July 30, 2007

RT0745
Network; Security 8 pages

Research Report

Easy-To-Use Programming Model for Web Services Security

Yumi Yamaguchi, Hyen-Vui Chung, Masayoshi Teraguchi, and Naohiko Uramoto

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

Easy-To-Use Programming Model for Web Services Security

Yumi Yamaguchi[1], Hyen-Vui Chung[2], Masayoshi Teraguchi[1], and Naohiko Uramoto[1]
[1]*Tokyo Research Laboratory, IBM Research*
1623-14, Shimo-tsuruma, Yamato-shi, Kanagawa, 242-8502, Japan
[2] *IBM Software Group*
11501 Burnet Road, Austin, TX 78758-3415, USA
{yyumi, teraguti, uramoto}@jp.ibm.com, hychung@us.ibm.com

Abstract

Even with a support tool, setting up a Web Services Security (WS-Security) configuration can be difficult for people who are not familiar with WS-Security. Some of the reasons are that the WS-Security is a very rich specification with many options and often the processing is complicated. This paper introduces an application programming model (WSSAPI) to simplify the programming experience for end users. It was designed by looking at WS-Security processing from an abstract level. Also, it is designed to consider correctness, efficiency, usability, flexibility, portability, and extensibility. End users just follow the six-step programming model provided in WSSAPI to configure WS-Security. The comparison of WSSAPI to others, like WSS4J, WSE, and JSR-105 shows that it is much easier for end users to use WSSAPI. In this paper, existing APIs for WS-Security are reviewed and compared with WSSAPI to evaluate the ease of use and utility.

1. Introduction

The Service-Oriented Architecture (SOA) using Web Services is emerging as a framework that enables the creation of applications that loosely couple services from various systems. When services are composed together, it is important to consider not only the functional requirements but also the nonfunctional requirements, such as security, reliability, and performance. Especially for security, the security requirements for an application are commonly described in a policy like WS-SecurityPolicy [1], distributed to other applications, and converted to an internal security configuration model used by the WS-Security [2] implementation of each application. But in general, different WS-Security implementation requires different security configuration models. In order to convert the requirements to the configuration model, it's not enough to use only the policy because it just includes abstract requirements, such as what part of the message is signed or encrypted and what token is used. To fill the gap between the policy and the configuration model, more concrete mappings from the policy to the model, such as what key store is used, are required. However, it forces end users to configure both the policy and the mappings based on their WS-Security implementation even as they develop and test their secure application, requiring much time and effort.

On the other hand, rapid prototyping and testing, as typified by Ruby on Rails [3], has recently become a focus. For rapid prototyping, it is necessary to provide an easy-to-use and understandable API and avoid unnecessary settings by using as many default values as possible. If this approach can be adopted in the WS-Security configuration, the utility will be drastically improved.

In this paper, we propose an API-based design approach that allows non-security experts to easily configure and enable WS-Security. McManus showed considerations for designing APIs that are completely correct for their functions, easy to use, easy to learn, sufficiently fast, and small [4]. In other words, it is important to design a simple and easy-to-understand API. Our design approach mainly focuses on abstraction of the WS-Security processing. Other APIs for setting up WS-Security tend to be complicated because their programming models do not sufficiently consider efficiency and also because the WS-Security processing is complicated. Abstraction of the security processing leads us to a simple API. In addition, we consider the efficiency and ease of use while designing the API. WSSAPI, our new API provides a six-step programming model to apply WS-Security. End users just follow the programming model to configure and test WS-Security easily.

The remainder of this paper is organized as follows: Section 2 describes the features of existing APIs for WS-Security. Section 3 introduces our design API, WSSAPI. Section 4 shows an evaluation of our proposed API against competing approaches. Section 5 concludes the paper.

2. Related Work

The OASIS WS-Security specification [2] consists of digital signatures based on W3C XML Digital Signature [5], encryption based on W3C XML Encryption [6], and methods for security token attachment. Figure 1 shows the WS-Security processing. The processing needs a WS-Security configuration as well as keys and certificates.

