Design and Prototyping of Web Service Security on J2ME based Mobile Phones

Ti-Shiang Wang

Nokia Research Center
5 Wayside Road, Burlington, MA 01803, USA.

Abstract: One of the main objectives in this paper is to investigate how to manipulate the Simple Object Access Protocol (SOAP) message and place security functions in the header of SOAP message. Here, we will present the design and implementation of web service security application on Java 2 Micro Edition (J2ME) based mobile devices. Basically this prototyping includes two-stage approach. In the first stage, we study the concept of proof in implementation of web services security on the IBM laptop using IBM WebSephere Studio Device Developer (WSDD V 5.6) IDE [1]. In addition we import kXML/kSOAP APIs to process SOAP message and use Bouncy Castle's API [2] supporting cryptographic algorithms for security implementations. In this paper, the security functions we present here include five tasks: non-security, data digest, data encryption using symmetric key, data encryption using asymmetric key, and digital signature. At each task, we will discuss its corresponding design, SOAP header message, time performance, and return results in emulator. Based on the expected results from the first stage, in the second stage, we use Nokia 6600/3650 mobile phones as target mobile devices to test our application and evaluate performance at each task. Finally we will share our experience and lessons on this work in the conclusion and do the demonstration using Nokia 3650 mobile phone in the conference.

1 Introduction

As mobile phone becomes a commodity that almost everyone will own one mobile phone but may not have the traditional landline, it is very likely that the mobile phones will replace PDA (Personal Digital Assistant) devices in some applications. The web services are services provided over Internet or intranet using standardized XML messages to exchange information among different nodes. In addition, web service is not tied to any platforms or programming languages, which may need extensive technical skill. Also, eXtensible Markup Language (XML) remote procedure calls (XML-RPC), SOAP [3] or even HTTP can be used to implement the messaging of web services. On top of the transport methods, web services use Web Services Description Language (WSDL) to define the service provided by service applications so requester and application provider can communicate each other regardless of programming language or platform. Because web services are self-describing, discoverable, and independent to any platforms, it can support automated application integration and help to improve the development process.

To illustrate the thoughts of implementing web service security, IBM WebSephere Studio Device Developer (WSDD V 5.6) IDE, which is a J2ME development tool, which supports automated stub generator and other advanced features, is considered in this paper. That is, we use IBM WSDD to generate prototype files called "*stubs*" and continue developing codes based on the generated files. The "*stubs*" are generated based on WSDL file from remote server. It contains the methods to process necessary parameters and arguments to access remote services. The "*stubs*" may not have complete codes but it serves as a base for further development. The ultimate goal is using web services to build an application-centric web, which has less human interaction involved. Thus, in this paper we will only focus the discussion on client-server web services security implementation rather than enterprise web services, which will be part of our future works. For manipulating SOAP message, though JSR 172 web services specification also supports access to remote SOAP/XML based web services and parsing XML data on the J2ME platform [4], it is not possible for J2ME mobile devices with limited processing power to include all JAXP functionalities. In addition, current JSR 172 specification does not support SOAP message header handler.

The kXML [5][6] is a project to provide XML pull parser for J2ME based mobile devices. It supports XML namespace, and XML writing. These APIs have ability to process SOAP message using XML parser engine from kXML. kXML/kSOAP API (an open-source J2ME XML and SOAP parser). In this work, both kXML and kSOAP have to be included in the java classpath to provide the functionalities of process SOAP messages. To implement the security functions in the SOAP header, W3C has suggested adding these security tags into SOAP header as its security extensions. This work will follow the recommendation of W3C to add security information into SOAP header. As far as cryptographic algorithms used for mobile devices [7] are concerned, we test and use Bouncy Castle's cryptographic API, which is an open-source Java cryptographic algorithm API for J2ME mobile devices. In this paper, five tasks of security function are to implement: no-security, digest, encryption with symmetric key, encryption with asymmetric key, and digital signature. At each task, we will show its SOAP message, demonstration of result, and time performance. For the web server we used for this demonstration, we adopt temperature web service provided by Xmethods [8][9]. As we mention, in this work, we focus on the implementations of security function at client side, that is, the mobile phone or user side. Then in the last section, we will draw a conclusion and discuss some future works.

