Rheinisch-Westfälische Technische Hochschule Aachen

Lehrstuhl für Informatik V
Prof. Dr. Matthias Jarke

Security Aspects Analysis in Mobile Web Service Provisioning

Master Thesis

Kiran Kumar Pendyala
Matriculation number: 248254

September 29th 2006

First Supervisor:	**Prof. Dr. Matthias Jarke** **Lehrstuhl für Informatik V, RWTH Aachen**
Second Supervisor:	**Prof. Dr. Wolfgang Prinz** **Lehrstuhl für Informatik V, RWTH Aachen**
Advisor:	**M. Sc. Satish Narayana Srirama** **Lehrstuhl für Informatik V, RWTH Aachen**

Statement

I guarantee that this thesis is done independently, with support of the Informatik V department at RWTH Aachen University and no other unmentioned helping resources are used.

Aachen, September 29, 2006

(Kiran Kumar Pendyala)

Abstract

As pervasive mobile data applications are becoming ubiquitous in parallel with the fast developing and easily readable web services (WS), the ability to provide secure and reliable communication across mobile web service applications became utmost important. Even though a lot of standardized security specifications and implementations exist for web services in the wired networks, not much has been standardized in the wireless environments. This thesis report addresses some of the critical challenges in providing WS-Security to the mobile web services domain.

The thesis first addresses security challenges for mobile web services, then discusses existing standards in the wired and wireless web service environments. It further tries to propose basic security requirements in the mobile web services domain. Then the thesis presents design models to achieve the WS-Security with various architectures and describes the implementation model and its details in an elaborative way. Finally the thesis presents evaluation details including performance model, generic mobile WS-Security evaluation analysis, and Mobile Host performance analysis and evaluation.

In summary, the thesis tries to show that at the least WS message-level security is achievable on mobile devices and on our Mobile Host. The thesis also draws out various possible design scenarios, especially Single Sign-on, for achieving WS end-point security with Mobile Host in the picture. Further, the thesis concludes with the summary of best possible scenario to achieve WS-Security in mobile web services domain and closes out explaining the future research directions in the mobile web service security domain.

Table of Contents

1 Introduction

1.1 Motivation

As mobile and wireless applications are becoming ubiquitous in conjunction with the fast growing web services and related domains, the ability to provide secure and reliable communication even in the vulnerable and volatile mobile ad-hoc topologies is vastly becoming necessary. Web services are simple and versatile XML-based communications, described by an XML-based grammar, called Web Services Description Language (WSDL), which binds abstract service interfaces, consisting of messages, expressed as XML Schema, and operations, to the underlying wire format.

Unfortunately in the pervasive networks, where a central administration is not always possible, the security implementation is extremely hard and challenging with the advent of easily readable web services. Secure provisioning of mobile web Services needs proper Identification mechanism, Access control, Data Integrity and Confidentiality. It also requires policies and trust relations to be established between users as well as between users and service providers.

Even though a lot of standardized security specifications, protocols and implementations like WS-Security [5], SAML [11] etc., exist for web services in traditional wired networks, not much has been explored and standardized in wireless environments, with feasibility, till date. Some of the reasons for this poor state might be the lack of active commercial data applications due to the extremely limited resources of the mobile terminals like memory, processing capability, and the low transmission rates of the wireless mediums. My thesis contributes to this work and tries to bridge this gap, with main focus at realizing some of the existing security standards even in the mobile web services domain.

1.2 Overview

The thesis report starts with discussing briefly the base project “Mobile Web Service Provisioning“ [1]; the security issues, challenges and difficulties in inducing the current existing security standards; and the realization proposals of the basic security issues in mobile web services domain. Common security breaches like Man-in-the-middle attacks, Denial of Service Attacks are considered along with the web services security breaches such as illegal access to services, Spoofing, Tampering, Reply Attacks etc. Further, the thesis describes the existing standards, followed by design, implementation and evaluation of WS-Security in mobile web service domain. The paper is organized as follows:

Section 2 addresses the complete state-of-the-art objects in the following order. First, the basic concept and applications of mobile web services are addressed along with Mobile Host. Second, the QOS challenges for web services and wireless environments are addressed which covers most of the existing security aspects and the relevant scalability aspects. Further, I have discussed the existing standards of web services and wireless environments which primarily covers WS-Security standard along with brief

introductions about Liberty Alliance and Open Mobile Alliance projects. Finally, the section ends with discussing some of the existing mobile technology standards.

Section 3 addresses the basic security requirements that need to be addressed in our Mobile Host to successfully deploy Web Services. This section then further presents both message-level security design models and end-point security design models including some of the existing design workflow methodologies.

Section 4 explains the mobile WS-Security implementation details. It first covers the relevant development tools and platforms such as J2ME, Sun Java Wireless Toolkit, Lightweight Bouncycastle Cryptographic API, and the adapted KSOAP2 API in detail. Then I have described the implementation model and details with a class diagram. The section ends with representing the output SOAP message structures produced with this implementation.

Section 5 addresses the mobile WS-Security evaluation details. I have started this section describing the performance model to understand various timestamps used for analysis. Then the section continues analyzing the generic experimental results and evaluation model of the complete web service cycle. Finally, Mobile Host performance evaluation model and analysis were explained to end the section.

Section 6 starts with conclusion, summarizing the thesis work, and section 7 ends this report with presenting the relevant future research directions.

2 State of the art

This section presents an overview of the state-of-the-art technologies related to this master thesis. The scope of the thesis topic will be outlined through the study of literature.

2.1 Mobile Web Services

2.1.1 Introduction

Web services are software components that can be accessed over the Internet using established web mechanisms and protocols such as SOAP [13] and HTTP [3]. Public interfaces of web services are described using Web Service Description Language (WSDL) [14]. Examples of web services range from simple requests, such as stock quotes or user authentication, to more complex tasks, such as comparing and purchasing items over the Internet.

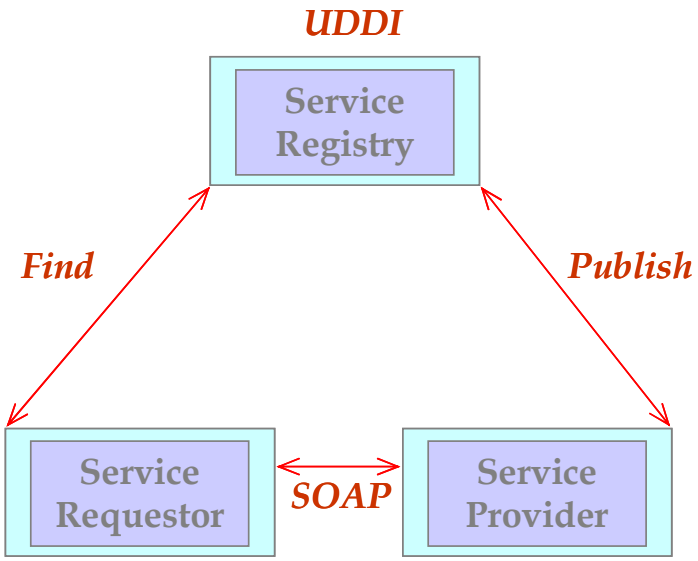


Figure 1: Basic Web Services Architecture

The basic Web Services Architecture consists of three basic components: the Service Provider, the Service Requestor (Client), and the Service Registry (UDDI [15]). The Service Provider basic functionalities are owns a service, registers a service at Service Registry and provides services to the Service Requestor. The Service Requestor functionalities include searching for a service at Service Registry and invoking services from the Service Provider. The Service Registry is a repository where you can register a service description and you can find a service description.

The basic web services component technologies are Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL) and Universal

Description, Discovery and Integrity (UDDI) protocol. SOAP is a simple lightweight XML messaging protocol by which applications can exchange information over various transport layer protocols such as HTTP, BEEP etc. Its typical structure contains mandatory SOAP Envelope, optional SOAP Header and mandatory SOAP Body. The Envelope is the root element. Body contains the mandated information to be sent to the intended receiver. As this proposal concerns security as a prime aspect, the optional SOAP header where one can provide Authentication information, Transactional & Payment details will be considered as an essential component.

With the introduction of Third and Interim Generation mobile communication technologies in the cellular domain like UMTS [22], GPRS/EDGE [23], the speed of wireless data transmission has increased significantly. Also processing power and device capabilities of mobile phones have increased drastically, thereby enabling better applications and usage of mobile devices in different application domains.

Combining these developments it is a logical next step to turn mobile devices into wireless web service requestors (clients). This enables communication via open XML web service interfaces and standardized protocols also on the radio link, where today still proprietary and application- and terminal-specific interfaces are required. Mobile web service clients lead to manifold opportunities to mobile operators, wireless equipment vendors, third-party application developers, and end users. It is easy to imagine that in the future mobile applications based on web service clients will generate a large percentage of all web service requests, and the first such solutions are currently appearing on the market. [[9], [10]]

However, in a sense, this role of mobile web service clients is still basic and the combination of cellular and web services domains would only be completed if it would become feasible to also offer some sort of standard web service providers on small mobile devices. My basic requirements study is based on one such prototype, the Mobile Host, which acts as Mobile Web Service Provider.

