the sender and the receiver have not exchanged the keys previously, the key can be sent in the message encrypted using public key system.

The pattern also has some (possible) liabilities:

- The structure is rather complex and users may get confused.

3.7. Related Patterns

- This pattern is a specialization of the Symmetric Encryption Pattern.
- WS-Security Pattern [Has09] is a standard for securing XML messages using XML signature, XML Encryption, and security tokens.
- Strategy Pattern [Gam94] defines how to separate the implementation of related algorithms from the selection of one of them.

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

- The XML Key Management Specification (XKMS) [W3C01] specifies the distribution and registration of public keys, and works together with XML Encryption.
- 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: Symmetric Encryption and XML Encryption, the latter a specialization of the first one. We showed these two patterns together to make clearer the logic behind XML Encryption, a rather complex pattern. Future work will include completing our development of other web services security patterns such as WS-Security [Has09], WS-Trust, WS-Federation, and WS-SecureConversations.

Acknowledgements

This work was supported by a grant from DISA, administered by Pragmatics, Inc. Our security research group provided useful comments.

References

[Bra98] 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/

[Fed99] Federal Information Processing Standards Publication, "Data Encryption Data (DES)," 25 October 1999, http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf

[Fed01] Federal Information Processing Standards Publication, "Advanced Encryption Standard," 26 November 2001, http://csrc.nist.gov/publications/fips/fips197/fips-197.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, "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

[Mica] Microsoft Corporation, .NET Framework Class Library, http://msdn.microsoft.com/en-us/library/e970bs09.aspx

[Micb] Microsoft Corporation, .NET Framework Class Library, http://msdn.microsoft.com/en-us/library/ms229749.aspx

[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

[Ope] The OpenSSL Project, OpenSSL, http://www.openssl.org/

[PGP] http://en.wikipedia.org/wiki/Pretty_Good_Privacy

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

[Sun] Sun Microsystems Inc., Java Cryptography Extension (JCE), http://java.sun.com/j2se/1.4.2/docs/guide/security/jce/JCERefGuide.html

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

[W3C02] W3C, XML Encryption Syntax and Processing, 10 December 2002, http://www.w3.org/TR/xmlenc-core/

[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