Several APIs have been proposed to support setting up a WS-Security configuration. WSS4J [7], the WS-Security implementation, was published by the Apache Software Foundation. Microsoft has released Web Services Enhancements (WSE) [8], which is an add-on for Microsoft .NET, which supports not only WS-Security but also WS-Security-related specifications. Each provides its own API for setting up the WS-Security configuration. There are two APIs for XML Security that has been proposed as Java Specification Requests (JSR): the XML Digital Signature API (JSR-105) [9] and the XML Encryption API (JSR-106) [10]. JSR-105 has already been released, but JSR-106 is at the stage of public review. This section describes the features of these APIs and shows sample code for a signature for each of them. One observation from the design of these APIs is that most of them are centered on the specifications. End users are required to have certain amount of knowledge, such as knowledge of the W3C XML Digital Signature specification [5], the W3C XML Encryption Specification [6], the OASIS WS-Security Specification [2], or other knowledge to make use of these APIs.

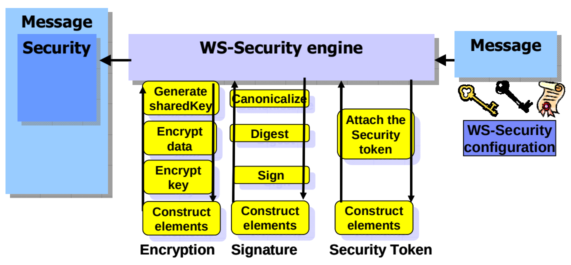

Figure 1 Process flow of WS-Security

2.1. JSR-105, JSR-106

JSR-105 and JSR-106 define APIs for the W3C XML Digital Signature and XML Encryption specifications [5, 6] respectively. These standards were basically designed from the perspective of the structure of messages secured with XML Digital Signature and XML Encryption specifications. End users need to construct all parts of the XML signature and XML encryption elements in the SOAP header by calling various methods in ascending order from the leaf element to the root element. Figure 2 shows a message construction flow in JSR-105. This complexity makes it difficult for people unfamiliar with the XML Digital Signature, XML Encryption, and WS-Security specifications to use them. Figure 3 shows sample code to sign a message using JSR-105. First, it generates an XMLSignContext to store the key. The XMLSignContext corresponds to the security token. The sample code assumes that the key for the signature is given in advance, but the end user actually needs to retrieve it from a key store file. Next the code constructs a template structure of the DOM tree with all of the necessary properties such as a signature method. Finally, the method XMLSignature.sign(XMLSignContext) calculates the digest and signature values, completes the DOM tree, and returns it.

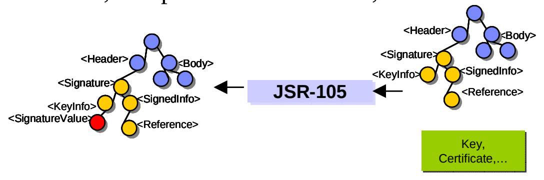

Figure 2. Concept of JSR-105

```
// Retrieve the SOAP header
Element header = getSOAPHeader;

// Prepare the security token (and key)
XMLSignContext sigContext =
new DOMSignContext(keypair.getPrivate(), header);
XMLSignatureFactory sigFactory =
XMLSignatureFactory.getInstance("DOM",
     (Provider)Class.forName(providerName).
     newInstance());

// Generate <Reference> element
Reference ref = sigFactory.newReference("#Body",
  sigFactory.newDigestMethod(DigestMethod.SHA1,
  null));

// Generate the <SignedInfo> element
SignedInfo signedInfo =sigFactory.newSignedInfo(
  sigFactory.newCanonicalizationMethod(
     CanonicalizationMethod.INCLUSIVE_WITH_COMMENTS,
     (C14NMethodParameterSpec) null),
     sigFactory.newSignatureMethod(
            SignatureMethod.DSA_SHA1, null),
     Collections.singletonList(ref));

//Generate the <KeyInfo> element
KeyInfoFactory kif =
   sigFactory.getKeyInfoFactory();
KeyValue kv =
   kif.newKeyValue(keypair.getPublic());
KeyInfo keyInfo =
kif.newKeyInfo(Collections.singletonList(kv));

// Generate <Signature> element
XMLSignature sig =
   sigFactory.newXMLSignature(signedInfo,
   keyInfo);

// Process the digital signature
sig.sign(sigContext);
```