2.1.2 Mobile Web Service Provider

The Mobile Host, Mobile Web Service Provider, was designed and tested on a SonyEricsson P800 Smart Phone and was developed in PersonalJava. The footprint of the fully functional prototype is only 130 KB. The Mobile Host has been developed as a web service handler built on top of a normal web server. The web service requests sent by HTTP tunneling are diverted and handled by the web service handler. The evaluation of Mobile Host showed that service delivery as well as service administration can be done with reasonable ergonomic quality by normal mobile phone users. As the most important result, it turns out that the total WS processing time at the Mobile Host is only a small fraction of the total request-response time (<10%) and rest all being transmission delay. The following Figure 2 shows the basic architectural setup of the Mobile Host. A detailed discussion of implementation and evaluation details of this Mobile Host [40] is beyond the scope of this thesis.

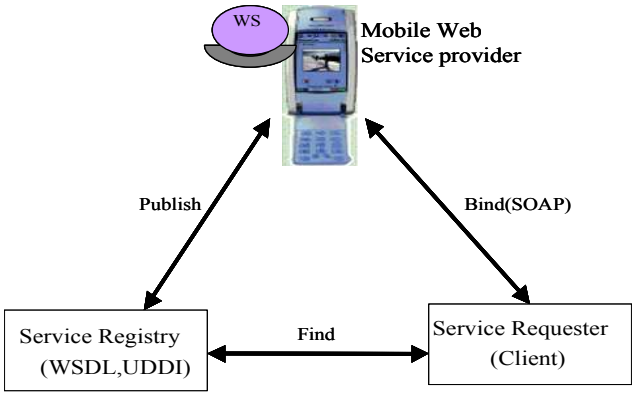


Figure 2 : Basic architectural setup of Mobile Host

The Mobile Host, once commercially viable, can serve some useful services. As a Mobile Host, the mobile terminal becomes a multi-user device where the owner/carrier of the device can work in parallel with users of the web service without explicit effort on his/her side. From a commercial viewpoint, there is a reversal of payment structures. While traditionally the information-providing web service client has to pay to upload his or her work results to a stationary server (where then other clients have to pay again to access the information), in the Mobile Host scheme responsibility for payment shifts to the actual clients -- the users of the information/services provided by the Mobile Host. Another commercial aspect is the possibility for small mobile operators to set up their own mobile web service business without resorting to stationary office structures, thus going one step further in the move from central to P2P architectures [40].

Of course, this additional flexibility generates a large number of interesting research questions which need further exploration, the immediate topics of interest being the security implications and the means of achieving them of this approach.

2.2 Security Challenges for Web Services and Wireless Environments

This section aims at analyzing most of the security issues in web services and wireless environments. The following subsections discuss in detail the security aspects, the security threats- vulnerabilities and their countermeasures of both web services and wireless domains.

2.2.1 Background for Security aspects

As web services use message-based technologies for complex transactions across multiple domains, traditional security processes fall short. Potentially, a Web-service message can traverse through several intermediaries before it reaches its final destination.

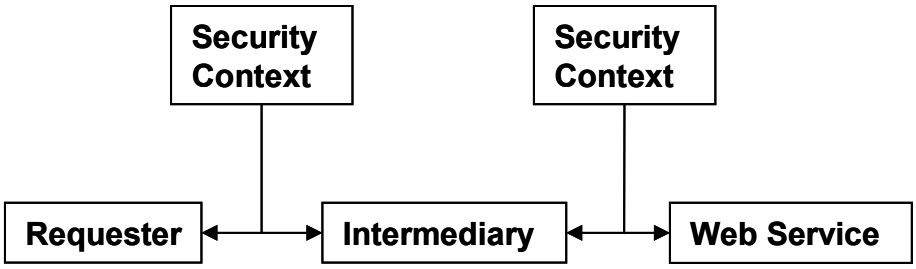


Figure 3 : Point-to-Point Security Paradigm

The Point-to-point security represented in figure 3 reflects the traditional security mechanism. In this case, the security context will change from or pertained to hop-to-hop when a secured message is transmitted from source to destination. HTTPS and SSL [65] are some of the examples of this kind. But this point-to-point security strategy is not recommendable for WS-Security; as web services are purely XML-based, which are easily readable, and sometimes the web service message might need to be transmitted through un-trusted intermediaries before reaching the destination.

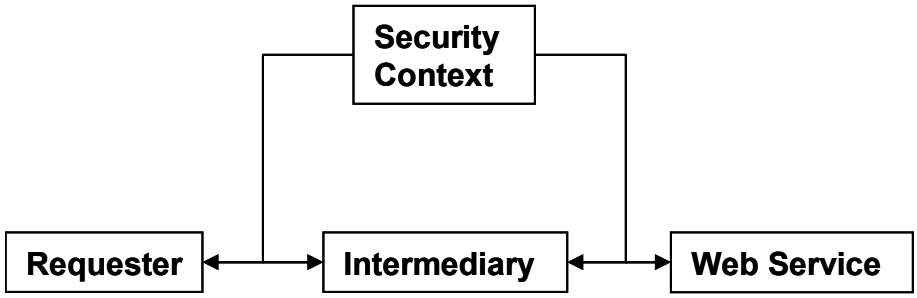


Figure 4 : End-to-End Security paradigm

Figure 4 depicts the end-to-end security strategy, where the security context holds till a message reaches its destination. To be summarized further, when the source transmits the message along with the security context using this end-to-end security mechanism, only part of the security context that needs will be accessible to the intermediaries. This partly information is sufficient enough for the intermediate hops to check the messages' basic security parameters such as authenticity and routing to the next immediate hop etc. The destination end can have access to the entire security context thereby achieving end-to-end security. Thus, this mechanism solves the WS-Security requirement by hiding the necessary security details from the intermediaries. Therefore, the need for sophisticated message-level security becomes a high priority and is not addressed by existing security technologies.

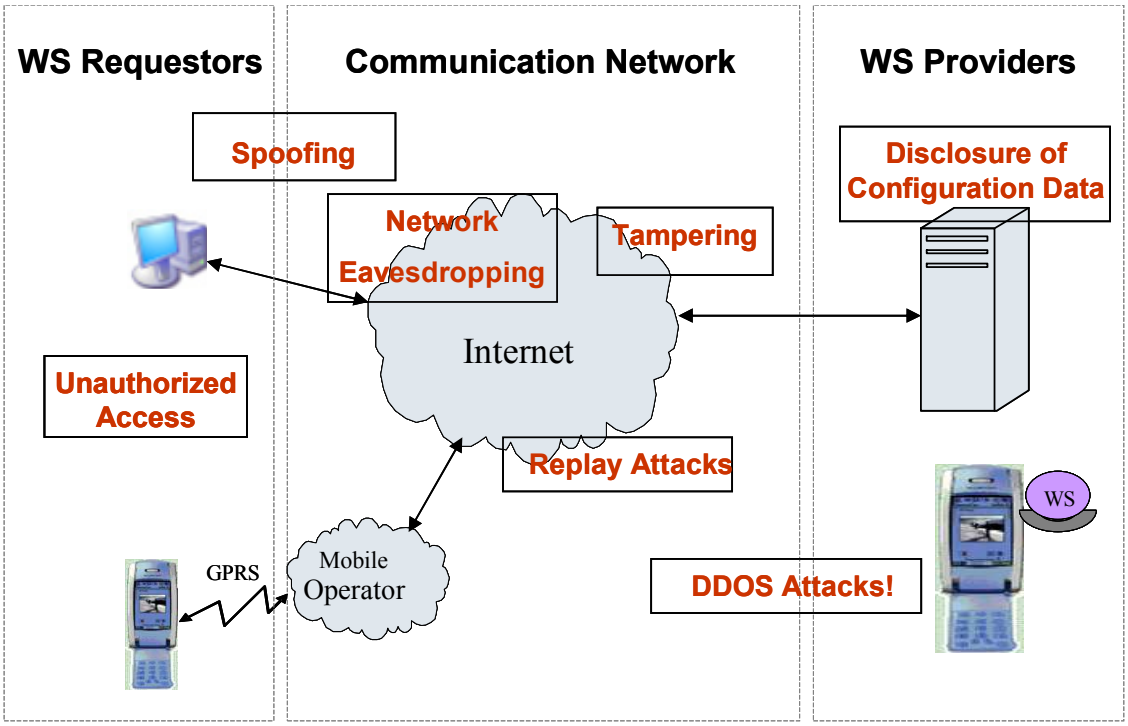


Figure 5 : Typical Security breaches in Mobile Web Services

The Figure 5 shows some of the typical security breaches [41] in web service and wireless environments. The following subsections discuss the breaches and their possible countermeasures in a step by step manner:

2.2.2 Spoofing & Unauthorized Access

Spoofing is a means of gaining access to a system by using a false identity. To accomplish this, an attacker can use fake source address that does not represent the actual source address or stolen user credentials. The purpose of spoofing would be to hide the original source of an attack or to gain access to a service as a legitimate user or host, thereby elevating sensitive privileges. Although carefully crafted spoofed packets may never be tracked to the original sender, a combination of filtering rules prevents spoofed packets from originating from your network, allowing you to block obviously spoofed packets.

