SOA Security Aspects in Web-Based Architectural Design

Asadullah Shaikh, Sheeraz Ali, Nasrullah Memon and Panagiotis Karampelas

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.
The paper provides an extensive description of the prevention of replay attacks and a detailed explanation for applying security measures.

Keywords: SOA Security, web-based SOA Security, SOA CRUD Security.

Asadullah Shaikh
Universitat Oberta de Catalunya Barcelona Spain,
e-mail: ashaikh@uoc.edu

Sheeraz Ali
Cursor Software Solutions, UAE,
e-mail: sheeraz@cursorsoft.net

Nasrullah Memon
Maersk Mc-Kinney Moller Institute,
University of Southern Denmark Denmark,
e-mail: memon@mmmi.sdu.dk

Panagiotis Karampelas
Hellenic American University, Athens, Greece,
e-mail: pkarampelas@hau.gr

1 Introduction

Service Oriented Architecture (SOA) is gaining popularity day by day due to the fact that it is useful for making interoperable web-based applications [1, 2]. 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 adequately. 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.

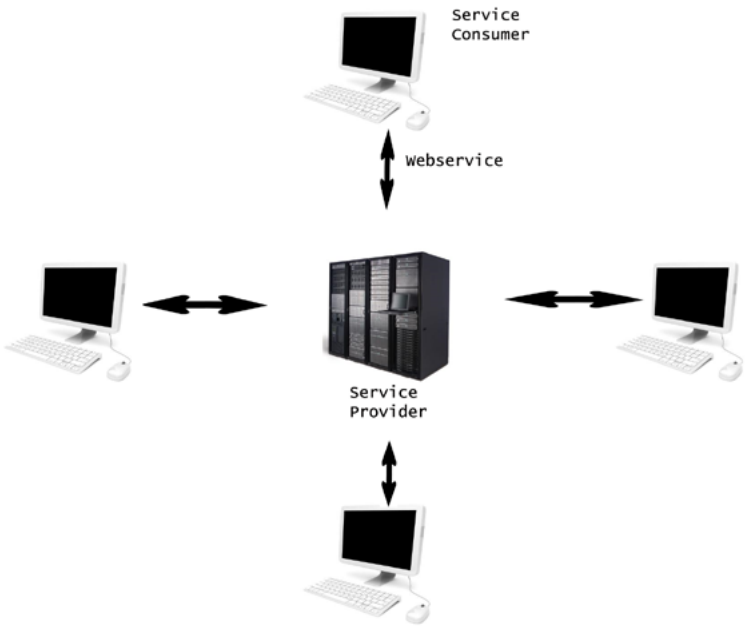

Fig. 1 Web Service Interaction

Apart from that, the current telemedicine web-based applications are being widely used in the e-health care system [3, 4], 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 [4]. Figure 1 refers to a normal structure of a

web application interaction based on web services between the service provider and service consumer applications.

Considering all of above aspects, this paper proposes the security aspects for our developed architectural design [5] to address the security issues in order to provide a suitable solution. To configure these security aspects appropriately, we took into account several standards of security implementation over the web and multiple distributed applications. As a result, we decided to design an architecture that integrates the necessary security measures in the developmental process and is very straightforward for the developers. Our proposed secure architectural design is based on Public Key Infrastructure (PKI) security for authentication and authorization [6]. Furthermore, the proposed security aspects have an implication on CRUD operations which are web-based services that were previously implemented [5].

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. Table 1 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 [7]. WS-Security provides a communication protocol to apply the security of web services [8]. 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.

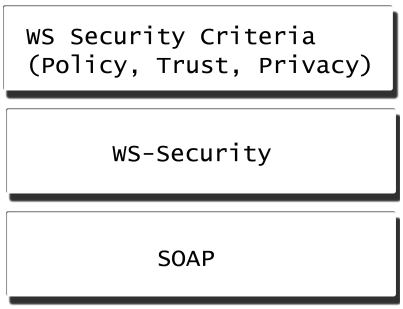

Fig. 2 WS-Security Structure [7]

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 the patient data?
- CRUD operations are persistent storage functions that are implemented in the form of web services [3], 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 [3], 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

Table 1 List of CRUD Services with Security [3]

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	Updates patient's record.	WS-Security (Username Token)

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 application. The flow of both interfaces is illustrated in Figure 3.

