The Security Aspects in Web-Based Architectural Design using SOA

Asadullah Shaikh
Research Group in Software Engineering
Universitat Oberta de Catalunya (Spain)
ashaikh@uoc.edu

Aijaz Soomro
Department of Computer Science
University of Durham (UK)
a.a.soomro@dur.ac.uk

Sheeraz Ali
Cursor Software Solutions (UAE)
sheeraz@cursorsoft.net

Nasrullah Memon
Maersk Mc-Kinney Moller Institute
University of Southern Denmark (Denmark)
and Hellenic American University Athens (GR)
memon@mmmi.sdu.dk

Abstract

Distributed web-based applications have been progressively increasing in number and scale over the past decades. There is an intensification of the need for security frameworks in the era of web-based applications when we refer to distributed telemedicine interoperability architectures. In contrast, Service Oriented Architecture (SOA) is gaining popularity day by day when we specially consider the web applications. SOA is playing a major role to maintain the security standards of distributed applications. This paper proposes a secure web-based architectural design by using the standards of SOA for distributed web application that maintains the interoperability and data integration through certain secure channels. We have created CRUD (Create, Read, Update, Delete) operations that has an implication on our own created web services and we propose a secure architecture that is implemented on CRUD operations.

Keywords: SOA Security, web-based SOA Security, SOA Security Aspects, SOA in CRUD, CRUD Security

1. Introduction

Service Oriented Architecture is gaining popularity day by day due to the fact that it is useful for making interoperable web-based applications. The non-secure SOA based applications create many problems for different web based applications especially concerning the security aspects. However, security in enterprise applications has not been addressed consciously. In the present situation, the security aspects in architectural designs are not considered until a serious problem occurs during the developmental stages that violates the policy. Architectural designs are always being considered as a preliminary stage of a development process, therefore, if the designing of an architecture for a system is planned to be secured, then it should not be a major problem to maintain the security during the implementation process. Apart from that, the current telemedicine web-based applications are being widely used in the e-health care system [1,2], and out of them the majority of applications are distributed due to the several placement locations. These applications store the patient's history, personal details etc, which are quite confidential data. Nevertheless, there is less attention paid to the security of these heterogenous telemedicine web based applications[2]. Figure 1 refers to a normal structure of a web application interaction based on web services between the service provider and service consumer applications.

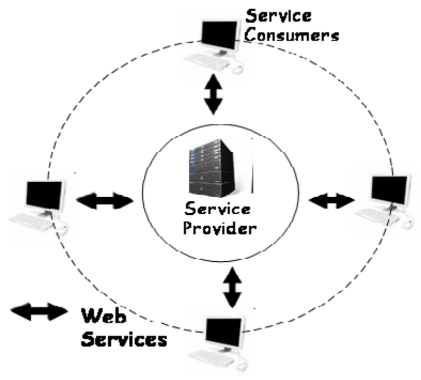


Figure 1: Web Service Interaction

Considering all of above aspects, this paper proposes the security aspects for our developed architectural design [3] to address the security issues in order to provide a suitable solution. To configure these security aspects correctly, we took an account of several standards of security implementation of the web for multiple distributed applications and therefore we decided to design an architecture that covers the security measures which are necessary in the developmental process and that will also help developers. Our proposed secure architectural design is based on Public Key Infrastructure (PKI) security for authentication and authorization [13]. Furthermore, our security aspects have an implication on CRUD operations which are web-based services that were implemented previously [3].

The rest of the paper is structured as follows. Section 1.1 introduces the concept of CRUD operations, section 1.2 defines the security using SOA and section 1.3 outlines the security problems. Furthermore, Section 2 provides an overview of web security in CRUD operations. Section 3 presents the proposed architecture. Later on, section 4 explains the experiments and results. Finally, section 5 is about previous work related to SOA security and section 6 draws on some conclusions and future work.