Therefore, web services that provide sensitive or restricted information should authenticate and authorize their callers. Weak authentication and authorization can be exploited to gain unauthorized access to sensitive information and operations at the Mobile Host. Vulnerabilities that can lead to unauthorized access through a web service include [41]:

- No authentication used
- Passwords passed in plaintext in SOAP headers
- Basic authentication used over an unencrypted communication channel

To counter the above mentioned vulnerabilities, we should use password digests, Kerberos tickets [63] and X.509 certificates [64] in the SOAP headers for authentication. Since SOAP messages are XML-based, all the digests and certificates should be converted to plain text while exchanging via network, which in turn inherits the risk of their disclosure. To accomplish this, the sensitive credentials and authentication can be encrypted using cryptographic hashing or signatures and are need to be addressed in mobile web service provisioning. After message has been received and successfully validated, the receiver side must decide [41]:

- Does it know who is requesting the operation (Identification)
- Does it trust the caller's identity claim (Authentication)
- Does it allow the caller to perform this operation (Authorization)

There is not much WS-specific activity that takes place at this stage – just several new ways of passing the credentials for authentication. Most often, authorization tasks occur completely outside of the web service implementation, at the Policy Server that protects the whole domain. There is another significant problem here – the traditional HTTP firewalls do not help at stopping attacks at the web services. An organization would need a XML/SOAP firewall, which is capable of conducting application-level analysis of the web server's traffic and make intelligent decision about passing SOAP messages to their destination.

2.2.3 Tampering

Tampering or Parameter manipulation is referred as unauthorized modification of data or message during network travel between web service Requestor and web service Provider. Attackers have high chance of intercepting and modifying web service messages passes through several intermediate nodes before reaching their intended destination points. The potential vulnerabilities of this kind of attack are the following:

- Messages are not digitally signed
- Messages are not encrypted

In communication protocols, there are usually some mechanisms like checksum applied to ensure packet's integrity. However, this mechanism is not sufficient in case of publicly exposed web services, since checksums are relatively easy to modify and difficult to track at the receiver end. The possible solution would be to combine message digests with either cryptographic signatures or with symmetric key-encryption to ensure that any change will immediately result in a cryptographic error.

Therefore, to counter tampering attacks, the messages should be digitally signed and encrypted. The digital signature will then be used at the recipient side to verify the message to find tampering attacks if any. Furthermore, message’s timestamps will be helpful to detect the parameter manipulations in the middle at the receiver end.

2.2.4 Network Eavesdropping

Eavesdropping or sniffing is termed as an act of monitoring traffic on the network for sensitive data such as plaintext passwords or configuration information. This can be accomplished by means of simple packet sniffers placed in the path of a network. Even encrypted packets by lightweight hashing algorithms can be easily deciphered by attackers.

With web services, the threat with this kind of network eavesdropping attacks will be potentially high since SOAP messages are XML-based and easily readable. The attackers can use network monitoring software to retrieve sensitive application level data or credential information from the web service messages as they flow across the network. Vulnerabilities that can enable successful network eavesdropping include [41]:

- Credentials passed in plaintext in SOAP headers
- No message level encryption used
- No transport level encryption used

The countermeasures for these kinds of attacks would be to use strong message level encryption to protect sensitive SOAP messages. Even though transport level encryption can be achieved using SSL or IPSec which effectively protects only between two service endpoints, the approach would not work in web services scenario where SOAP messages travels through several intermediary nodes before reaching final destination. Thus, encrypting the message payload is necessary most of the times.

With message level encryption, we can encrypt the entire message being processed or only to certain parts of the message. Normally, symmetric encryption algorithms are used to encrypt bulk data, since it is significantly faster than the asymmetric ones. Asymmetric encryption is useful to protect the symmetric session keys and can be discarded once the session is established after exchanging the secret symmetric keys.

Applying encryption requires conducting an extensive setup work, since the communicating parties now have to be aware of which keys they can trust, deal with certificate and key validation, and know which keys should be used for communication. In many cases, encryption is combined with signatures to provide both integrity and confidentiality. Normally, signing keys are different from the encrypting ones, primarily because of their different lifecycles – signing keys are permanently associated with their owners, while encryption keys may be invalidated after the message exchange.

One more important issue for the above mentioned kind of attack would be Information disclosure. Information disclosure is the unwanted exposure of private data. Any sort of readable information can be very useful to the attacker. So, the message constructor should be well aware about what should be exposed and what should not.

2.2.5 Disclosure of Configuration Data

There are two main ways in which a web service can disclose configuration data. First, the web service may support the dynamic generation of Web Service Description Language (WSDL) or it may provide WSDL information in downloadable files that are

available on the web server. This may not be desirable because WSDL describes the characteristics of a web service such as method signatures and supported protocols. Second, with inadequate exception handling the web service may disclose sensitive internal implementation details useful to an attacker. Vulnerabilities that can lead to the disclosure of configuration data include [41]:

- Unrestricted WSDL files available for download from the Web server
- An unrestricted web service at Mobile Host supports the dynamic generation of WSDL and allows unauthorized consumers to obtain web service characteristics
- Weak exception handling

To counter the above potential vulnerabilities, use authorization to access WSDL files at Mobile Host and disable the documentation protocols to prevent the dynamic generation of WSDL if possible. Furthermore, handle exceptions by throwing circumspect SoapExceptions through which only minimal and harmless information will be returned back to the client.

2.2.6 Message Replay

Web service messages can potentially travel through multiple intermediate servers. With a message replay attack, an attacker captures and copies a valid message and replays it to the web service impersonating the client. The message may or may not be modified. Vulnerabilities that can enable message replay include [41]:

- Messages are not encrypted
- Messages are not digitally signed to prevent tampering
- Duplicate messages are not detected because no unique message ID is used

The most common types of message replay attacks are Man in the middle attacks where the attacker captures the messages, changes the contents, replays them to web service and Basic replay attacks where the attacker captures and copies messages, replays the same messages by impersonating the clients. The former attack mentioned can be counter measured using Encryption and Digital signatures which will prevent man in the middle attacks where the message contents are modified before being replayed.

The basic reply attacks are the difficult ones to detect because the receiving side receives the unchanged and expected message. The possible solution would be to use unique message ID or nonce, a cryptographically unique value, to detect duplicate messages at the server/receiver side. The unique message ID can be accomplished as follows. When the server responds to the client it sends a unique ID and signs the message, including the ID. When the client makes another request, the client includes the ID with the message. The server ensures that the ID sent to the client in the previous message is included in the new request from the client. If it is different, the server rejects the request and assumes it is subject to a replay attack. The attacker cannot spoof the message ID, because the message is signed.

The basic replay attacks can also be countered by using a relatively short validity time window. In the web services world, information about the message creation time is

usually communicated by inserting timestamps, which may just tell the instant the message was created, or have additional information, like its expiration time, or certain conditions. This solution, however, requires clock synchronization and also sensitive to issues such as message queuing at busy and non-responsive servers.

2.2.7 Denial of Service

Denial of service is the process of making a system or application unavailable. A denial of service attack can be accomplished by bombarding a server with requests to consume all available system resources or by passing it malformed input data that can crash an application process. Furthermore, this attack can be attained by many methods aimed at several targets within the infrastructure. At the host, an attacker can disrupt service by brute force against the application, or an attacker may know of a vulnerability that exists in the service the application is hosted in or in the operating system that runs your server.

The SYN flood attack is a common example of a network level denial of service attack. It is easy to launch and difficult to track. The aim of the attack is to send more requests to a server than it can handle. The attack exploits a potential vulnerability in the TCP/IP connection establishment mechanism and floods the server's pending connection queue.

Though web services have not much to do with this kind of attacks, it should be addressed in mobile web services domain because of two reasons. First, as the access point for a mobile device would only be the base station, the channel will be sensitive for Distributed Denial of service attacks [DDOS] because of limited resources in such medium. Second, our domain includes a web server in the Mobile Host. The Countermeasures to help prevent denial of service are as follows [41]:

- Configure applications, services, and operating system with denial of service in mind.
- Harden the TCP/IP stack by applying the appropriate registry settings to increase the size of the TCP connection queue, decrease the connection establishment period, and employ dynamic backlog mechanisms to ensure that the connection queue is never exhausted.
- Make sure the account lockout policies cannot be exploited to lock out well known service accounts.
- Make sure that applications are capable of handling high volumes of traffic and the thresholds are in place to handle abnormally high loads.
- Review the application's failover functionality.
- Use a network Intrusion Detection System (IDS) that can detect potential denial of service attacks.