Figure 3. Sample program using JSR-105

2.2 WSS4J

WSS4J is the Apache Open Source of WS-Security implementation. WSS4J provides an API for setting up a WS-Security configuration. It also allows using the configuration file called a property file to set up the WS-Security configuration. It has three scenarios for setting up the configuration: one is to use the property file only, another is to use both the property file and the APIs, and the third is to use only the APIs. In the second scenario, it assumes that the property file specifies the keys and the certificates and APIs provide the other properties. In most cases, the first or second scenario is used. But this paper describes the third scenario because this paper is focusing on APIs.

The basic approach of WSS4J wraps the complicated WS-Security procedures into the signature or encryption classes with security tokens including the private keys or the X.509 certificates. It has the advantage of simplifying the programming model without regard to the message structure, but WSS4J doesn't sufficiently abstract the security token. It provides two different ways for signature and encryption with security tokens. When the signature requires a X.509 security token, the Crypto interface is used. If the signature requires a username token, the WSSecUsernameToken class is used. Figure 4 shows the concept of the programming model in WSS4J. In addition, WSS4J provides the Tokens interface which is for WS-SecurityPolicy, not WS-Security implementation. The existence of some interfaces and classes about the security token would confuse end users.

Figure 5 shows a sample code to sign a message using WSS4J. First, the application instantiates the Crypto class which manages all private keys and X.509 certificates. A Crypto object is a single object in the application's code although the other APIs have separate objects for storing keys for each process, such as signature and encryption. The application needs to load a key store file and pass the key store to this Crypto object. Next, the application instantiates the WSSSignature class and pass an alias and a password used to retrieve the private key and the X.509 certificate for signature (line 21). This is a notable feature of a program using WSS4J. If there is no code to specify the alias and the password, the WS-Security engine cannot be applied to the SOAP message because there is no way to retrieve the key or the X.509 certificate. The runtime of WSS4J has default values for some properties, such as signature and canonicalization methods. Default values helps to make the programming easier. The example of Figure 5 uses the default values. If the default value is not appropriate for the processing, the application can change the property value by calling certain methods. Finally, the method WSSecSignature.build(Document, Crypto, WSSecHeader) signs the message and returns the Document including the signature.

The merits of WSS4J are to wrap the complicated processing into the signature and encryption classes and to support default values. The drawback of WSS4J is the inadequate and inconsistent abstraction of the security tokens.