2 Architecture and Implementation

In the client side, developer can use IDE or automated tool to generate stub, a prototype or template file to access web services based on the WSDL file. In previous version of IBM WSDD (Version 5.5), which supports both Document-style and RPC-style web services and the IDE can help us to generate ***Temperature_Stub.java*** file as a way to automate the application development. However, for the WSDD 5.6 version, only document-style web services are to support. Thus we use document-style temperature WSDL for our implementation in the client side. Figure 1 shows the generic architecture for our work. In the first stage, we use laptop and IBM WSDD tool kit and emulator to demonstrate the concept of this implementation. Then in the second stage, we use Nokia 6600/3650 mobile phone as the client component.

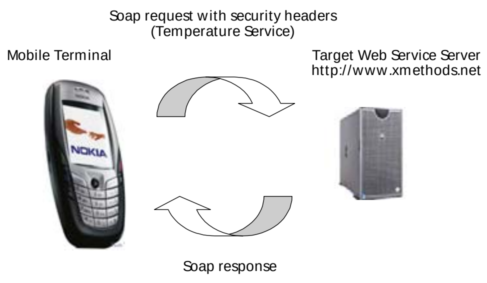


Fig. 1. Proposed web service architecture using mobile phone

There are two modules, *cryptographic algorithm module (CAM) and SOAP message parser module (SMPM)*, required to implement web services security. The CAM includes following files:

- ***Encryptor.java***: an abstract class to define the interfaces of encryption and decryption algorithm. The "*Encryptor*" class acts as a parent class for all security classes. As an abstract class, the real implementation needs to be done after inheriting from it, so that further security extensions can be added or integrated under the "*Encryptor"* class.
- ***DigestEncryptor.java***: the implementation of data digest algorithm. This class implements the abstract method of ***Encryptor.java*** file;
- ***SymmetricEncryptor.java***: the implementation of secret key data encryption algorithm. This class implements the abstract method of ***Encryptor.java*** file;
- ***AsymmetricEncryptor.java***: the implementation of public key data encryption algorithm. This class implements the abstract method of ***Encryptor.java*** file;
- ***DSEncryptor.java***: the implementation of digital signature algorithm. This class implements the abstract method of ***Encryptor.java*** and composite ***DigestEncryptor*** and ***AsymmetricEncryptor*** classes.

For the SMPM, it includes following files:

- ***SoapEnvelope.java***: a SOAP message parser without security extension;
- ***SecSoapEnvelope.java***: a SOAP message parser with security extension;
- ***HttpTransportTest.java***: Responsible for delivery of SOAP message.

The SOAP message handler has two classes: ***SOAPEnvelope*** and ***SecSOAPEnvelope***. ***SOAPEnvelope*** is the modified version of original class from kSOAP package and the ***SecSOAPEnvelope*** adds the header and body process capability so that security and cipher data can be replaced in the SOAP message. In addition, to interact between user/client side and server side, there are two java application files implemented as well. ***SecTemperature_Midlet.java*** is the main class for J2ME application and ***Temperature_Stub.java*** is the interface between ***SecTemperature_Midlet.java*** and other modules mentioned above. In this work, we have implemented a temperature query web services application on J2ME based mobile phone. User of the mobile application will be asked for ***zipcode*** and the selection of desired encryption algorithm. There are five different cryptographic algorithms available for selection, including no-security (Task 1), data digest (Task2), data encryption with symmetric key (Task 3), data encryption with asymmetric key (Task 4), and digital signature (Task 5).

After user enters ***zipcode*** and chooses one of the encryption algorithms, the application will take the ***zipcode*** (for example, 01803) as input and encrypt this ***zipcode*** based on the selected encryption algorithm. Then, the application will generate a cipher value and attach this value to the body on SOAP message. In addition, the cryptography algorithm name and the web services security tags will be added to the SOAP header and body. All the name spaces and XML tags in web services security have been defined in the standard of OASIS Web Services Security [10]. It should be noticed that the original ***zipcode*** is not replaced with cipher text because the existing of plaintext and cipher value can help us to verify our implementation of cryptographic algorithms and get the result (i.e., temperature degree) from the web server.