2.2.8 Repudiation

Repudiation is the ability of users to deny that they performed specific actions or transactions [60]. Without adequate auditing, repudiation attacks are difficult to prove.

Normally, this can be achieved by saving server logs, called audit trails, in a secure location.

The auditing ensures non-repudiation which means that a message can be verifiably traced back to the caller. The standard web service practice is to require cryptographic digital signatures over any content that has to be legally binding and the documents with such signature are saved in the audit log. With this process, the information saved in audit log can be reliably traced to the owner of the signing key. In our Mobile Host scenario, as it can get cumbersome to store the audit logs due to its scarce resources, a trusted-third party server can solve the purpose by maintaining the audit logs.

2.2.9 Other Notable Issues

In advent of flexible interoperability in web services between different parties and domains, the need for tightly controlled authentication and access rights mechanism is evident. But, it is extremely difficult to have a single authentication and access control scheme for all the involved parties exchanging multiple formats. Single Sign-on might help to some extent. It's possible to map credentials across diverse systems with Single Sign-on so that each web service can deal with their own credential system.

Another critical factor for mobile web services would be threat containment. Consistent troubleshooting information is not feasible in mobile wireless networks with decentralized administration. User mobility and free access to communication channels make mobile wireless networks more vulnerable. Traditional solution to shut down systems won't be possible in this medium. Log format standardization and developing certain policies for threat containment should help for some extent. Cross network coordination among administrators should be helpful for better responses in this issue.

On the whole, as mobile networks and devices have relatively less resources when compared to traditional networks, the efficiency and quality of secure service provisioning will become critical factors. The considerable factor would be resource consumption of handheld devices which will increase linearly with security deployments. But with the emerging new networks such as UMTS [22] technologies and various optimization techniques for mobile web Services such as WSOAP [18], SOAP optimization techniques like WBXML [9], Jzlib [12], Differential Encoding [8], MTOM/XOP [[19],[20]], Fast Web Services [21], HHFR [31], KSOAP [4] etc. should make the potential security deployments commercially viable in the wireless networks.

2.3 Existing/Emerging Standards for Web Services and Wireless Environments

In the previous section, security challenges for the mobile web services domain have been addressed. Before considering the realization of those issues, this section discusses the current security standards, specifications and some relevant notable projects in web services and wireless domains. This section starts with detailed explanation of complete web services security hierarchy and ends with brief descriptions of Liberty Alliance project and Open Mobile Alliance project.

To start with this section, listed below is some of the Standard committees and organizations working with web services and wireless domain security:

- W3C [39] is an industry group whose contribution in this are XML Schema, SOAP, XML Encryption, XML Description and WSDL standards.
- OASIS [5] is an organization which has larger interest in web service specific standards and it owns primary areas of our interest such as WS-Security and SAML standards.
- Web Service Interoperability group [38] was formed to promote general framework for interoperable web services. Mostly its work consists of taking other broadly accepted standards, and develop so-called profiles, or set of requirements for conforming web service implementations. In particular, its Basic Security Profile (BSP) relies on the OASIS' WS-Security standard and specifies sets of optional and required security features in web services that claim interoperability.
- Liberty Alliance [17] consortium was formed to develop and promote an interoperable Identity Federation framework. Although this framework is not strictly web service-specific, but rather general, it is important for this topic because of its close relation with the SAML standard developed by OASIS.
- Open Mobile Alliance (OMA [36]) was formed to develop and promote interoperability for mobile data services. The under-development specifications from this organization have support of all the major mobile related companies, vendors, operators etc. The OMA Web Service Enabler specification is the relevant document in this master's thesis context.

2.3.1 Web Services Security Introduction

The idea of secure web services led to the development of various security specifications among which WS-Security specification serves as the base for all other specifications. The rest of the specifications constitute WS-Policy which defines the rules for service interaction, WS-Trust which defines trust model for secure exchanges, WS-Privacy which states the maintenance of privacy of information, WS-Secure Conversation that specifies how to establish and maintain secured session for exchanging data, WS-Federation which defines rules of distributed identity and its maintenance, and WS-Authorization specification which processes the access rights and exchangeable information.

In web services, Traditional SOAP protocol which defines the communication framework does not exactly provide protection mechanisms for exchanging messages in a secure way. The following sub-sections describe ways to understand, embed and implement protection for message security in a step-by-step manner [42]:

2.3.1.1 XML Security Specifications

XML Signature (XML-dsig [26]), and XML Encryption (XML-enc [25]) add cryptographic protection to plain XML documents. XML-dsig specification provides data integrity and authentication features which can be wrapped within the XML format while

XML Encryption specification addresses the data confidentiality issues by supporting encryption/decryption of whole XML documents or only of some elements inside them.

The highly flexible framework behind these standards provides references to the data being processed with the help of secret keys and key pairs. The results from signing and encrypting operations will be in form of XML which makes them relatively easy to embed into the original document. But XML-Dsig and XML-enc themselves do not the solve the problem of securing SOAP-based web service interactions because both the end parties of a message first have to agree on the order of operations, cryptographic token retrieval methods, signature look-ups, message validation and so on. The following sections will address those higher level specifications.

2.3.1.2 SAML Security Specification

A broad set of security-related specifications are currently under development for various aspects of web service operations. One of them is SAML [11], which defines how identity, attribute, and authorization assertions should be exchanged among participating services in a secure and interoperable way. Some of the SAML specification standard features are briefly described below:

Security Assertion Markup Language SAML [11] from OASIS primarily provides Single Sign-on (SSO), Cross Domain interoperability, means of implementing the basic WS security standard through assertions, and helps in managing identity control across domains and organizations - for enhanced user experience. SAML builds on top of the WS security specifications which are discussed below and provides a means by which security assertions can be exchanged between different service entity endpoints.

The basic components of interest are Assertions, protocols, bindings and profiles. SAML Assertions carry the authentication information while SAML Request/Response protocols tell how and what assertions can be requested. Bindings define the transportation of SAML protocols over SOAP/HTTP protocol. A SAML profile can be created using the bindings, protocols along with the assertion structure. The SAML Requestor or SAML Response will reside in SOAP Body.

SAML Request/Response protocol binding over SOAP will provide Assertions in the SOAP Body with information about authentication and authorization. Then SAML Assertions are used along with the WS security element which will reside in SOAP Header. As the SAML Assertions contain key of the holder, it can be used to digitally sign the SOAP Body. At the Receiver end, the signature is verified with the help of the key and the access controls within the Assertion.

XACML [11] – Extensible Access Control Mark-Up Language defines syntax and semantics of a language to express and evaluate access control policies. SAML can also be used independently with other access control mechanisms. When both SAML and XACML are used together, they result in two additional components: Policy Enforcement Point (PEP) and Policy Decision Point (PDP). When PEP receives requests from Requestor, it accesses assertions from the Requestor and extracts other typical information such as time of request, location etc. and sends it to PDP. PDP then evaluates the request by obtaining related policies and passes on the decision to PEP which enforces the decision towards the Requestor.

The following sub-section constitutes the core WS-Security specification standard which is considered as the foundation for all other specifications in this domain, creating a basic infrastructure for developing message-based security exchange.

2.3.2 WS-Security (WSS) Standard Specification

The WSS standard main aim is to address the specifications of most of the core security areas and to leave the higher-end details to profiles which will manage them. The standard designed core areas are [59]:

- Ways to add security headers (WSSE Header) to SOAP Envelopes
- Attachment of security tokens and credentials to the message
- Inserting a timestamp
- Signing the message
- Encrypting the message
- Extensibility

Signing and encrypting message mechanisms are not expected to change significantly where as types of tokens and ways of attaching those to the message can change considerably. The WSS standard defines, at high level specification, three basic types of security tokens which can be attached to WS-Security header. They are Username/password, Binary, and XML tokens and are explained in detail in the forthcoming subsections.

One of the main advantages of WS-Security standard is its extensibility. This feature helps in adding new security token and protocol types developed by defining additional profiles. These profiles will then make way to add the new tokens into the WS-Security framework.

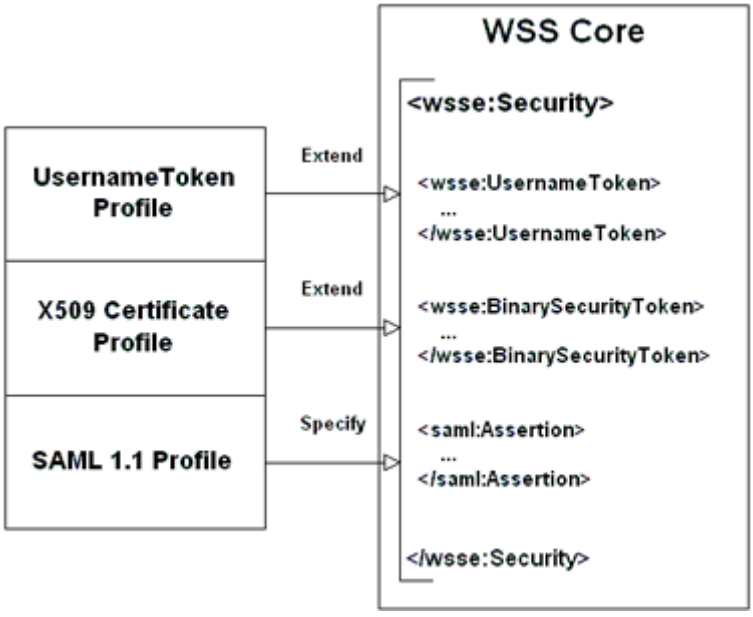


Figure 6 : WS-Security Specification Hierarchy [59]

The WS-Security standard main goal is to provide the message-level security to each message or even individual parts of a message, which passes through network, by ensuring authenticity, confidentiality, integrity and freshness. This is not possible traditional networks where the entire stream must be presented with the security context as explained in the security challenges for web services and wireless environments subsection.