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 consume 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, 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.

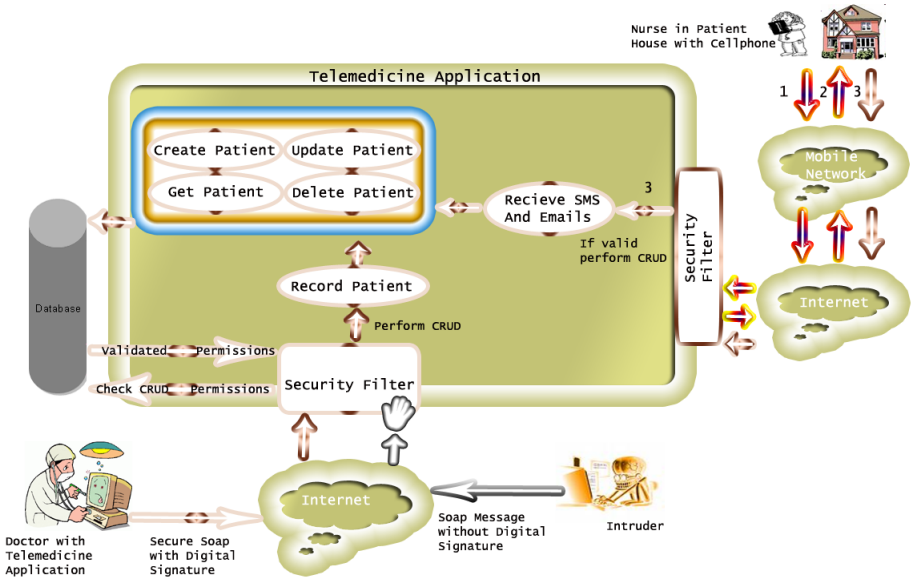

Fig. 3 Proposed Security Architectural Design

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 using Apache Axis [9], which is an open source XML based Web Service framework. We have used Apache Web Services Security for Java (WSS4J) [10] 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) [11]. 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 listings 1 to 5 .

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 *RequestFlow* (Listing 1, Line 5), with every incoming request for a CRUD operation, there are two security handlers that authenticate and authorize the request. The TeleWoundServiceSecurityHandler (Listing 1, Line 21) decrypts the SOAP message with a public key of the Doctor using PKI security. Once the message is decrypted, WSDoAllReceiver (Listing 1, Line 13) verifies the username and password for authorization. Meanwhile, in the *ResponseFlow* (Listing 1, Line 18), 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, a password callback class called PWCallback (Listing 2, Line 1) 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 exists in the application, then there will be the verification of permission on the selected CRUD operation (Listing 2, Line 8). For example, a user may have the access only on the CRUD operation READ, while the CRUD operation UPDATE is not permitted, then PWCallback class will not allow any action on UPDATE operation due the assigned permissions.

Listing 1 Code for Request and Response Flows

```
<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 Code for Password Callback

```
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");
   }}}
```

In Listing 3, TeleWoundServiceSecurityHandler class is securing the message by encryption and decryption using PKI security. The message is digitally signed to authenticate the provider of the message. With every incoming CRUD request, handleRequest() method (Listing 3, Line 5) is called to decrypt the SOAP message with sender's public key. Therefore, for every outgoing response, handleResponse() method (Listing 3, Line 23) is called for encrypting the SOAP message and to digitally sign the message.

Listing 3 Code for Authentication

```
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 Code for Client Request

```
<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>
</deployment>
```