1.1. CRUD Operations

Our previously designed telemedicine architecture is based on SOA. We have implemented CRUD operations as a basic application function in order to interact with the database. These CRUD operations perform database operations such as data retrieval, data creation, data deletion and data updation at an application level. Figure 3 briefly describes the mapping of CRUD functions with database operations.

1.2. Security using SOA

The SOA approach is widely used to develop the several components of web services. These services contain their own security techniques [4]. WS-Security provides a communication protocol to apply the security of web services [5]. It describes Simple Object Access Protocol (SOAP) messaging to enable security services specially in terms of integrity, message confidentiality and message authentication, and furthermore, it helps to provide encryption techniques. This kind of security provides a flexible design for security models such as Secure Sockets Layer (SSL) and Kerberos. Nevertheless, it provides security tokens, trust domains, signatures and encryption technologies. In order to exchange the secured messages using WS-Security, there will be common tokens which should be shared between requester and provider. Figure 2 describes the general structure of WS-Security.

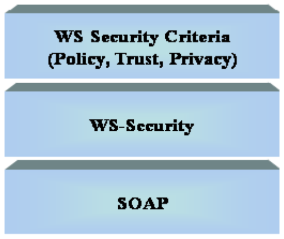


Figure 2: WS-Security Structure [4]

1.3. Security Problems

WS-Security is quite flexible and capable, however, its configuration in real time examples is difficult for users. So far and to the best of our knowledge, the security for CRUD operations used in web services has not been discussed sufficiently. Therefore the security aspects of the following points are not considered:

- CRUD operations are developed in web services, and so what should the security measurements be in order to get patient data?
- CRUD operations are persistent storage functions that are implemented in the form of web services [1], If the patient data is updated, how can its authentication be ensured?
- In the case of deleting a record, how can somebody be authorized to do this?

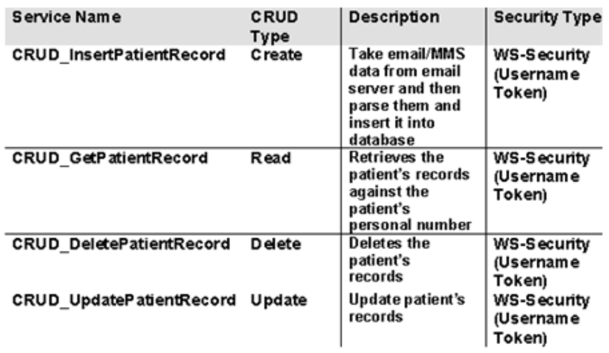
This paper presents security aspects to handle security in accordance with the CRUD operations that are implemented and accessible as a web service.

2. WS-Security in CRUD operations

CRUD is the combination of four basic operations: Create, Read, Update, and Delete which are used for permanent storage[1], and are the major components of every computer software application. Since data is the most important and valuable in any web-based application, therefore it should be transmitted securely. In our proposed work, we have provided the security for our CRUD operations using WS-Security.

3. Proposed Architecture

In this section, we describe the proposed architecture of our web services along with the CRUD security implementation. The major structure of the proposed process of SOA security for CRUD operations is divided in two external interfaces; one interface is between the nurse and the application and the other interface is between the doctor and the



Service Name	CRUD Type	Description	Security Type
CRUD_InsertPatientRecord	Create	Take email/MMS data from email server and then parse them and insert it into database	WS-Security (Username Token)
CRUD_GetPatientRecord	Read	Retrieves the patient's records against the patient's personal number	WS-Security (Username Token)
CRUD_DeletePatientRecord	Delete	Deletes the patient's records	WS-Security (Username Token)
CRUD_UpdatePatientRecord	Update	Update patient's records	WS-Security (Username Token)

Figure 3: List of CRUD Services with Security [1]

application. The flow of both interfaces is illustrated in Figure 4.