WSS standard is not by itself a way to develop applications to ensure security. It provides the ways to represent a message with security and leaves the rest to so-called higher-end protocols. These protocols achieve the security goals by pertaining to the standards like Liberty Alliance, WS-Policy etc. and use the WSS information to implement authorization control, message confidentiality, and integrity and so on.

2.3.2.1 WS-Security Building Blocks

The WS-Security standard provides a core document and many other profile documents. The core document describes the way to add security headers in to the SOAP envelope and emphasizes the security context information which security header should contain to regard it as valid. While profile documents take care of defining the higher-end context which is not represented in core specification and providing extensibility for new security token types. Core WSS 1.0 specification [31] defines various security tokens and their referencing mechanisms, way to represent timestamps, and approach to include XML encryption and signature information in to the security headers.

Associated specifications are [59]:

- Username profile 1.0 [32] which adds various password-related extensions to the basic UsernameToken from the core specification.
- X.509 certificate token profile [33] which specifies, how X.509 certificates may be passed in the BinarySecurityToken, specified by the core document.
- SAML Token profile [34] specifies how XML-based SAML tokens can be inserted into WSS headers.

The WS-Security specification basically works with two different types of data. One is the security information which consists of signature, digest, timestamps, security tokens etc. The other is message data which includes all the rest information from the SOAP envelope excluding the security information. As WS-Security information, XML-based standard, is represented in XML format, all the binary data produced from the likes of signatures and security tokens needs to be transformed using Base 64 encoding/decoding. To handle encoding parts, an encoding algorithm's identifier, defined in WS-Security specification documents, is carried with the data so that the decoding side will know how to apply the relevant decoder to interpret it.

2.3.2.2 Security Header Structure

The security header resides in the optional SOAP header within a SOAP Envelope. It potentially acts as a seal to a letter. The Figure 7 shows the typical structure of a SOAP message envelope with WS-Security header.

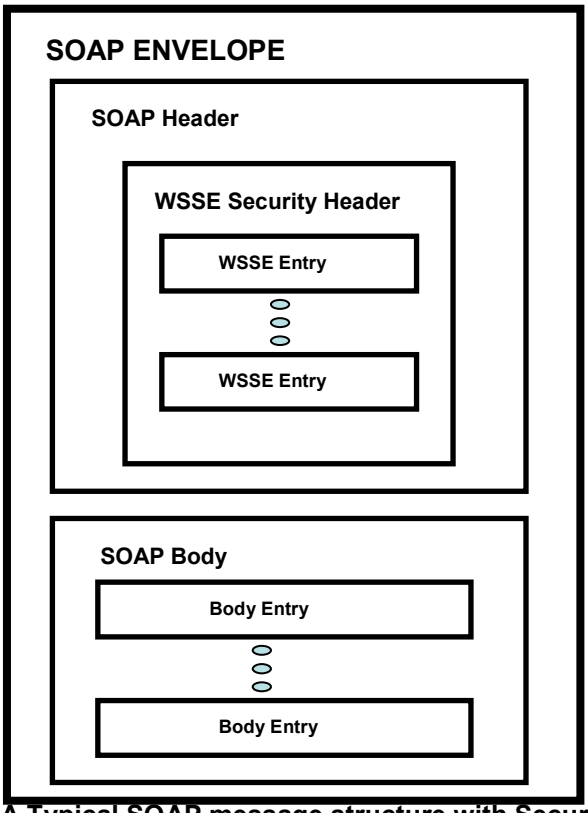


Figure 7 : A Typical SOAP message structure with Security header

Multiple security headers can be present in a SOAP Header, provided they are assigned to different actors. Though there is a scope of referencing each other within the SOAP Header, its advised to avoid such situations as it presents a complicated logistical problem in understanding the order of signature or decryption verifications. WS-Security header is an optional one, as SOAP header is itself optional, to be contained in the SOAP envelope. The following example 1 represents a minimalist SOAP envelope without a message:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header>
        <wsse:Security
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss- wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-
      open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
      soap:mustUnderstand="1">
      .....
            .....
        </wsse:Security>
    </soap:Header>
    <soap:Body>
      .....
            .....
    </soap:Body>
</soap:Envelope>
```

Example 1 : An empty SOAP message with WS-Security header [59]

A Typical SOAP message with XML Signature, XML Encryption elements and one or more security tokens with references will look like the following example 2:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <wsse:Security
      xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
      wssecurity- secext-1.0.xsd" xmlns:wsse="http://docs.oasis-
      open.org/wss/2004/01/oasis-   200401-wss-wssecurity-secext-
      1.0.xsd" xmlns:wsu="http://docs.oasis-o
      pen.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
      soap:mustUnderstand="1">
      <wsse:BinarySecurityToken
      EncodingType="http://docs.oasis-  open.org/wss/2004/01/oasis-
      200401-wss-soap-  message-security-1.0#Base64Binary"
      ValueType="http://docs.oasis- open.org/wss/2004/01/oasis-200401-
      wss-x509-token-profile-1.0#X509v3"  wsu:Id="aXhOJ5">
      MIICtzCCAi...
      </wsse:BinarySecurityToken>
      <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
<xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-
1_5"/>
      <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
        <wsse:SecurityTokenReference>
<wsse:Reference URI="#aXhOJ5" ValueType="http://docs.oasis-
      open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-
1.0#X509v3"/>
        </wsse:SecurityTokenReference>
      </dsig:KeyInfo>
      <xenc:CipherData>
          <xenc:CipherValue>...... </xenc:CipherValue>
      </xenc:CipherData>
      <xenc:ReferenceList>
          <xenc:DataReference URI="#aDNa2iD"/>
      </xenc:ReferenceList>
      </xenc:EncryptedKey>
      <wsse:SecurityTokenReference wsu:Id="aZG0sG">
      <wsse:KeyIdentifier ValueType="http://docs.oasis-
      open.org/wss/2004/XX/oasis-   2004XX-wss-saml-token-profile-
      1.0#SAMLAssertionID"
      wsu:Id="a2tv1Uz">1106844369755</wsse:KeyIdentifier>
      </wsse:SecurityTokenReference>
<saml:Assertion AssertionID="1106844369755" IssueInstant="2005-01-
      27T16:46:09.755Z" Issuer="www.my.com" MajorVersion="1"
      MinorVersion="1"
xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion">
            ...
      </saml:Assertion>
      <wsu:Timestamp wsu:Id="afc6fbe-a7d8-fbf3-9ac4-f884f435a9c1">
      <wsu:Created>2005-01-27T16:46:10Z</wsu:Created>
      <wsu:Expires>2005-01-27T18:46:10Z</wsu:Expires>
      </wsu:Timestamp>
<dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
Id="sb738c7">
            <dsig:SignedInfo Id="obLkHzaCOrAW4kxC9az0bLA22">
            ...
         <dsig:Reference URI="#s91397860">
```

```
            ...
  <dsig:DigestValue>
      5R3GSp+OOn17lSdE0knq4GXqgYM=
  </dsig:DigestValue>
         </dsig:Reference>
        </dsig:SignedInfo>
<dsig:SignatureValue Id="a9utKU9UZk">LIkagbCr5bkXLs8l...
</dsig:SignatureValue>
            <dsig:KeyInfo>
        <wsse:SecurityTokenReference>
<wsse:Reference URI="#aXhOJ5" ValueType="http://docs.oasis-
      open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-
1.0#X509v3"/>
        </wsse:SecurityTokenReference>
        </dsig:KeyInfo>
      </dsig:Signature>
    </wsse:Security>
  </soap:Header>
  <soap:Body
      xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-  wssecurity-utility-1.0.xsd" wsu:Id="s91397860">
    <xenc:EncryptedData
      xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" Id="aDNa2iD"
      Type="http://www.w3.org/2001/04/xmlenc#Content">
<xenc:EncryptionMethod
      Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
      <xenc:CipherData>
      <xenc:CipherValue>XFM4J6C... </xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedData>
  </soap:Body>
</soap:Envelope>
```

Example 2 : A Typical SOAP message with WS-Security [59]

2.3.2.3 Types of Tokens

A WS-Security Header can contain the following types of security tokens [59]:

- Username Token

  The username token defines the way to pass the username and the optional password. If the token is not encrypted as a whole, then the password should be transferred through secured channel. If in case the complete secure transmission is not possible, then sending hashed password with nonce and a timestamp is recommended. The following password hash algorithm without ambiguity is defined in the profile document:

  ```
  Password_Digest = Base64 (SHA-1 (nonce + created + password))
  ```

- Binary token

These binary tokens are required to convert binary data into text-encoded format.