3.6 Prevention From Replay Attack

Replay attacks are also called "man-in-the-middle attack" where a hacker makes independent connections in order to break the barrier of an application. As discussed before, we have implemented the security on a bottom level which is efficient, less expensive and easy to maintain for developers and designers [12]. One of the major factors in our proposed architecture is the prevention from replay attacks where an attacker can access our encrypted SOAP messages to extract the patient's confessional data. Figure 4 illustrates the scenario in which the hacker/intruder tries to breach encrypted security messages in order to reach at the telemedicine web-based application. To prevent this, we have used PKI and digital signatures for encryption,

authentication and authorization. However, the authentication is not limited to only that but also the digitally signed message can be recorded through several capturing techniques which can be resend. We call this a Replay Attack

Replay attacks can be prevented by making each SOAP message unique. There are different type of techniques used for uniqueness of SOAP message like a number or bit string used only once (nonce), time stamping etc. Fortunately, WSS4J provides time stamping for each SOAP message to prevent replay attack easily. In order to avoid these types of attacks, we have used timestamping in our security architecture. Timestamping is a chain of characters, pointing the data or time at which an event is occurred. In timestamping, each event is recorded by a computer. Listing 5 represents the code used in our telemedicine application to avoid the replay attacks through timestamping. We have set the maximum duration of 10 seconds of each timestamp. Figure 5 shows an over all view after prevention from replay attacks.

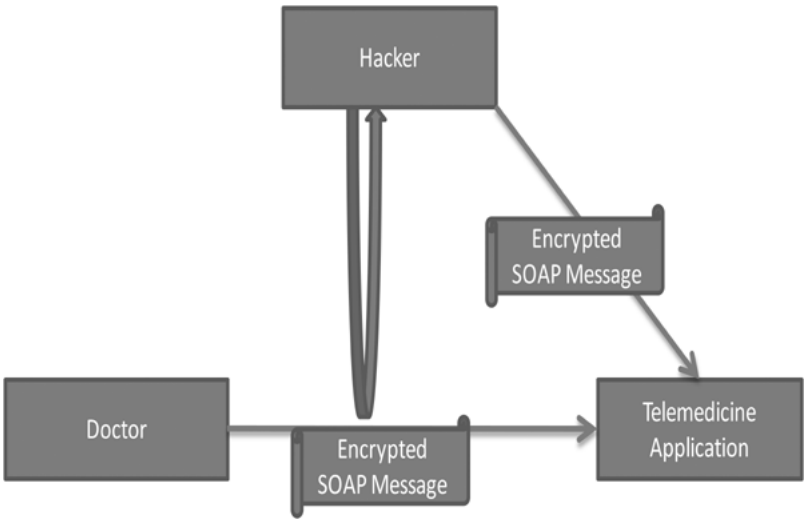

Fig. 4 Proposed Security Architectural Design

Listing 5 Code for Timestamping

```
<S:Envelope     xmlns:S="http://www.w3.org/2001/12/envelope"
   xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
<S:Header>
                    <wsu:Timestamp>
                            <wsu:Created>2009-09-15T06:27:00Z</wsu:Created>
                            <wsu:Expires>2009-09-15T06:37:00Z</wsu:Expires>
                            <wsu:Received Actor="http://telemedicine.com/" Delay="60000">
                     2009-09-15T06:30:00Z
                            </wsu:Received>
                    </wsu:Timestamp>
                    <!--Other Header Details-->
          </S:Header>
          <S:Body>
          <!--SOAP Message here-->
          </S:Body>
</S:Envelope>
```