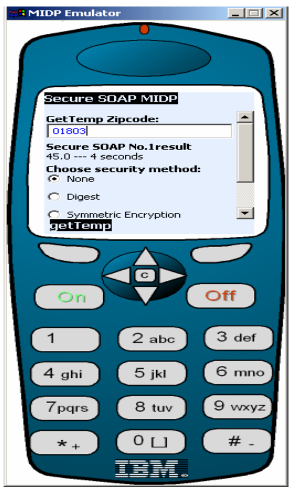
3 Detail Task Implementation and Results

In this section we discuss 5 tasks of our security implementation. In addition, the corresponding SOAP message, and demonstration are presented and general time performance is introduced as well.

3.1 Task 1:No-Security

In this task, we study what the SOAP message looks like without adding any security function using IBM WSDD as the starting point for the rest of the following tasks. The following SOAP message shows the regular SOAP message without security extension. Figure 2 shows the snapshot and results from emulator. The time of result responded from server side is ~4 seconds.

```
    <SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
      <SOAP-ENV:Header />
      <SOAP-ENV:Body SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
       <getTemp xmlns="urn:xmethods-Temperature" id="o0" SOAP-ENC:root="1">
        <zipcode xmlns="" xsi:type="xsd:string">01803</zipcode>
       </getTemp>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```



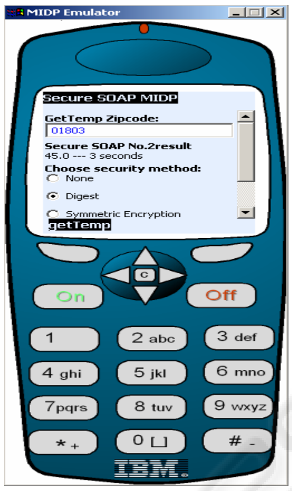
Secure SOAP MIDP

GetTemp Zipcode:
01803

Secure SOAP No.1result
45.0 --- 4 seconds

Choose security method:
[✓ None]
Digest
Symmetric Encryption

Fig. 2. Snapshot of no-security implementation.



Secure SOAP MIDP

GetTemp Zipcode:
01803

Secure SOAP No.2result
45.0 --- 3 seconds

Choose security method:
None
[✓ Digest]
Symmetric Encryption

Fig. 3. Snapshot of data digest implementation.

3.2 Task 2: Data Digest

Data integrity is to ensure that the data is from original sender without any modification by unauthorized users. It is important to understand that both sender and receiver choose a key before creating or verifying the digest data. Once receiver receives the data, the digest value from the received plaintext is generated using a pre-determined key to both sides. The new digest value generated at receiver side will be compared with the digest data sent from sender side. Both of them should be the same, otherwise the data sent from sender side possibly have been modified. The following SOAP message illustrates the SOAP message with data integrity security extension.

```
   <SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header>
     <wsse:Security xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
1.0.xsd">
      <ds:DigestMethod xmlns="http://www.w3.org/2000/09/xmldsig#sha1" Algorithm="SHA256" />
      <ds:SignedInfo xmlns="">
       <ds:CanonicalizationMethod Algorithm="http://www.w3c.org/2001/10/xml-exc-c14n#" />
       <ds:Reference URI="#zipcode">
        <ds:Transforms Algorithm="http://www.w3c.org/2001/10/xml-exc-c14n#" />
       </ds:Reference>
      </ds:SignedInfo>
     </wsse:Security>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
     <getTemp xmlns="urn:xmethods-Temperature" id="o0" SOAP-ENC:root="1">
      <zipcode xmlns="" xsi:type="xsd:string">01803</zipcode>
     </getTemp>
```