3.1. Interface between Nurse and Application

Initially, the first operation that needs to be performed is that the nurse sends the patient record to a telemedicine application from a cell phone with a premium phone number registered in the application. Premium numbers are the special type of cell phone numbers that are designed especially for telemedicine applications in order to process secure data transmission from nurse end to doctor end. Secondly, an application sends a random unique code for verification to the nurse end. Thirdly, the nurse replies for verification and finally, if the verification is performed successfully, then the data will be processed (update or create) to CRUD operations.

3.2. Interface between Doctor and Application

In order to maintain the security for our telemedicine application, the following steps are undertaken in the case of a doctor's interaction with application.

1. The doctor selects a CRUD operation from the given User Interface (UI) which will be in the form of a web service. Afterwards, the running telemedicine application encrypts the message with a private key and then the secure SOAP message with encryption will be sent to the telemedicine application.
2. The security handler of a telemedicine application decrypts the message with the doctor's public key.
3. The security handler checks certain permission given to a particular doctor in order to select the CRUD operation to be performed.
4. CRUD operation is performed.

3.3. Security Measures for an Intruder

In the worst case scenario, the intruder can also try to attack our telemedicine application, therefore, what will happen initially if the intruder sends the operation message, the SOAP message without encryption will be sent to a telemedicine application to access the CRUD operations. Afterwards, the security handler of a telemedicine application may decrypt the message with the doctor's public key but the decryption will fail and the message will be discarded.

3.4. Security Implementation at Nurse's End

Authentication and Authorization (AA) at the nurse's end is performed in two levels. In the first level of security, the messages sent by nurse can only be accepted from premium cell phone numbers that are registered in an application. In the second level of security, once the messages are received, the telemedicine application sends a unique code to the nurse for verification, and the nurse replies to the message with the same code. The idea to introduce the second level security is to ensure that the message was sent from a premium number through a cell phone as the message can also be sent from Web2SMS or Web2MMS with any number. Therefore, a nurse needs to send the reply for verification in order to pass the secure data into CRUD operations.

3.5. Security Implementation at the Doctor's End

We have developed the security implementation of telemedicine system architecture [1] using Apache Axis [10], which is an open source XML based Web Service framework. We have used Apache Web Services Security for Java (WSS4J) [11] which is the implementation of the OASIS Web Services Security (WS-Security) from the Organization for the Advancement of Structured Information Standards (OASIS) Web Services Security Technical Committee (TC) [12]. WSS4J is used to sign and verify SOAP messages with WS-Security information. Furthermore, WSS4J is used for securing our CRUD web services along with the support of the Apache Axis web service framework. WSS4J generates and processes the SOAP bindings for XML Security with *XML Signature* and *XML Encryption*. It also provides the *Tokens* for username, timestamps and Security Assertion Markup Language (SAML) tokens. The security of CRUD operations is deployed with username tokens.
The configuration of the security deployment and usage is described by the implementation given in listing 1 to 4.