X.509 and Kerberos certificates are examples of binary data and default encoding format is Base64. BinarySecurityToken element is defined by core specification and additional attributes and sub-elements are taken care by profile documents.

```
<wsse:BinarySecurityToken   EncodingType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-soap-message-security-
1.0#Base64Binary"             ValueType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-
1.0#X509v3" wsu:Id="aXhOJ5">

     MIICtzCCAi...

</wsse:BinarySecurityToken>
```

Example 3 : A Typical Binary Security Token [Example 2]

- XML token

XML tokens are the carriers of SAML assertions. The WS-Security core specification points out the possibility of these tokens in the security header. All the remaining details should be handled by the profile documents.

```
<saml:Assertion
   AssertionID="1106844369755"           IssueInstant="2005-01-
   27T16:46:09.755Z"    Issuer="www.my.com"    MajorVersion="1"
   MinorVersion="1"
   xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion">
      ...
</saml:Assertion>
```

Example 4 : A Typical SAML Assertion structure [Example 2]

2.3.2.4 Referencing

Referencing is one of the essential parts of WS-Security specification. It helps in recognizing the encrypted and signed message parts and retrieving security tokens transferred in the message. A special attribute, wsu:Id, has been defined in the core specification for this purpose. The attribute's only mandatory clause is that it should be unique within the scope of XML message where it is defined. The advantage of this referencing in an application is that the intermediate processors are not required to understand the XML Schema of the message. Note that in case of encryption and signature element identification, local referencing should be considered first because the XML encryption and signature specifications do not allow attribute extensibility.

WS-Security core specification also defines a general mechanism for referencing security tokens via SecurityTokenReference element. An example of this kind of element representing a SAML assertion is shown here:

```
<wsse:SecurityTokenReference wsu:Id="aZG0sGbRpXLySzgM1X6aSjg22">
  <wsse:KeyIdentifier
```

```
    ValueType="http://docs.oasis-open.org/wss/2004/XX/oasis-
    2004XX-wss-saml-token-profile-1.0#SAMLAssertionID"
    wsu:Id="a2tv1Uz">
       1106844369755
  </wsse:KeyIdentifier>
</wsse:SecurityTokenReference>
```

Example 5 : A Typical Security Token Reference Element [Example 2]

This SecurityTokenReference element supports almost all the token types including certificates, SAML assertions and encryption keys. This feature makes it complicated due to its high extensibility. So, the WSS specification suggests only two reference types, Direct References and Key Identifiers, out of the possible reference types. Further, profile documents can add extensibility to this mechanism to have custom token types.

2.3.3 WSS Message Protection Methods

As we discussed previously, traditional transport level security does not serve well in case of web services. WSS helps addressing some of them at the SOAP message level, using the mechanisms described in the sections below.

2.3.3.1 Integrity

WS-Security specification ensures message integrity using XML digital signature standard. Before signing a XML element or document, the XML content should undergo transformation to result in to canonical representation. XML digital signature standard provides two main transformations, Inclusive and Exclusive Canonicalization Transforms. These two transformations differ in their way of declaring namespaces. Exclusive Canonicalization Transformation is recommended by the WS-Security core specification as it liberates the duplication of signed XML documents or elements into other elements with valid signature.

Security Token Reference de-reference transform option is provided by WS-security to address the digitally signed tokens in a uniform way. Two types of referencing are possible with Security Token Reference mechanism as mentioned in WSS core specification. One is addressing the referenced signing keys defined by XML digital signature standard. The other is to allow referencing to signed security token. The latter one is possible by creating custom token types extending the profile documents.

Example 2 explained in the previous subsection depicts a typical signature. On a whole, XML digital signature standard allows the sign secure elements passed in the SOAP request like SOAP body, user credentials, and the timestamp. Note that a problem can arise when a definite element is both encrypted and signed as these operations can be done in any particular order. This problem multiples when the SOAP envelope contains multiple security headers where encryption and signature content might overlap. To address this issue to a certain extent, the WS-Security core specification mandates that each element should pre-pended when adding it into the security header.

2.3.3.2 Confidentiality

WS-Security specification ensures message confidentiality using XML encryption standard. This encryption standard works very much similar to the XML digital signature standard except that it replaces the complete element(s) that are encrypted and placing a sub-element containing the encrypted bytes. For proper decryption at the other end, the WSS core specification recommends to use a unique value (nonce/hash) for referencing. This unique value should be pre-pended, with possible encryption to ensure protection, in the security header. Example 2 represents a typical SOAP envelope with WS-Security which also reflects the encrypted body. The specification also supports both symmetric and asymmetric encryptions to varied encryption requirements.

2.3.3.3 Freshness

Timestamp mechanism is used to address the freshness of the SOAP messages. A timestamp in a SOAP message helps in preventing reply and tampering attacks. A single timestamp per security header, based on UTC time, is enough to guarantee the message freshness. Below explained XML content summarizes further about a timestamp. Note that a single timestamp pertained to a security header is valid to the entire SOAP message as the message can have more than one security headers. Clock synchronizations are not addressed in WSS core specification and should be dealt in external domains.

```
<wsu:Timestamp wsu:Id="afc6fbe-a7d8-fbf3-9ac4-f884f435a9c1">
<wsu:Created>2005-01-27T16:46:10Z</wsu:Created>
<wsu:Expires>2005-01-27T18:46:10Z</wsu:Expires>
</wsu:Timestamp>
```

Example 6 : A Typical Timestamp Element [Example 2]

2.3.4 WSS Access Control Methods

Secured access control methods are not directly offered by WS-Security specification. Instead the specification provides the way to the access control data and tokens between sender and receiver SOAP endpoints in a secure manner.

2.3.4.1 Introduction

Identification ensures a claimer's identity by attaching its relevant information in the SOAP header. The information can be anything which can represent the identity claimed by using a token, a Kerberos ticket, a username or a SAML assertion. The WS-Security core specification extensibility mechanism liberated this usage of different security token types to a message for different purposes.

2.3.4.2 Authentication

Authentication can be achieved in two ways. One is credentials verification which can be accomplished quite straightforward by pre-pending the concerned to the security header. Two is token validation where the token contains the user's identity with its integrity proof and these tokens are created prior to the authentication process.

WS-Security specification supports most of the standard authentication protocols by binding sender information with the appropriate protocol relevant tokens. However, the verification of the authentication information and requirements of a specific authentic process are left to the web service who wants to claim the identity or authenticity of the caller. As there no specific standards or requirements mentioned by WSS specification in this regard, one can choose anything from username/password checks or signature verification scheme using PKI.

2.3.4.3 Authorization

Authorization can be achieved using XACML explained in the SAML specification subsection. The authorization rules are not web service-specific and are not specified by WSS core specification. This mechanism is generally left to the web service deployment domain. Depending on the scope of security required, domain size etc., there can be several layers of authorization process at the service provider.

2.3.4.4 Policy Agreement

WS-Policy specification helps in addressing complex policy requirements and the concerned security parameters to determine to endpoint's security requirements. This is needed because the general web services' communication depends on endpoint's public interface defined in WSDL file which does not define any higher-end security parameters. With this policy specification, the endpoints can publish their security requirements and clients can fetch those and address them accordingly in their web service request.

Figure 8 further explains the policy mechanism with the help of WS-Trust enabled services. The requestor checks the endpoints' policy and interprets it to construct the request. If some security tokens are not available at requestor to address the policy, he will fetch them from WS-Trust component.

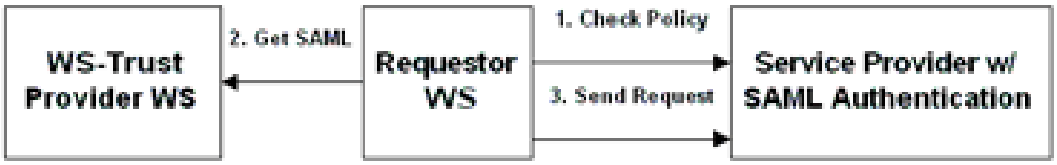


Figure 8 : Trust Service [59]

Before reading further sub-sections, the contents of this section can be summarized as follows. W3C and IETF's XML Signature specification provides data integrity and authentication features which can be wrapped within the XML format. XML Encryption specification by W3C addresses the data confidentiality issues using encryption techniques. Joining these two specifications, OASIS's WS-security defines a way to include integrity, confidentiality and single message authentication features within a SOAP message.