```
  <ds:CipherValue
xmlns="http://www.nokia.com/nrc/boston/security/">d085119a2d49e7099ebf9f3fd5801bf9bebbaf77
  b2be07805577cec7598b9aa1</ds:CipherValue>
   </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

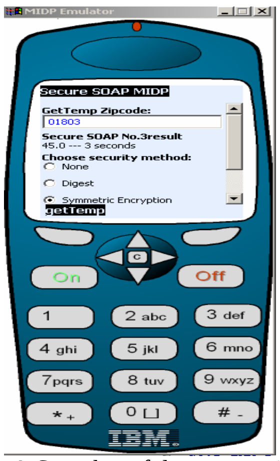
In this SOAP message, several lines have been added to the SOAP header. The **<*wsse:Security*>** defines the standard of the web services security for this application. The **<*ds:DigestMethod*>** defines the cryptographic algorithm used in this task. The XML tag **<*ds:Reference*>** is used to indicate what data will be encrypted from the SOAP body. Receiver is able to use this tag to re-construct the original plaintext. Here, we only encrypt the ***zipcode*** but not the whole body of SOAP message. There are also some lines adding to the BODY of SOAP message. The **<*ds:CipherValue*>** is the data digest value calculated after entering zipcode by the user. In this case, the SHA cryptographic algorithm is implemented. Please note that the digest data is placed outside **<*getTem*>** tag because there will be no response if we insert other data into the tag defined by WSDL to receive request information. Figure 3 shows the result we get from WSDD emulator and the time to receive the result is ~ 3 seconds.

3.3 Task 3: Data Encryption Using Symmetric Key

Symmetric encryption takes plaintext as input and use secret key to encryption the plaintext to a cipher text. In this project, we implemented the AES encryption, which has 128-bit block size of plain text. Compared with previous data digest algorithm, this task experiences more complicated since padding issue on the input message and the key needs to be considered. The following SOAP message illustrated the implementation of symmetric encryption in SOAP message.

```
  <SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Header>
    <wsse:Security xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss
  -wssecurity-secext-1.0.xsd">
     <ds:DigestMethod xmlns="http://www.w3.org/2000/09/xmldsig#sha1" Algorithm="AES" />
     <ds:SignedInfo xmlns="">
     <ds:CanonicalizationMethod Algorithm="http://www.w3c.org/2001/10/xml-exc-c14n#" />
      <ds:Reference URI="#zipcode">
       <ds:Transforms Algorithm="http://www.w3c.org/2001/10/xml-exc-c14n#" />
      </ds:Reference>
     </ds:SignedInfo>
    </wsse:Security>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <getTemp xmlns="urn:xmethods-Temperature" id="o0" SOAP-ENC:root="1">
     <zipcode xmlns="" xsi:type="xsd:string">01803</zipcode>
    </getTemp>
    <ds:CipherValue
xmlns="http://www.nokia.com/nrc/boston/security/">dec921ebadb8dbec94a1340f532a7
  ef6</ds:CipherValue>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The difference between the SOAP message for symmetric key encryption task and data digest encryption is the attribute of <ds:DigestMethod> XML tag has been replaced with "AES" to reflect the change of cryptographic algorithm. Also, the cipher value in the body of SOAP message is replaced with corresponding cipher data. Figure 4 shows the snapshot of symmetric key encryption application on emulator. In this case, ~3 seconds is required to the result sent back from the server.



MIDP Emulator

Secure SOAP MIDP

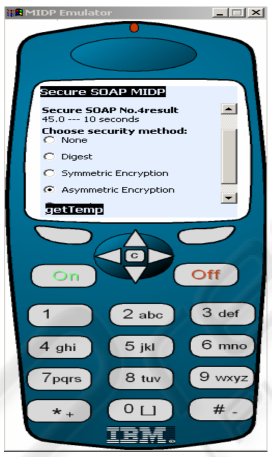
GetTemp Zipcode:
01803

Secure SOAP No.3result
45.0 --- 3 seconds

Choose security method:
- None
- Digest
- [✓ Symmetric Encryption]

getTemp

Fig. 4. Snapshot of data encryption using symmetric key.



MIDP Emulator

Secure SOAP MIDP

Secure SOAP No.4result
45.0 --- 10 seconds

Choose security method:
- None
- Digest
- Symmetric Encryption
- [✓ Asymmetric Encryption]

getTemp

Fig. 5. Snapshot of data encryption using asymmetric key.

3.4 Task 4: Data Encryption Using Asymmetric Key

The asymmetric key encryption is also called "public key encryption" algorithm. Sender uses receiver's public key to encrypt data. The encrypted (cipher text) data (here, 01803 is used) is sent to the receiver and the receiver uses its own private key to decrypt the cipher data to original plaintext. According to our test, it will take 4 or 5 minutes to generate the key pair on Nokia 6600/3650 mobile device, even though the time required in the emulator is much shorter (~10 seconds), as shown in the Figure 5. The following SOAP message illustrates the implementation of asymmetric encryption in SOAP message.