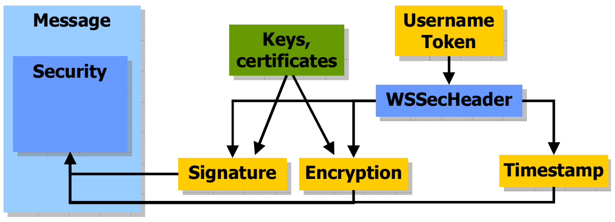

Figure 4. Concept of WSS4J

```
1. // Retrieve the SOAP message
2. SOAPEnvelope unsignedEnvelope = getSOAPEnvelope();
3.
4. // Generate crypto from the key store file
5. Crypto crypto =CryptoFactory.getInstance();
6. char[] secret = "secret".toCharArray();
7. KeyStore ks = KeyStore.getInstance("JKS");
8. ClassLoader loader =
9. Thread.currentThread().getContextClassLoader();
10.InputStream is =
11.   loader.getResourceAsStream("keystore.jks");
12.if (is == null) {
13.     throw new IOException(
14.            "failed to load key store resource");
15.}
16.ks.load(is, secret);
17.is.close();
18.crypto.setKeyStore(ks);
19.
20.//Generate the signature class
21.WSSecSignature sig = new WSSecSignature();
22.
23.// Set the required information into the signature
24.sign.setUserInfo("keyAlias", "password");
25.Document doc = unsignedEnvelope.getAsDocument();
26.
27.//Insert the certificate to SOAP message
28.WSSecHeader secHeader = new WSSecHeader();
29.secHeader.insertSecurityHeader(doc);
30.
31.// Process the digital signature
32.Document signedDoc =sig.build(doc, crypto,
33.     secHeader);
```

Figure 5. Sample program using WSS4J

2.3 WSE

Microsoft has released WSE, which is an software extension component for .NET to support Web Services and related technologies, such as WS-Security and Web Services Addressing [11]. This API design is based on the structure of the message secured with WS-Security and uses classes encapsulating the WS-security operations. Figure 6 shows the concept of the programming model in WSE. It is simpler than that of WSS4J. When it signs a SOAP message, both the signature class and the security token class are inserted into the header class. The MessageSignature class is inserted into security.elements and all SecurityToken classes are inserted into security.tokens, even if a SecurityToken has already been inserted into MessageSignature as shown in Figure 6. An exception will be thrown if the user forgets to insert the SecurityToken element into both the MessageSignature and security.tokens sections.

WSE provides a SecurityToken interface and some security token classes that implement the interface, such as the X509SecurityToken and the UsernameToken. These objects are generated with detailed information, such as the username and key store. For example, when an X509SecurityToken object is generated, the end user needs to retrieve the X.509 certificate from a key store file. That requires work by end users. Some default values are embedded in the runtime. When end users want to update the default value of a certain property, they can change the field value directly (since it is public). Figure 7 shows a sample code using WSE. The merits of WSE are to make the programming model simple and to support default values. The drawback of WSE is that it is difficult to generate the security tokens.

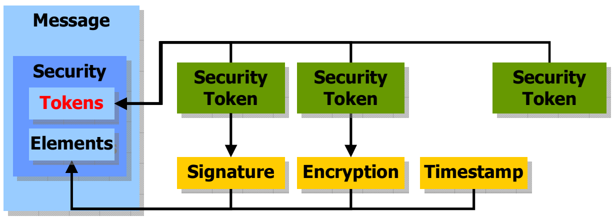

Figure 6. Concept of WSE

```
//Retrieves the soap message
SoapEnvelope envelope = getSOAPEnvelope();

// Retrieve the Security object
Security security = getSecurity();

// Prepare the security token
X509SecurityToken signatureToken = null;
X509Store store = new
    X509Store(StoreName.My,StoreLocation.CurrentUser);
store.Open(OpenFlags.ReadOnly);

try{
    X509Certificate2Collection certs =
     store.Certificates.Find(
          X509FindType.FindBySubjectDistinguishedName,
          "CN=WSE2QuickStartClient", false);

    X509Certificate2 cert;
    if (certs.Count == 1) {
        cert = certs[0];
        signatureToken = new X509SecurityToken(cert);
    } else signatureToken = null;
} catch (Exception ex) {
    Console.WriteLine(ex.ToString());
} finally {
    if (store != null) store.Close();
}

if (signatureToken == null) {
   throw new SecurityFault(
      "Message Requirements could not be satisfied.");
 }

// Insert the security token to SOAP message.
security.Tokens.Add(signatureToken);

// Specify the security token to sign SOAP message with.
MessageSignature sig =
        new MessageSignature(signatureToken);
security.Elements.Add(sig);
```

Figure 7 Sample program of WSE

3. WSSAPI

The keys of API design are correctness, simplicity, and efficiency [4]. We focused on abstracting the WS-Security processing in the design of the API. As shown in Figure 1, WS-Security defines complicated procedures for the signature and encryption processes. The common features of both processes are the use of keys, certificates, and security tokens. We designed to link the security tokens and the keys and to link the security tokens and the certificates. This means the security tokens wrap the keys and the certificates. We designed the security token in one consistent programming construct. The classes which are responsible for signature and encryption were designed to use the security tokens. This is a crucial feature of WSSAPI. In the other APIs, the security token generation is difficult for end users. To make it easy, WSSAPI exploits the Java Authentication and Authorization Service (JAAS) [12] mechanism. This design simplifies the code to generate security tokens.