The WS Security standard basically adds security information to the SOAP message headers. The optional SOAP headers, security tokens, can carry Authentication

information, Transactional & Payment details and etc. Security token is also called a term which represents a collection of claims could be implemented in various forms such as X.509 certificates, IETF Kerberos tickets, or username and password combinations. An example security token is a signed certificate that binds a sender's identity with a public key. Confidentiality can be achieved with the help of WS security tokens combined with XML Encryption. Similarly, XML Signatures combined with WS security tokens serves towards the integrity.

2.3.5 Liberty Alliance

Liberty Alliance [17] Project is the only global body which is working to define and provide technology, knowledge and certifications to build identity into the foundation of mobile and web serviced communication. Its main concentration was on Federated Identity because of the lack of connectivity between identities for internet applications in the current wireless technology especially in mobile networks.

The Basic components of Liberty Alliance are Principal, Identity Provider and Service Provider. Principal is the Requestor who needs to be authenticated. Identity Provider is the one which authenticates and asserts the Principal's identity. The basic provisions this project are Federation which establishes relationship between any two of the above mentioned components, Single Sign-on (SSO) where the authentication provided to Principal by the Identity Provider can be maintained to other components such as Service Providers, and Circle of Trust where Trust will be established between Service Providers and Identity Providers with agreements upon which Principals can make transactions and exchange information in a seamless and secure way.

2.3.6 Open Mobile Alliance

Open Mobile Alliance (OMA [36]) is an organization whose primary motive is to drive interoperability of mobile data services. OMA is a collaboration of almost all of the mobile specific companies, customers, operators etc. The primary benefit of this approach would be seamless mobile services for end user worldwide, thereby, achieving end-to-end interoperability of mobile services. It is imperative that once interoperability standards are available, mobile services can be developed which then can connect users with all types of services providers and enterprises. The following Figure 9 gives a view OMA approach towards developing open/global standards and specifications for mobile data services.

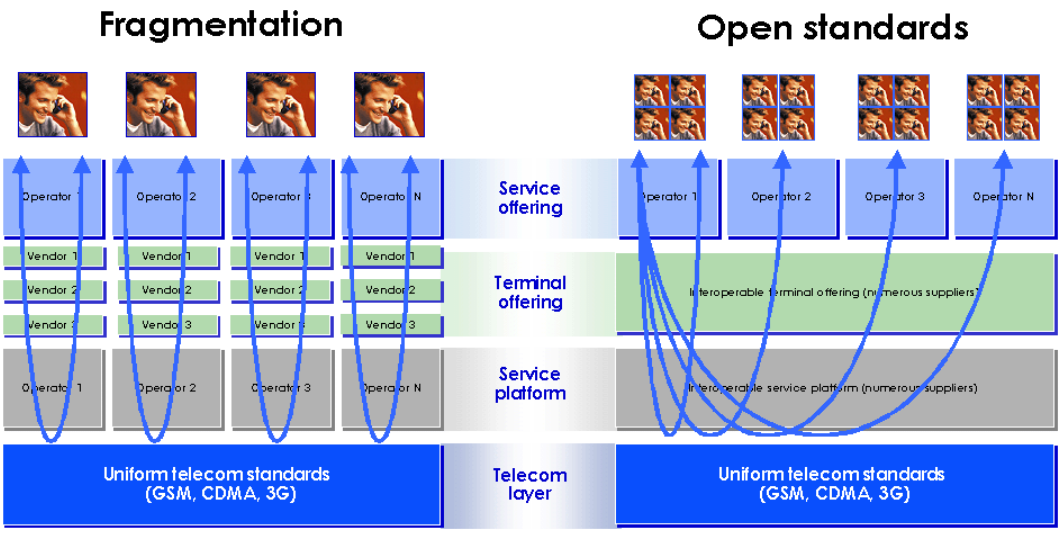


Figure 9 : OMA Motive Overview [36]

Though the specifications are still in development phase and not released, it is worth mentioning, in brief, the current thesis related core web service specification from OMA, the "OMA Web Services Enabler "[35]. Mobility and roaming are the obvious key characteristics which are hindrances to mobile web service interactions. The current possible mobile web service applications have a number of drawbacks as following. First, the applications should be created through tightly-coupled, costly and close alliances between value-added service providers. Second, they have to be created based on a mixture of mostly propriety models and disparate/overlapping standards such as WAP, Location, Presence, Identity etc. Furthermore, most of the standards to develop those applications have been devised specifically for the mobile environment from the ground up. All these drawbacks will draw high complexity to deploy, integrate and use those applications/services.

The OMA Web Services Enabler specification is destined to cover all the drawbacks mentioned above and envisioned to support the following mobile web service interactions [36]:

- Server-to-server
- Server-to-mobile terminal
- Mobile terminal-to-server
- Mobile terminal-to-mobile terminal (peer-to-peer)

2.4 Existing Mobile Technology Standards

Today's second-generation GSM networks deliver high quality and secure mobile voice and data services like SMS, circuit switched Internet access etc., with full roaming capabilities and across the world. The GSM platform is a widely successful wireless technology and it is the world's leading mobile standard. But, with the advent of the 2.5-

generation (Interim Generation) technologies like GPRS and EDGE, and 3G technologies like UMTS, higher transmission rates are achieved in the wireless domain.

The General Packet Radio Service (GPRS) is a non-voice value added service that allows information to be sent and received across a mobile telephone network. The General Packet Radio Service is an extension to second generation GSM. It provides short connection setup times and packet switched connections. GPRS offers faster data transmission via a GSM network within a range of 9.6Kbits to 115Kbits. The available bandwidth can be shared among different users. The high bandwidth is achieved by combining up to eight time slots at the radio interface, where the data is transported in a packet-oriented way.

With the third generation technologies like UMTS (Universal Mobile Telecommunication System), which specifies 3G IP broadband mobile networks that will offer data rates between 156 Kbps and 2Mbps, better transmission rates can be achieved. Some of the UMTS smart phones are already available in commercial market.

Trust Computing Group (TCG [62]) is focusing on mobile hardware security with increasing data applications on mobile phones. This group which is a coalition of many big mobile network companies is planning to realize it as soon as year 2008. Adding hardware-based security to cell phones can enable services such as electronic ticketing and mobile payments, according to the TCG. It can also provide for secure storage of personal information such as an address book, text messages, e-mail and pictures. And, in the future, payment data such as credit card numbers will be also a possibility according to TCG.

The RIM (Research in Motion) organization is incorporating Intel's processor chip called HERMONE into their Blackberry devices which gives 1.0 GHz processing speed. This will help the smart phone computing power which compensates limited resources to an extent. Initial realization would be on EDGE (Enhanced Data for GSM Evolution) i.e., enhancement of 2G and 2.5G wireless networks. The HERMONE processor can also be tapped into UMTS (Universal mobile Telephony System) wireless networks.

Based on above security awareness study, I can conclude that securing web service provisioning in mobile networks is a great challenge. The mechanisms developed for traditional networks are not always appropriate for the mobile environment. Thus, this area still holds ample room for further research.

Summary:

This section, in summary, covered the complete state-of-the-art of the thesis. The section first explained the mobile web services and our working domain in it which includes our Mobile Host. Then some of the security challenges in both web services and wireless environment domains are discussed. Further, the section addressed the existing/emerging standards in web service and wireless domain including the likes of core WSS specification, LA and OMA specifications etc. Finally, the section ended by discussing briefly the existing mobile technology standards.

3 Mobile WS-Security Design

This design section will first zero-in on the basic security requirements for mobile web services to be realized in this thesis domain. Then the section will explain the design models to realize both the message-level security and end-point security for mobile web services domain concentrating on our Mobile Host.

3.1 Security requirements for Mobile Web Services

This subsection will address the basic mobile web service centric security requirements. Though it's quite complex and not commercial to configure and use the existing disparate and overlapping standards for developing secure mobile web service applications, my goal is to round up the best possible standards and ways to achieve the basic quality of service.

Once the web service is deployed on the Mobile Host, the service is prone to different types of security breaches mentioned in Section 2. For avoiding these sorts of attacks, the web service communication should support the following basic security requirements- Confidentiality and Data Integrity in addition to Authentication and Access Control. Secure message transmission is achieved by ensuring Confidentiality and Data Integrity, while authentication and authorization will ensure the service is accessed only by the trusted service requestors. The overview of the basic security requirements of mobile web service provisioning is shown in Figure 10.