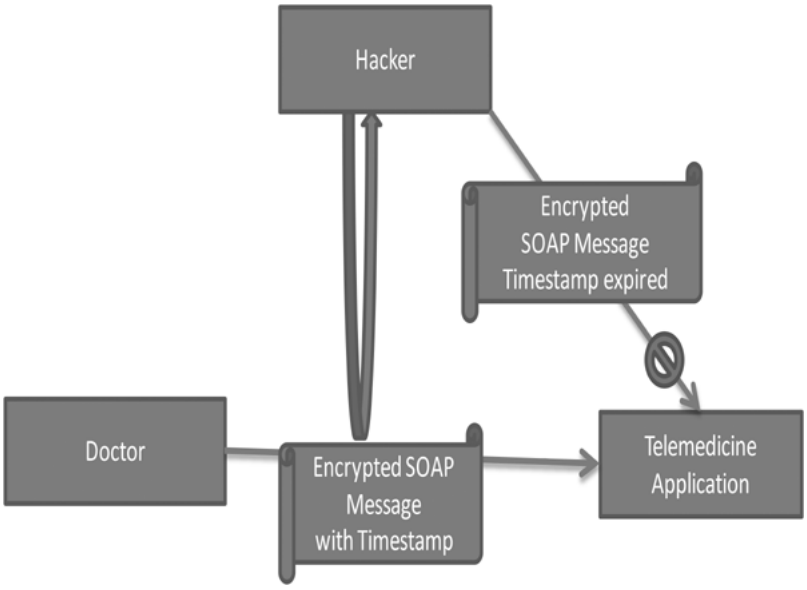

Fig. 5 Proposed Security Architectural Design

Table 2 Scenarios for Timestamping

	SOAP Message	Encrypted SOAP Message	Message Created at	Message Expired at	Current Time	End Result
Without Timstamping (Doctor)	getPatient(”abc”)	Encrypted with digital signature	N/A	N/A	N/A	Message executed successfully
Without Timstamping (Intruder)	Encrypted Message recorded to replay	(Recorded) Encrypted with digital signature	N/A	N/A	N/A	Recorded Message executed successfully
With Timstamping (Doctor)	getPatient(”abc”) with timestamp	Encrypted with digital signature	2009-09-15T06:27:00Z	2009-09-15T06:37:00Z	2009-09-15T06:32:00Z	Message executed successfully before expiry time
With Timstamping (Intruder)	Encrypted Message recorded to replay	(Recorded) Encrypted with digital signature	2009-09-15T06:27:00Z	2009-09-15T06:37:00Z	2009-09-15T06:48:00Z	Recorded Message rejected as time is expired.

4 Experiments and Results

”In this section, we present the context in which the experiment examines the assumptions that are required to maintain the consistency and dynamics of the proposed security architecture. Table 3 summarizes the experimental results obtained by the implementation of our proposed technique. Column *Actor*, represents an actor who is the user of the telemedicine application, column *CRUD Operations(Encode)* describes the operation called by an actor who has certain rights. Column *CRUD Data* displays patient’s data, column *Encryption + Signature* describes the encrypted CRUD data along with the signature and the column *Security Type* shows the type of security implemented on prescribed actor depending on permissions. Finally, the column *Decryption* shows 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.

Table 3 Results of proposed security Architecture

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

4.1 Security Scenarios

For the sake of brevity and without loss of generality, in this section we consider different scenarios of security. These scenarios are based on obtained result from our prototype (Table 2):

Scenario 1 (No Security Implemention): In simple SOA implementation without the security, there is a risk that intruder can access the confidential information by simple capture of network traffic.

Scenario 2 (Message is Encrypted to Protect Confidential Data of a Patient): To avoid the access of confidential information to intruder, we have implemented the encryption to secure the SOAP messages during communication between Telemedicine System and its client application for doctors. However, intruder can use the same encryption mechanism to generate SOAP messages and exploit the functionality of Telemedicine System.

Scenario 3 (Message is Signed with the Digital Signature for Authentication and Authorization): For authentication and authorization of the doctor, we have used PKI infrastructure to digitally sign the SOAP messages to ensure that the messages are coming from the real and registered doctor, who has the access to the system. Although intruder cannot decrypt and change the message but still there is a chance to capture and record the signed encrypted message and replay the same message to exploit the functionality of the system. We call this replay attack.