Another feature of WSSAPI is that all of the processes such as signature and encryption are inserted into the WSSGenerationContext. WSSAPI provides a single method called WSSGenerationContext.process() to secure a message with WS-Security. Once it is called, all of the required operations are performed. This design contributes to the implementation's performance because the implementation can work on the signature and encryption in parallel if there are no dependencies. Performance is one of the significant issues in WS-Security implementations, as shown in several studies [13, 14, 15]. Figure 8 shows the concept of the programming model in WSSAPI.

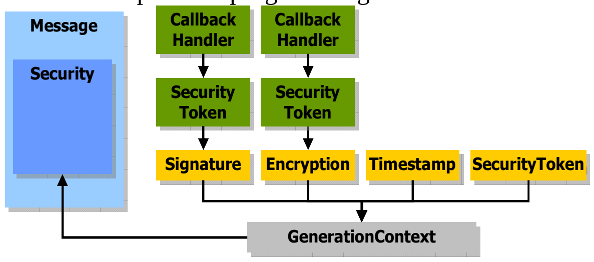

Figure 8 Concept of WSSAPI

The signature and encryption classes have all of the necessary properties, such as signature methods or encryption methods. WSSAPI provides default values for some properties, like convention (or typical usage) over configuration [3]. One of the advantages of WSSAPI is to declare the default values in an external file which can be customized without changing the implementation, although the other APIs embed such values in their implementations. When end users want to overwrite the default value of a property, they merely call a method to change the value. If they want to change the default value of a property, they can directly customize the default value declared in the file.

These concepts lead to a six-step consistent programming model to apply the signature and encryption. Here are the six-steps required in the programming model for a signature:

1. Set the required information about a security token in the callback handler.
   CallbackHandler new CallbackHandler(arguments)

2. Instantiate a security token using the callback handler
   SecurityToken
   WSSFactory.newSecurityToken(SecurityToken.class, CallbackHandler)

3. Instantiate a signature class using the security token.
   WSSSignature
   WSSFactory.newWSSSignature(SecurityToken)

4. Overwrite some properties, if their default values are not appropriate, such as the parts to be signed, the signature method, canonicalization method, transform method, and/or digest method.
   void WSSSignature.setSignatureMethod(String)

5. Register the signature into the WSSGenerationContext.
   void WSSGenerationContext.add(WSSSignature)

6. Perform the signature process and insert the signature header into the message.
   void WSSGenerationContext.process(Object)

WSSAPI is also designed to consider some of the factors defined in the Software Quality Metrics [16].

- Correctness:
  - WSSAPI is designed to conform to the OASIS WS-Security Specification 1.1. It also provides the methods for signature confirmation, header encryption, and so on.
- Efficiency:
  - Efficiency here refers to the fewest possible lines of code using WSSAPI. The bottleneck of programming using the other APIs is in generating the security tokens. WSSAPI reduces the amount of code by using the JAAS API internally to generate the security tokens.
  - The default property values also reduce the amount of code.
- Usability:
  - High level constructs are easier for end users to understand. WSSAPI provides a uniform way to generate security token classes, the Signature class, and the encryption class. All of the classes are generated from the factory class.
- Flexibility:
  - Generating security tokens requires a JAAS login module and a callback handler with some properties, for example to retrieve a key or a certificate from a key store file. This means that all end users have to do is to set the properties in the callback handler if they use the default security token. This is an advantage of WSSAPI for flexibility and usability. When using the other APIs, end users need to retrieve a key or a certificate directly from a key store file and generate a security token object using their data. The way of generating a security token object is different for the different kind of security token. If a new security token must be supported, they will need to study how to generate its objects for the other APIs.
  - Another flexibility feature is to declare default values of the properties in the external file, not embedding them in the implementation. End users can customize the file to match various environments.
- Portability:

  WSSAPI can be used on any other implementation of Web Services engine, like DOM or JAX-WS programming model, because it is designed to be independent of any specific implementation. It means the end users does not have to have understanding of the underneath runtime implementation.

```
//Retrieve the soap message
Object msgContext = getMessageContext();

WSSFactory factory = WSSFactory.newInstance();
WSSGenerationContext gencont =
            factory.newWSSGenerationContext();

// Prepare the security token
X509GenerateCallbackHandler xgCallbackHandler =
    new X509GenerateCallbackHandler("",
   "keystore.jks", "JKS", "keyAlias",
   "password".toCharArray(), "", null);
SecurityToken st =
   factory.newSecurityToken(X509Token.class,
    xgCallbackHandler );

// Generate the signature class
WSSSignature sig = factory.newWSSSignature(st);

// Register the signature object
//into the WSSGenerationContext
gencont.add(sig);

//Process the WS-Security including the signature
gencont.generate(messagecontext);
```

Figure 9 Sample program using WSSAPI

Figure 9 shows sample code using WSSAPI. End users just prepare a callback handler to generate a security token. The required information about a key and a key store is stored in the callback handler. The information needed for the JAAS login module invocation can be omitted because the default value for the JAAS login module is declared in the external file.

Figure 10 shows the class diagram of WSSAPI. This API is available in the WebSphere Application Server 6.1 Web Services Feature Pack (WAS v6.1 WS FP) [17].

To conclude this section, Table 1 lists the features of WSSAPI and the other APIs. WSS4J, WSE, and WSSAPI are easier to use than JSR-105 and JSR-106 because they use process-centric programming models that encapsulate the signature and encryption processes and they support default values. For security tokens, WSS4J is not well designed for simplicity. Considering the features, WSE and WSSAPI seem superior to JSR-105, JSR-106, and WSS4J.

4. Evaluation

This section compares WSSAPI and the other APIs using the following the evaluation metrics to assess their ease of use:

1. Lines of code (LoC) using APIs

1-1, LoC of the sample code

1-2. LoC of the sample code excluding generation of the security token

2. Number of classes (NoC)
3. Cyclomatic Complexity (CC) [18]

First, we evaluated the LoC of the sample programs (Figures 3, 5, 7, and 9). The program statements are counted by the number of the semicolons in the sample programs. The code sample of JSR-105 uses a given key although the other code samples retrieve keys from key store files. Retrieving a key would add more than 10 lines of code to the JSR-105 example. The difference between items 1-1 and 1-2 shows the difficulties in storing the keys and certificates into the security token classes. In WSS4J and WSE, two-thirds of the sample code is used to generate the security tokens.

The number of classes measures the ease of learning to use an API. The count for WSS4J includes all of the classes in WSS4J. In WSE, we count the classes in the MicroSoft.Web.Services3.Security.* package that contains all of the classes that secure messages. In WSSAPI, we count the classes in the three security-related packages, com.ibm.websphere.wssecurity.wssapi.*, com.ibm.websphere.callbackhandler.*, and com.ibm.ws.wssecurity.wssapi.* packages. We can not count the NoC of the JSR-105 implementation because it is not available. Even if it were available, we could not compare it directly to any of the others because JSR-105 supports only signature, not encryption. WSSAPI provides the fewest classes of all the APIs, which means WSSAPI is the easiest programming model to use for programmers unfamiliar with WS-Security. In addition, WAS v6.1 WS FP makes the com.ibm.ws.wssecurity.wssapi.* package invisible for ease of use. Actually, the user can write code using only 42 classes.