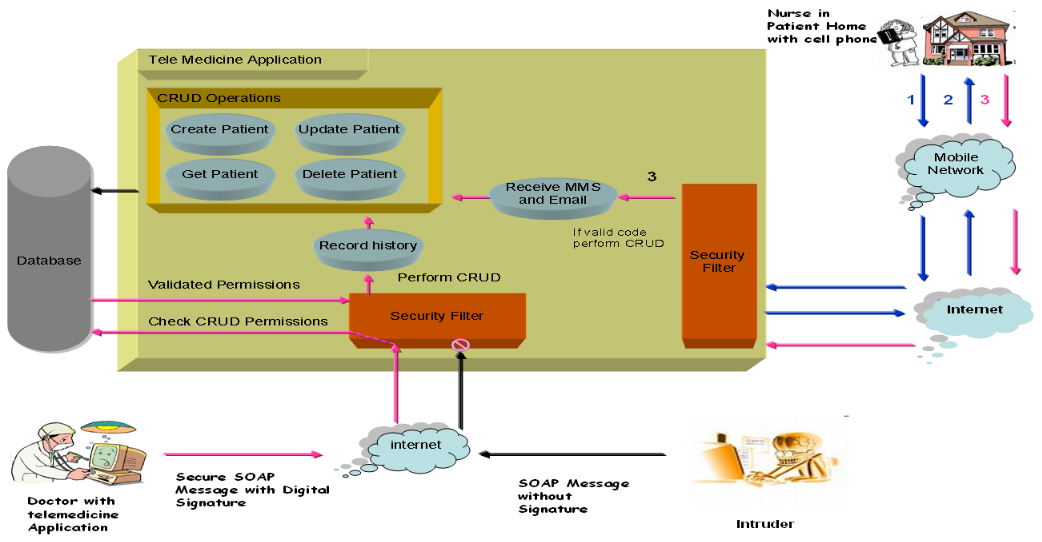


Figure 4: Proposed Security Architectural Design

In listing 1, the WSS4J handlers are added to the service deployment descriptor in the Web Service Deployment Descriptor (WSDD) file for adding the WS-Security layer to our telemedicine CRUD services. Afterwards, adding handlers, the server side deployment descriptor also defines the request and response flows. In the *Request Flow*, with every incoming request for a CRUD operation, there are two security handlers that authenticate and authorize the request. The TeleWoundServiceSecurityHandler decrypts the SOAP message with a public key of the Doctor using PKI security. Once the message is decrypted, WSDoAllReceiver verifies the username and password for authorization. Meanwhile, in the *Response Flow*, every response is encrypted with the doctor's public key and the message is digitally signed for authentication with the telemedicine's private key.

In listing 2, PWCallback class is created by implementing the CallbackHandler interface. This CallbackHandler is called before every CRUD operation request to check the authorization of the provided username and password. If the username and password exist in the application, then there will be the verification of permission on the selected CRUD operation. For example, the user has permission to access the CRUD operation READ the record while CRUD operation UPDATE the record is not permitted, then PWCallback class will not allow any action on UPDATE operation.

In Listing 3, TeleWoundServiceSecurityHandler class is securing the message by encryption and decryption using PKI security. Message is digitally signed to authenticate the provider of the message. With every incoming CRUD request, handleRequest() method is called to decrypt the SOAp message with sender's public key. Therefore, for every outgoing response, handleResponse() is called for encrypting the SOAP message and to digitally sign the message.