Scenario 4 (The Message is Time stamped to Prevent Replay Attack): To prevent the system from replay attack, we have used the time stamping. In time stamping, we append the creation and expiry time with each message. As recording and replaying of message take some time therefore, within that time frame, the message will be expired, thus the expired messages will be rejected by the Telemedicine System.

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 [13] 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 [14] 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. [15] 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. [16] 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 [7] 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 rep-

resents 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, the application of similar efficient security techniques can be explored in cloud computing.

References

1. Xiangping Chen, Gang Huang, and Hong Mei. Towards automatic verification of web-based soa applications. In *APWeb*, pages 528–536, 2008.
2. Nelly A. Delessy and Eduardo B. Fernandez. A pattern-driven security process for soa applications. *Availability, Reliability and Security, International Conference on*, 0:416–421, 2008.
3. Asadullah Shaikh, Muhammad Misbahuddin, and Muniba Shoukat Memon. A system design for a telemedicine health care system. In *IMTIC*, pages 295–305, 2008.
4. Wail M. Omar and A. Taleb-Bendiab. Service oriented architecture for e-health support services based on grid computing over. *Services Computing, IEEE International Conference on*, 0:135–142, 2006.
5. Asadullah Shaikh, Muniba Memon, Nasrullah Memon, and Muhammad Misbahuddin. The role of service oriented architecture in telemedicine healthcare system. *Complex, Intelligent and Software Intensive Systems, International Conference*, 0:208–214, 2009.
6. MSDN. X.509 technical supplement, Accessed December.24,2009 [Online]. `http://msdn.microsoft.com/en-us/library/aa480610.aspx`.
7. H.F. Yamany and M.A.M. Capretz. Use of Data Mining to Enhance Security for SOA. In *Convergence and Hybrid Information Technology, 2008. ICCIT'08. Third International Conference on*, volume 1, 2008.
8. Giovanni Satoshi Bob Atkinson, et al. Web services security (ws-security), copyright © 2002-2002 international business machines corporation, microsoft corporation, Accessed December.24,2009 [Online]. http://www.cgisecurity.com/ws/ws-secure.pdf.
9. Apache. Apache Axis, Accessed December.24,2009 [Online]. `http://ws.apache.org/axis`.
10. Apache. Apache WSS4J, Accessed December.24,2009 [Online]. `http://ws.apache.org/wss4j`.
11. Organization for the Advancement of Structured Information Standards(OASIS). Web services security technical committee, Accessed April.3,2009 [Online]. `http://www.oasis-open.org/committees/tchome.php?wgabbrev=wss`.

12. Asadullah Shaikh, Aijaz Soomro, Sheeraz Ali, and Nasrullah Memon. The security aspects in web-based architectural design using service oriented architecture. In *13th International Conference on Information Visualisation, IV 09, 15-17 July 2009, Barcelona, Spain*, pages 461–466, 2009.
13. C. Phan. Service Oriented Architecture (SOA)-Security Challenges and Mitigation Strategies. In *IEEE Military Communications Conference, 2007. MILCOM 2007*, pages 1–7, 2007.
14. X. Larrucea and R. Alonso. ISOAS: Through an independent SOA security specification. In *Proceedings of the Seventh International Conference on Composition-Based Software Systems (ICCBSS 2008)*, pages 92–100. IEEE Computer Society, 2008.
15. Fumiko Satoh, Yuichi Nakamura, Nirmal K. Mukhi, Michiaki Tatsubori, and Kouichi Ono. Methodology and tools for end-to-end soa security configurations. In *SERVICES '08: Proceedings of the 2008 IEEE Congress on Services - Part I*, pages 307–314, Washington, DC, USA, 2008. IEEE Computer Society.
16. Robert Bunge, Sam Chung, Barbara Endicott-Popovsky, and Don McLane. An operational framework for service oriented architecture network security. In *HICSS '08: Proceedings of the Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, page 312, Washington, DC, USA, 2008. IEEE Computer Society.