Cyclomatic Complexity is one of the software metrics that measures the number of linearly independent paths through a program's source code [18]. We evaluated the CC of the code samples using each API as the measurement of the code complexity. This shows WSE is the worst because there are three if statements in the sample code. This implies that it is difficult to generate the secu-

Table 1 Features of WSSAPI and the other APIs.

	1, Programming	2, Security Tokens	3, How to set the property values	4, Default values
JSR-105	Message-centric	XMLCryptoContext object with raw data	Constructor	Not supported
WSS4J	Process-centric	Many interfaces and classes for security tokens are provided	Method	Embedded in runtime
WSE	Process-centric	SecurityToken objects with raw data	Change variable directly	Embedded in runtime
WSSAPI	Process-centric	SecurityToken with information on security tokens using callback handlers	Method	Declared in external file

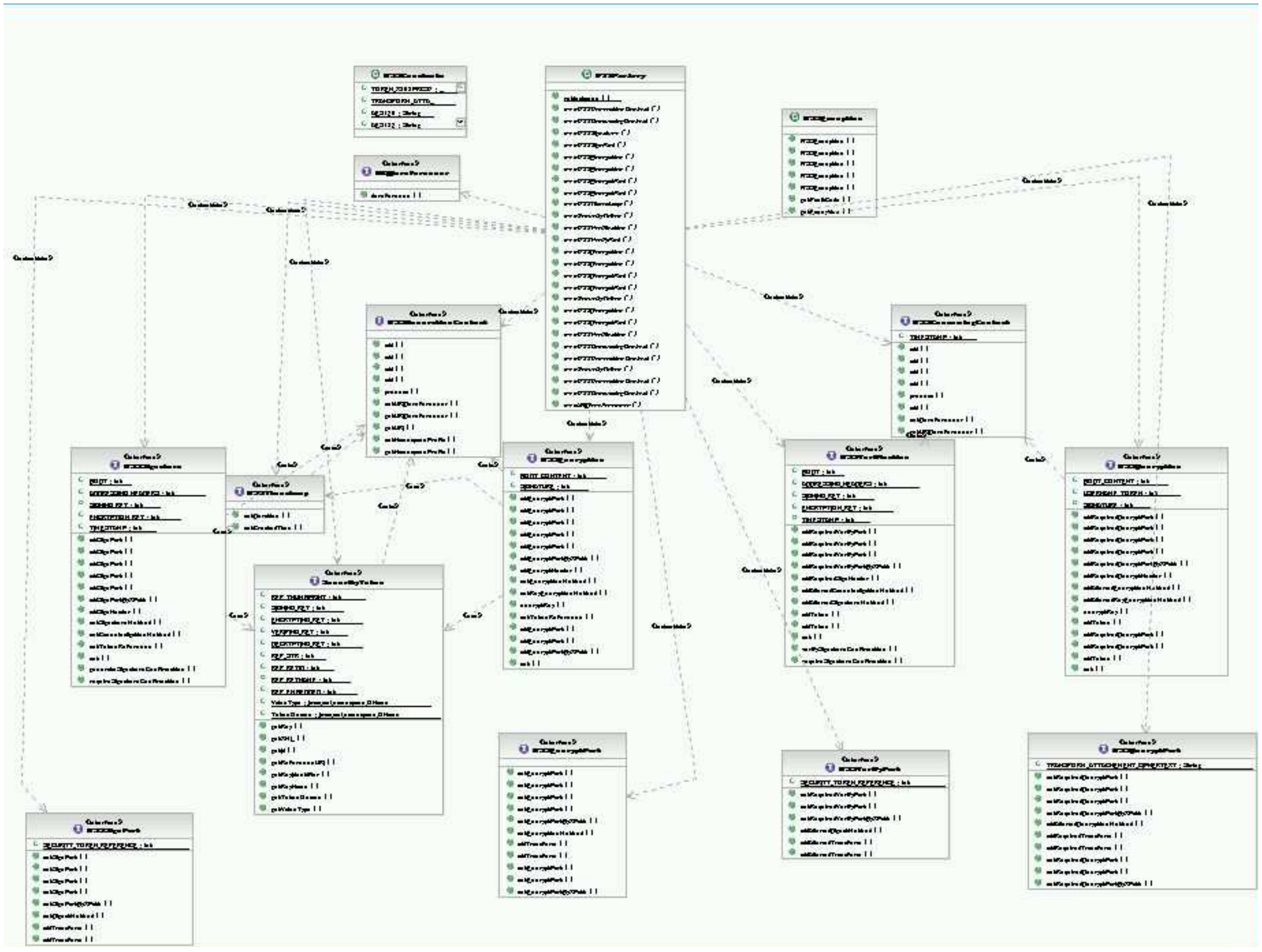

Figure10 Class diagram of WSSAPI

rity tokens.

These evaluations indicate that WSSAPI is the easiest API to use.

5. Conclusion

WS-Security configuration using an API is convenient in rapid prototyping. This paper surveys the WS-Security APIs for users unfamiliar with that specification. We designed WSSAPI by considering the WS-Security operations at a high level of abstraction. WSSAPI has been released as a part of WAS v6.1 WS FP. It has some important features for correctness, efficiency, usability, maintainability, flexibility, and portability. Considering their features, WSE and WSSAPI are superior to JSR-105, JSR-106, and WSS4J. In addition, the evaluation of the ease of use shows that WSSAPI is the most usable. Based on the evaluation and study, it is easier for end users to use WSS API to enable WS-Security compared with the other APIs. WSSAPI is superior to the other APIs in generating the security tokens and in supporting the default values in the external file.

