XMLSecurity DataFormat

XMLSecurity Data Format

The XMLSecurity Data Format facilitates encryption and decryption of XML payloads at the Document, Element, and Element Content levels (including simultaneous multi-node encryption/decryption using XPath). To sign messages using the XML Signature specification, please see the Camel XML Security component.

The encryption capability is based on formats supported using the Apache XML Security (Santuario) project. Symmetric encryption/decryption is currently supported using Triple-DES and AES (128, 192, and 256) encryption formats. Additional formats can be easily added later as needed. This capability allows Camel users to encrypt/decrypt payloads while being dispatched or received along a route.

Available as of Camel 2.9
The XMLSecurity Data Format supports asymmetric key encryption. In this encryption model a symmetric key is generated and used to perform XML content encryption or decryption. This "content encryption key" is then itself encrypted using an asymmetric encryption algorithm that leverages the recipient's public key as the "key encryption key". Use of an asymmetric key encryption algorithm ensures that only the holder of the recipient's private key can access the generated symmetric encryption key. Thus, only the private key holder can decode the message. The XMLSecurity Data Format handles all of the logic required to encrypt and decrypt the message content and encryption key(s) using asymmetric key encryption.

The XMLSecurity Data Format also has improved support for namespaces when processing the XPath queries that select content for encryption. A namespace definition mapping can be included as part of the data format configuration. This enables true namespace matching, even if the prefix values in the XPath query and the target xml document are not equivalent strings.

Basic Options

Option	Default	Description
`secureTag`	`null`	The XPath reference to the XML Element selected for encryption/decryption. If no tag is specified, the entire payload is encrypted/decrypted.
`secureTagContents`	`false`	A boolean value to specify whether the XML Element is to be encrypted or the contents of the XML Element • `false` = Element Level • `true` = Element Content Level
`passPhrase`	`null`	A String used as passPhrase to encrypt/decrypt content. The passPhrase has to be provided. If no passPhrase is specified, a default passPhrase is used. The passPhrase needs to be put together in conjunction with the appropriate encryption algorithm. For example using `TRIPLEDES` the passPhase can be a `"Only another 24 Byte key"`
`xmlCipherAlgorithm`	`TRIPLEDES`	The cipher algorithm to be used for encryption/decryption of the XML message content. The available choices are: • `XMLCipher.TRIPLEDES` • `XMLCipher.AES_128` • `XMLCipher.AES_128_GCM` **Camel 2.12** • `XMLCipher.AES_192` • `XMLCipher.AES_192_GCM` **Camel 2.12** • `XMLCipher.AES_256` • `XMLCipher.AES_256_GCM` **Camel 2.12** • XMLCipher.SEED_128 **Camel 2.15** • XMLCipher.CAMELLIA_128, XMLCipher.CAMELLIA_192, XMLCipher.CAMELLIA_256 **Camel 2.15**
`namespaces`	`null`	A map of namespace values indexed by prefix. The index values must match the prefixes used in the `secureTag` XPath query.

Asymmetric Encryption Options

These options can be applied in addition to relevant the Basic options to use asymmetric key encryption.

Option	Default	Description
`recipientKeyAlias`	`null`	The key alias to be used when retrieving the recipient's public or private key from a KeyStore when performing asymmetric key encryption or decryption.
`keyCipherAlgorithm`	**Camel 2.12** `XMLCipher.RSA_OAEP`	The cipher algorithm to be used for encryption/decryption of the asymmetric key. The available choices are: • `XMLCipher.RSA_v1dot5` • `XMLCipher.RSA_OAEP` • `XMLCipher.RSA_OAEP_11`

`keyOrTrustStoreParameters`	`null`	Configuration options for creating and loading a KeyStore instance that represents the sender's trustStore or recipient's keyStore.
`keyPassword`	`null`	**Camel 2.10.2 / 2.11:** The password to be used for retrieving the private key from the KeyStore. This key is used for asymmetric decryption.
`digestAlgorithm`	XMLCipher.SHA1	**Camel 2.12** The digest algorithm to use with the RSA OAEP algorithm. The available choices are: • `XMLCipher.SHA1` • `XMLCipher.SHA256` • `XMLCipher.SHA512`
`mgfAlgorithm`	EncryptionConstants.MGF1_SHA1	**Camel 2.12** The MGF Algorithm to use with the RSA OAEP algorithm. The available choices are: • `EncryptionConstants.MGF1_SHA1` • `EncryptionConstants.MGF1_SHA256` • `EncryptionConstants.MGF1_SHA512`
addKeyValueForEncryptedKey	true	**Camel 2.14.1** Whether to add the public key used to encrypt the session key as a KeyValue in the EncryptedKey structure or not.

Key Cipher Algorithm

As of Camel 2.12.0, the default Key Cipher Algorithm is now XMLCipher.RSA_OAEP instead of XMLCipher.RSA_v1dot5. Usage of XMLCipher.RSA_v1dot5 is discouraged due to various attacks. Requests that use RSA v1.5 as the key cipher algorithm will be rejected unless it has been explicitly configured as the key cipher algorithm.