In Listing 4, Deployment Description for client side is provided to handle the Request Flow of a SOAP message. Client receives the response from TeleWound application as an encrypted soap message. This encrypted soap message is decrypted using the public key of TeleWound and after that the result of the CRUD operations are extracted from the normal decrypted soap message.

```
Listing—1
<deployment xmlns=http://xml.apache.org/axis/wsdd/ xmlns:java=
"http://xml.apache.org/axis/wsdd/providers/java">
 <service name="TeleWound—wss—01" provider="java:RPC" style=
 "document" use="literal">
  <requestFlow>
   <handler type="java:org.apache.axis.handlers.JAXRPCHandler">
    <parameter name="scope" value="session"/>
    <parameter name="className" value="TeleWoundServiceSecurityHandler"/>
    <parameter name="keyStoreFile" value="c:\\TeleWound\\key\\server.ks"/>
    <parameter name="trustStoreFile" value="c:\\TeleWound\\key\\server.ts"/>
    <parameter name="certEntryAlias" value="clientkey"/>
   </handler>
   <handler type="java:org.apache.ws.axis.security.WSDoAllReceiver">
    <parameter name="passwordCallbackClass" value="PWCallback"/>
    <parameter name="action" value="UsernameToken"/>
   </handler>
  </requestFlow>
  <responseFlow>
   <handler type="java:org.apache.axis.handlers.JAXRPCHandler">
    <parameter name="scope" value="session"/>
    <parameter name="className" value="TeleWoundServiceSecurityHandler"/>
    <parameter name="keyStoreFile" value="c:\\TeleWound\\key\\server.ks"/>
    <parameter name="trustStoreFile" value="c:\\TeleWound\\key\\server.ts"/>
    <parameter name="certEntryAlias" value="clientkey"/>
   </handler>
  </responseFlow>
  <parameter name="scope" value="application"/>
  <parameter name="className" value="TeleWound"/>
  <parameter name="allowedMethods" value="CRUD_InsertPatientRecord"/>
  <parameter name="allowedMethods" value="CRUD_UpdatePatientRecord"/>
  <parameter name="allowedMethods" value="CRUD_DeletePatientRecord"/>
  <parameter name="allowedMethods" value="CRUD_GetPatientRecord"/>
 </service>
</deployment>
```

```
Listing—2
public class PWCallback{
 public void handle(Callback[] callbacks) throws
  IOException,UnsupportedCallbackException {
  for (int i = 0; i < callbacks.length; i++) {
   if (callbacks[i] instanceof WSPasswordCallback) {
    WSPasswordCallback pc = (WSPasswordCallback)callbacks[i];
    // set the password given a username
    if ("TeleMedicineAdmin".equals(pc.getIdentifier(){
      // set the password
    }}
   else{
```

```
  throw new UnsupportedCallbackException(callbacks[i],"Unrecognized Callback");
}}}
```

```
Listing-3
Public class TeleWoundServiceSecurityHandler implements Handler {
  private String keyStoreFile, keyStoreType, keyStorePassword,
       keyEntryAlias, keyEntryPassword, trustStoreFile,
       trustStoreType, trustStorePassword, certEntryAlias;
  public boolean handleRequest(MessageContext context) {
    try {
      SOAPMessageContext soapContext = (SOAPMessageContext)context;
      SOAPMessage soapMessage = soapContext.getMessage();
      Document doc = SOAPUtility.toDocument(soapMessage);
      Utility.decrypt(doc, keyStoreFile, keyStoreType,
            keyStorePassword, keyEntryAlias, keyEntryPassword);
      Utility.verify(doc, trustStoreFile, trustStoreType,
            trustStorePassword);
      Utility.cleanup(doc);
      soapMessage = SOAPUtility.toSOAPMessage(doc);
      soapContext.setMessage(soapMessage);
    } catch (Exception e){
        System.err.println("handleRequest Exception:" + e);
        return false;
    }
    return true;
  }
  public boolean handleResponse(MessageContext context) {
    try {
      SOAPMessageContext soapContext = (SOAPMessageContext)context;
      SOAPMessage soapMessage = soapContext.getMessage();
      Document doc = SOAPUtility.toDocument(soapMessage);
      Utility.sign(doc, keyStoreFile, keyStoreType,
          keyStorePassword, keyEntryAlias, keyEntryPassword);
      Utility.encrypt(doc, trustStoreFile, trustStoreType,
          trustStorePassword, certEntryAlias);
      soapMessage = SOAPUtility.toSOAPMessage(doc);
      soapContext.setMessage(soapMessage);
    } catch (Exception e){
      System.err.println("handleResponse Exception:" + e);
      return false;
    }
    return true;
  }
  public boolean handleFault(MessageContext context) {
    return true;
  }
  public void init(HandlerInfo config) {
    Map configProps = config.getHandlerConfig();
    keyStoreFile = (String)configProps.get("keyStoreFile");
    keyStoreType = (String) configProps.get("keyStoreType");
    keyStorePassword = (String) configProps.get("keyStorePassword");
    keyEntryAlias = (String) configProps.get("keyEntryAlias");
    keyEntryPassword = (String) configProps.get("keyEntryPassword");
    trustStoreFile = (String)configProps.get("trustStoreFile");
    trustStoreType = (String) configProps.get("trustStoreType");
    trustStorePassword = (String) configProps.get("trustStorePassword");
    certEntryAlias = (String) configProps.get("certEntryAlias");
  }}
```

```
 Listing-4
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
java="http://xml.apache.org/axis/wsdd/providers/java">
 <transport name="http" pivot="java:org.apache.axis.
 transport.http.HTTPSender">
  <globalconfiguration>
   <requestFlow>
    <handler type="java:org.apache.axis.handlers.JAXRPCHandler">
     <parameter name="scope" value="session"/>
     <parameter name="className" value="TeleWoundServiceSecurityHandler"/>
     <parameter name="keyStoreFile" value="c:\\TeleWound\\key\\server.ks"/>
     <parameter name="trustStoreFile" value="c:\\TeleWound\\key\\server.ts"/>
     <parameter name="certEntryAlias" value="clientkey"/>
    </handler>
  </globalconfiguration>
 </transport>
</transport>
```