```
    <SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
     <SOAP-ENV:Header>
      <wsse:Security xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
1.0.xsd">
      <ds:DigestMethod xmlns="http://www.w3.org/2000/09/xmldsig#sha1" Algorithm="RSA" />
      <ds:SignedInfo xmlns="">
      <ds:CanonicalizationMethod Algorithm="http://www.w3c.org/2001/10/xml-exc-c14n#" />
        <ds:Reference URI="#zipcode">
         <ds:Transforms Algorithm="http://www.w3c.org/2001/10/xml-exc-c14n#" />
        </ds:Reference>
       </ds:SignedInfo>
      </wsse:Security>
```

```
  </SOAP-ENV:Header>
  <SOAP-ENV:Body SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <getTemp xmlns="urn:xmethods-Temperature" id="o0" SOAP-ENC:root="1">
    <zipcode xmlns="" xsi:type="xsd:string">01803</zipcode>
   </getTemp>
   <ds:CipherValue
xmlns="http://www.nokia.com/nrc/boston/security/">015d8dbccb65a206ccd0cee6abfe3f344a456e204e159
b11e119c48c5b0a347018263ba8341be1872cf83e58c6922a91d2758565076099583b9e84d0c946b01b425f1
d812dfc0651c40d3fc32e35bd82fd21d066d8b28eef9134dc4c60f0bcbd3c0ae0c354987aee407a3bd0cddf2e9
0d56e4f934268b93eae71406c7aa7ec81</ds:CipherValue>
  </SOAP-ENV:Body>
 </SOAP-ENV:Envelope>
```

It is obvious that the more time consumption to get the result is experienced due to the limited processing power in the mobile phones.

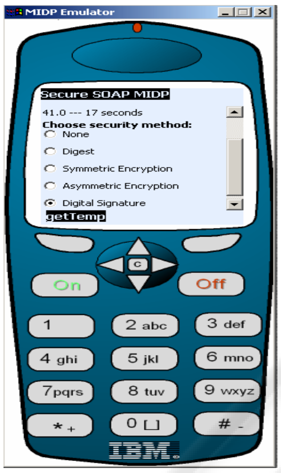
3.5 Task 5: Digital Signature

In this task, both HASH and RSA algorithms are used to implement digital signature function. The original message is calculated to a unique digest value using SHA-1 hash algorithm. Then, the digest is signed by sender's private key as a signature message. Both signature message and original plaintext data are sent to receiver side. Once the receiver receives the signature message and plaintext from the sender side, the signature message is decrypted using sender's public key at receiver side. After the signature message is decrypted to a hash message, which the is used to compare with hash message generated in the receiver side using plaintext sent from sender to check the integrity of the data. The following SOAP message illustrates the implementation of digital signature.