Marshal

In order to encrypt the payload, the `marshal` processor needs to be applied on the route followed by the **`secureXML()`** tag.

Unmarshal

In order to decrypt the payload, the `unmarshal` processor needs to be applied on the route followed by the **`secureXML()`** tag.

Examples

Given below are several examples of how marshalling could be performed at the Document, Element, and Content levels.

Full Payload encryption/decryption

```
from("direct:start")
    .marshal().secureXML()
    .unmarshal().secureXML()
    .to("direct:end");
```

Partial Payload Content Only encryption/decryption

```
String tagXPATH = "//cheesesites/italy/cheese";
boolean secureTagContent = true;
...
from("direct:start")
    .marshal().secureXML(tagXPATH, secureTagContent)
    .unmarshal().secureXML(tagXPATH, secureTagContent)
    .to("direct:end");
```

Partial Multi Node Payload Content Only encryption/decryption

```
String tagXPATH = "//cheesesites/*/cheese";
boolean secureTagContent = true;
...
from("direct:start")
    .marshal().secureXML(tagXPATH, secureTagContent)
    .unmarshal().secureXML(tagXPATH, secureTagContent)
    .to("direct:end");
```

Partial Payload Content Only encryption/decryption with choice of passPhrase(password)

```
String tagXPATH = "//cheesesites/italy/cheese";
boolean secureTagContent = true;
...
String passPhrase = "Just another 24 Byte key";
from("direct:start")
    .marshal().secureXML(tagXPATH, secureTagContent, passPhrase)
    .unmarshal().secureXML(tagXPATH, secureTagContent, passPhrase)
    .to("direct:end");
```

Partial Payload Content Only encryption/decryption with passPhrase(password) and Algorithm

```
import org.apache.xml.security.encryption.XMLCipher;
....
String tagXPATH = "//cheesesites/italy/cheese";
boolean secureTagContent = true;
String passPhrase = "Just another 24 Byte key";
String algorithm= XMLCipher.TRIPLEDES;
from("direct:start")
    .marshal().secureXML(tagXPATH, secureTagContent, passPhrase, algorithm)
    .unmarshal().secureXML(tagXPATH, secureTagContent, passPhrase, algorithm)
    .to("direct:end");
```

Partial Payload Content with Namespace support

Java DSL

```
final Map<String, String> namespaces = new HashMap<String, String>();
namespaces.put("cust", "http://cheese.xmlsecurity.camel.apache.org/");

final KeyStoreParameters tsParameters = new KeyStoreParameters();
tsParameters.setPassword("password");
tsParameters.setResource("sender.ts");

context.addRoutes(new RouteBuilder() {
    public void configure() {
        from("direct:start")
           .marshal().secureXML("//cust:cheesesites/italy", namespaces, true, "recipient",
                                testCypherAlgorithm, XMLCipher.RSA_v1dot5, tsParameters)
           .to("mock:encrypted");
    }
}
```

Spring XML

A namespace prefix that is defined as part of the `camelContext` definition can be re-used in context within the data format `secureTag` attribute of the `secureXML` element.

```
<camelContext id="springXmlSecurityDataFormatTestCamelContext"
              xmlns="http://camel.apache.org/schema/spring"
              xmlns:cheese="http://cheese.xmlsecurity.camel.apache.org/">
    <route>
        <from uri="direct://start"/>
            <marshal>
                <secureXML secureTag="//cheese:cheesesites/italy"
                           secureTagContents="true"/>
            </marshal>
            ...
```

Asymmetric Key Encryption

Spring XML Sender

```
<!--  trust store configuration -->
<camel:keyStoreParameters id="trustStoreParams" resource="./sender.ts" password="password"/>

<camelContext id="springXmlSecurityDataFormatTestCamelContext"
              xmlns="http://camel.apache.org/schema/spring"
              xmlns:cheese="http://cheese.xmlsecurity.camel.apache.org/">
    <route>
        <from uri="direct://start"/>
            <marshal>
                <secureXML secureTag="//cheese:cheesesites/italy"
                           secureTagContents="true"
                           xmlCipherAlgorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"
                           keyCipherAlgorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"
                           recipientKeyAlias="recipient"
                           keyOrTrustStoreParametersId="trustStoreParams"/>
            </marshal>
            ...
```

Spring XML Recipient

```
<!--  key store configuration -->
<camel:keyStoreParameters id="keyStoreParams" resource="./recipient.ks" password="password" />

<camelContext id="springXmlSecurityDataFormatTestCamelContext"
              xmlns="http://camel.apache.org/schema/spring"
              xmlns:cheese="http://cheese.xmlsecurity.camel.apache.org/">
    <route>
        <from uri="direct://encrypted"/>
            <unmarshal>
                <secureXML secureTag="//cheese:cheesesites/italy"
                           secureTagContents="true"
                           xmlCipherAlgorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"
                           keyCipherAlgorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"
                           recipientKeyAlias="recipient"
                           keyOrTrustStoreParametersId="keyStoreParams"
                           keyPassword="privateKeyPassword" />
            </unmarshal>
            ...
```

Dependencies

This data format is provided within the **camel-xmlsecurity** component.