4. Experiments and Results

In this section, we present the environment in which the experiment examines the assumptions that are needed to maintain the consistency and dynamics of the proposed security architecture. Table 1 summarizes the experimental results obtained by implementation of our proposed technique. Furthermore, the column *Actor*, represents an actor which is the user of the telemedicine application, and the column *CRUD Operations(Encode)* describes the operation called by an actor that depends upon certain rights. Similarly, the column *CRUD Data* shows the patient's data, and the column *Encryption + Signature* describes the encrypted CRUD data along with the signature, however, the column *Security Type* shows the type of security implemented on prescribed actor depending on permissions. Finally, the column *Decryption* displays the patient's decrypted data if and only if all the security steps become successful and the column *Receiving Message(Decode)* shows a message once the decryption has failed or passed. All these results provide an overview of our developed security over CRUD operations.

Actor	*CRUD Operations (Encode)*	*CRUD Data*	*Encryption + Signature*	*Security Type*	*Decryption*	*Receiving* Message *(decode)*
Nurse	Create,Update Patient Record	Patient Data	N/A	Premium Number +Verification Code	N/A	1. Patient Record 2. Verification Code
Doctor	Update,Delete,Read Patient Record	Patient Data	BN89bOi + 978654	PKI Encryption+ Digital Signature	Patient Data	CRUD Operation Perfomed
Intruder	Create,Update,Delete Read,Patient Record	Patient Data	Any Encryption or Signature	Invalid PKI Encryption +Digital Signature	No Data Received due to wrong Encryption + Signature	CRUD Operation Failed

Table 1: Results of proposed security Architecture

5. Related Work

In this section, we discuss the existing work on SOA Security. Most of the work in this area is done for SOA security specification. Their goal of implementing the security is to achieve the authentication, authorization and so on for web services. However, research work on the CRUD operations using WS-Security is hardly found in the literature.

Phan, Cecilia [6] addressed the security challenges for SOA. The author described the problems raised from XML which is not secure enough and causes problems in security protocol. They also presented certain strategies to cope with vulnerabilities against attacks and other security policy consideration.

Larrucea[7] proposed an approach describing a holistic view of a SOA environment. In this research, ISOAS framework allows functionality criterions of security policies with service specification that allows the definition of functional and non functional components in coherent way and is dependent on the metamodel. This effort is implemented in Eclipse, and it is due to that, that it is an open approach. Apart from that, their approach is aligned with OMG standards.

Satoh F et al. [8] discussed a process of security configuration that defines the responsibilities of developers. In this end-to-end SOA security configuration, several kinds of information are needed such as requirements, platform information and so on. Due to that, they defined the roles of developers during the development phase. SOA security is complex therefore the domain federation is considered in this research. In general, they contribute to the correct configuration to reduce the workload of developers.

Robert Bunge et al. [9] proposed an operational framework of a network administrator using SOA network security. In this research, they characterize the steps in SOA network security in order to collect the information regarding threats and SOA deployments. Furthermore, they collect the SOA security efforts. As a result, by considering the factors of SOA network security, they provide recommendations for dealing with the XML network traffic for SOA applications. The proposed approach is filtered to inspect XML at the network's level. Their framework contributes to secure SOA design by clarifying the duties of network administrators and software engineers using XML-based services.