```
 <SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
   <wsse:Security xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-w
 ssecurity-secext-1.0.xsd">
    <ds:DigestMethod xmlns="http://www.w3.org/2000/09/xmldsig#sha1" Algorithm="DS" />
    <ds:SignedInfo xmlns="">
     <ds:CanonicalizationMethod Algorithm="http://www.w3c.org/2001/10/xml-exc-c14n#" />
     <ds:Reference URI="#zipcode">
      <ds:Transforms Algorithm="http://www.w3c.org/2001/10/xml-exc-c14n#" />
     </ds:Reference>
    </ds:SignedInfo>
   </wsse:Security>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <getTemp xmlns="urn:xmethods-Temperature" id="o0" SOAP-ENC:root="1">
    <zipcode xmlns="" xsi:type="xsd:string">01803</zipcode>
   </getTemp>
   <ds:CipherValue
xmlns="http://www.nokia.com/nrc/boston/security/">17638ec2a1d3a52a40ec6cd06f2242287756e84c51eb
3cb1ca75d4cd678ec92b890a92f222c8a907de81dce87caec1a1cbdf02b0d02cba5e5f9d13d30bf48f3c926222
e9d4fd568f1b1c6f01cf4933c3087427be3502f0b141d7ed70afe7364744d1af5587d7f9fb6fe11a494a3b48432
3ed403851aeccea0eae62a1edd57960
  </ds:CipherValue>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 6 shows the application running digital signature function. Because it needs to generate private-public key pair as asymmetric algorithm, thus it takes more time (~17 seconds) than any one of the tasks in this paper.

.



Secure SOAP MIDP

41.0 --- 17 seconds

Choose security method:

- ( ) None
- ( ) Digest
- ( ) Symmetric Encryption
- ( ) Asymmetric Encryption
- [✓ Digital Signature]

getTemp

Fig. 6. Snapshot of digital signature.

4 Conclusion

As wireless networks have been widely deployed and mobile phones are popular year after year due to cost reduction, people reply on the use of mobile phone more in their daily life for specific services through the web server and Internet. It is well-known that security is one of key components that should be implemented in the web services and still remains as one of the challenging issues so far. In this paper, we have prototyped five security tasks in the client side (or mobile phone) using IBM WSDD and demonstrated these tasks using Nokia 6600/3650 series mobile phones. We also presented the corresponding SOAP message communicated between client and server sides. We also evaluated and compared the time performance of each task. Based on our design and implementation, it takes more time to generate cipher text for asymmetric key encryption and digital signature than other tasks. Due to the limited processing power of current mobile phones, when the application is running on real mobile phones, we experienced more time delay to get the result from the server side than we expected in the order of minutes. Thus how to improve the time performance at client side to meet the practical need of people is part of future work. With this time performance obtained from all the tasks in this paper, the application using web service security using mobile devices needs to consider carefully and appropriately from both technical/technology side and business side. In addition, we are also investigating the security functions implemented in the server side and planning to integrate with existing results. Furthermore, some possible applications and implications using web services using mobile phones are under investigation as well.

Acknowledgements

The author is grateful to Edmond Chang’s contribution on this project when he works as an intern in Nokia Research Center. In addition, the author also thanks Dr. Senthil Sengodan and Dr. Tat Chan for their encouragement, discussion and comments on this work as well.

References

1. IBM WSDD, http://www-306.ibm.com/software/wireless/wsdd/
2. Bouncy Castle, http://www.bouncycastle.org/index.html.
3. M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, and H. Nielsen, SOAP Version 1.2 Part 1: Messaging Framework, http://www.w3.org/TR/2003/REC-soap12-part1-20030624/, June, 2003.
4. Jon Rllid and Mark Young, Sun Microsystems, J2ME Web Services 1.0 final Draft, http://www.jcp.org/en/jsr/detail?id=172, October 15, 2003.
5. kXML project - http://www.kxml.org
6. Enhydra.org, http://kxml.objectweb.org/project/aboutProject/index.html
7. Enterprise J2ME: Developing Mobile Java Applications, Michael Juntao Yuan, ISBN: 0131405306, Prentice Hall Publisher, 2003.
8. Xmethods.Inc, http://www.xmethods.net/, 2004.
9. WSDL files for temperature, http://www.xmethods.net/sd/2001/TemperatureService.wsdl.
10. OASIS, Web Services Security: SOAP Message Security 1.0 (WS-Security 2004), http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf, March 2004.