Table 2 Complexity comparisons of the APIs

	1-1, LoC	1-2, LoC	2, NoC	3, CC
JSR-105	10	10	N/A	0
WSS4J	16	7	159	1
WSE	16	5	214	3
WSSAPI	8	5	120	0

References

[1] WS-Security Policy, http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200512

[2] Web Services Security: SOAP Message Security 1.1, http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf
[3] Ruby on Rails, http://www.rubyonrails.com/
[4] E. McManus, "Java API Design Guidelines," http://www.artima.com/weblogs/viewpost.jsp?thread=142428, December, 2005.
[5] XML-Signature Syntax and Processing, http://www.w3.org/TR/xmldsig-core/
[6] XML Encryption Syntax and Processing, http://www.w3.org/TR/xmlenc-core/
[7] WSS4J, http://ws.apache.org/wss4j/
[8] WSE, http://msdn2.microsoft.com/en-us/webservices/aa740663.aspx
[9] JSR-105, http://jcp.org/en/jsr/detail?id=105
[10] JSR-106, http://jcp.org/en/jsr/detail?id=106
[11] Web Services Addressing, http://www.w3.org/Submission/ws-addressing/
[12] Java Authentication and Authorization Service, http://java.sun.com/products/jaas/
[13] S. Makino, et. al., "Implementation and Performance of WS-Security," *International Journal of Web Services Research*, Vol. 1, pp. 58 - 72, 2004.
[14] M. Teraguchi, et. al., "Optimized Web Services Security Performance with Differential Parsing," *International Conference on Service Oriented Computing*, pp. 277 – 288, December, 2006.
[15] W. Zhang and R. V. Engelen, "A Table-Driven Streaming XML Parsing Methodology for High-Performance Web Services," *IEEE International Conference on Web Services*, pp. 197-204, 2006.
[16] J. A. McCall, "An Introduction to Software Quality Metrics," In J. D. Cooper and M. J. Fisher (Eds.), *Software Quality Management*, Petrocelli, 1979.
[17] WebSphere Application Server 6.1 Web Services Feature Pack, https://www14.software.ibm.com/iwm/web/cc/earlyprograms/websphere/wsvwas61/
[18] T. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering*, Vol. SE-7, No. 4, pp. 308 – 320, September 1976.