Yamany H, Capretz M [4] described an intelligent security service that is embedded in a framework to secure web services in SOA. This framework is designed to interact with authentication to run the authentication process and it also helps to secure a possible web attack. An SOA environment holds several security environments that interact through multiple channels. In their work, they have examined the security service layer and message security layer.

All the above work presented so far is not similar to ours, because of implementing security over CRUD operations. If the CRUD operations are secured enough, then there is no need to apply high level security which is a definitely a complex task. Our CRUD operations are interacting with created web services, therefore if we apply the security on CRUD, the web-service will also be secured simultaneously. However, we have designed and implemented a system architecture that represents the scenario of security standards by considering CRUD operations along with web services.

6. Conclusions and Future Work

In this work, we have proposed an architectural design by considering the security aspects for our designed CRUD operations using SOA. We believe that SOA has multiple solutions of web services. The core use of CRUD operations is to fetch, update, delete, read the data from a perspective database, therefore if CRUD is secure enough, then there is no such need to implement the high level security. In our designed architecture, communication is done through SOAP messages and we have implemented WSS4J and PKI security in order to protect SOAP headers. It creates the efficiency of the security process and prevents web attacks. As a future work, we would like propose the efficient security techniques using cloud computing.

References

[1] Asadullah Shaikh, Muniba Memon, Muhammad Misbahuddin, A System Design for a Telemedicine Health Care System. IMTIC 2008, CCIS 20, pp. 295-305, 2008. Springer-Verlag Berlin Heidelberg 2008.

[2] W.M.Omar and A.Taleb-Bendiab, SOA for e-health support services based on grid computing, Proceedings of the IEEE International Conference on Services Oriented.

[3] Asadullah Shaikh, Muniba Memon, Muhammad Misbahuddin , Nasrullah Memon. The Role of Service Oriented Architecture in Tele-Medicine Healthcare System. IEEE Computer Society, 2009. s. 208-213. CISIS 2009, IEEE Computer Society, Fukuoka, Japan.

[4] Yamany, H, Capretz, M,. Use of Data Mining to Enhance Security for SOA. 978-0-7695-3407-7 IEEE. Third International Conference on Convergence and Hybrid Information Technology, 2008. ICCIT '08.

[5] Bob Atkinson, Giovanni Della-Libera, Satoshi Hada et al. Web Services Security (WS-Security). Copyright 2002-2002 International Business Machines Corporation, Microsoft Corporation. [Online] Available: http://www.cgisecurity.com/ws/ws-secure.pdf. [Accessed April. 3, 2009].

[6] Phan, Cecilia. Service Oriented Architecture (SOA) - Security Challenges and Mitigation Strategies. 29-31 Oct. 2007 Page(s):1-7 IEEE Computer Society.

[7] Larrucea, X, Alonso, R. ISOAS: Through an independent SOA Security Specification. ICCBSS 2008, Page(s):92-100 IEEE Computer Society.

[8] Satoh F et al. Methodology and Tools for End-to-End SOA Security Configurations. 2008. IEEE Congress on Services - Part I. On page(s): 307-314 IEEE Computer Society.

[9] Bunge R, Chung S, Endicott-Popovsky B, McLane D. An Operational Framework for Service Oriented Architecture Network Security. Proceedings of the 41st Annual Hawaii International Conference on System Sciences, Page(s):312-312 IEEE Computer Society.

[10] Apache Axis, [Online] Available: http://ws.apache.org/axis/. [Accessed April.3,2009].

[11] Apache WSS4J, [Online] Available: http://ws.apache.org/wss4j/. [Accessed April.3,2009].

[12] Organization for the Advancement of Structured Information Standards, [Online] Available: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss. [Accessed April.3,2009].

[13] X.509 Technical Supplement, [Online] Available: http://msdn.microsoft.com/en-us/library/aa480610.aspx. [Accessed April.3,2009].