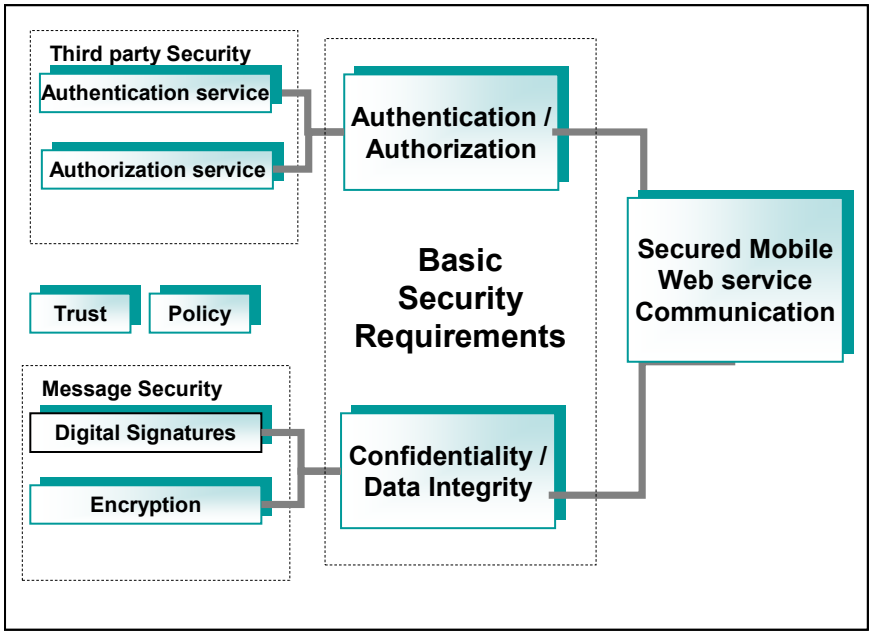


Figure 10 : Mobile Web Services Security Overview

The means of secure and reliable information exchange in wireless networks is a fundamental need yet not feasible with the current standards, in the mobile web services

domain. The flexibility of web services communication between different involved parties such as providers, suppliers, customers and etc., requires proper Authentication and Authorization principles, which are considered as two most critical issues in mobile networks. Authentication is the proof of identity of the mobile nodes in dynamic mobile networks. The network identity of the nodes should be maintained a bit differently as opposed to traditional wired networks. As of now, it is only possible with the help of mobile network operators; the current state-of-the-art suggests that the federated identities should be maintained at the concerned network operators. Authorization is the proof of access controls of the mobile nodes. It is also a critical issue in mobile web service domain as web services are programmatic interfaces with which monitoring suspicious activities might become difficult. So, there is a need to authorize not only the users and applications but also the individual operations of an application.

As discussed earlier, in the mobile web service provisioning domain, the web service Requests/Responses are transmitted as SOAP messages over HTTP. So the secured HTTPS, with SSL encryption [16] over HTTP can be used for message transmission. But HTTPS would not be enough to provide confidentiality as it offers only session based, point-to-point data privacy over transport layer and the SOAP messages need to pass through many intermediate nodes, across different transport protocols. Additionally mobile web services are loosely coupled and dynamic; the messages can be decrypted and encrypted at various intermediate hops with secure transports. So with the transport layer security mechanisms, the confidentiality and Data Integrity are not feasible as they can be read and changeable at intermediaries.

XML Encryption [25] of the SOAP messages provides a way to encrypt the sensitive information and manage the encryption till it reaches its ultimate destination. This will allow end-to-end data privacy. Also we have to make sure that the messages are not modified before it reaching final destination. Digital signatures can help in signing messages to provide end-to-end data integrity among diverse systems. Thus, the above mentioned basic security requirements can be achieved with the WS-Security specification and relevant APIs such as WSS4J [37] in standalone web services. But, it is extremely difficult to adapt the same security specification on to the mobile web services domain, in view of ad-hoc networks sensitivity and constrained resources of the handheld devices.

As discussed earlier, secure provisioning of mobile web services needs proper message-level security consisting data integrity, confidentiality and end-point access security that constitutes authentication, access control. Since, there exists no approved specific mobile web service standards and lot of propriety interfaces are involved, the security was analyzed on a case-by-case scenario.

3.2 Message-level Security Design Models

To start with, the security features confidentiality and data integrity will be analyzed on Mobile Web Service Client, because not many security implementations are available even for standalone Web Service applications, let alone Mobile WS clients. The following Figure 11 shows the basic setup for realizing confidentiality and Data Integrity on the Mobile WS client.

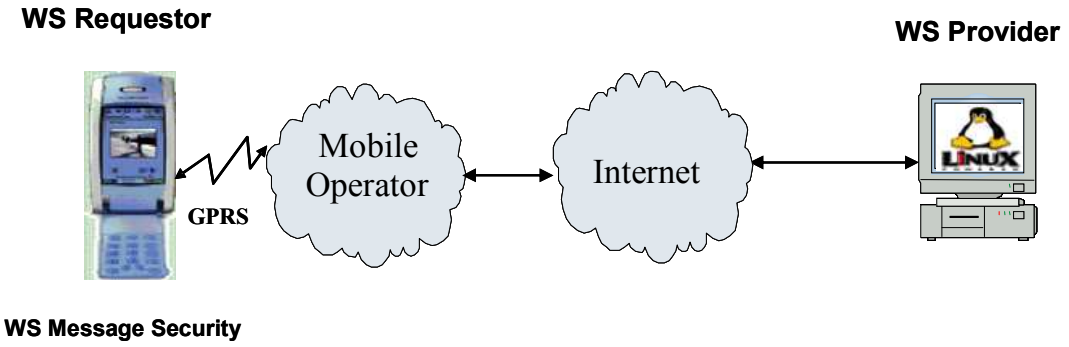


Figure 11 : Proposed Message-level Security Scenario of Mobile WS Client

As mentioned in the state of the art, to achieve confidentiality, the message should be ciphered by symmetric encryption and the symmetric key generated should be exchanged using Public Key Infrastructure (PKI [66]). Integrity is achieved by using digital signatures. Both PKI, public key encryption to exchange symmetric key and digital signature mechanisms are described below.

3.2.1 Public Key Infrastructure

Public Key Infrastructure is a mechanism built to manage digital certificates and their associated keys and uses these managed digital certificates for authentication purposes. The digital certificates are themselves can identify as users who claim to be owner of a specific public key. Public key encryption, also called asymmetric encryption, works with the help of two keys named public key and private key. Public key can be visible to all the parties who want to involve in encrypted communication where as private key should be kept secret pertained to the owner. If public key is used to encrypt the message, then the message decryption is only possible using private key which provides confidentiality. If private key is used to encrypt the message, then the message decryption is only possible using this public key which helps to verify the authentication of the sender. However, the public key authenticity needs to be checked before considering the asymmetric encryption.

The most common way to authenticate the public keys is using digital certificates. This is generally being done at a trusted third party named certification authority. Whenever a user wants to involve in public key encryption, he will generate a key pair and verifies with the certification authority according to its specific process by sending his public key. Once verified, the certification authority generates a digital certificate by binding the information about the user, user's public key, an expiration date, and the digital signature of the certification authority (signed with its private key). The certification authority then places this digital certificate at a public repository which can be a database, viewed as a commonplace to fetch these certificates. Though these digital certificates can also made available at user itself, due to our resource constrained mobile device scenario, the former way of approach looks optimistic.

Thus, we in our public key encryption implementation made assumptions that the public key is readily available at the required party to make things simpler. This is due to

our platform and device limitations. The following sub-section will explain the work flow of both public key encryption and digital signature.

3.2.2 Public Key Encryption and Digital Signature Workflow

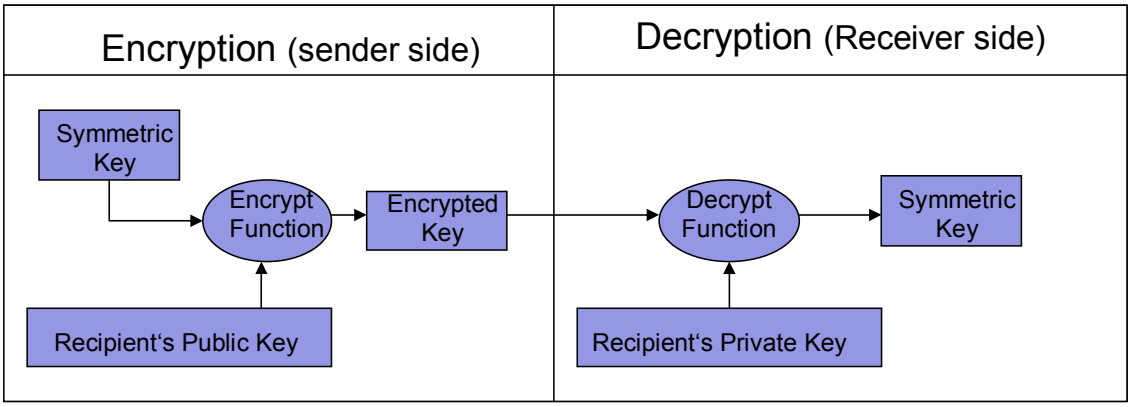


Figure 12 : Public Key Encryption workflow (achieves Confidentiality)

The Figure 12 represents the process of achieving confidentiality using PKI. The mechanism is quite straight-forward and simple. In this workflow, the sender encrypts the symmetric key with receiver's public key using encrypt function (cryptographic encrypt engine) and sends the encrypted key to the receiver. The receiver then decrypts the encrypted key with his private key using the same encrypt function.

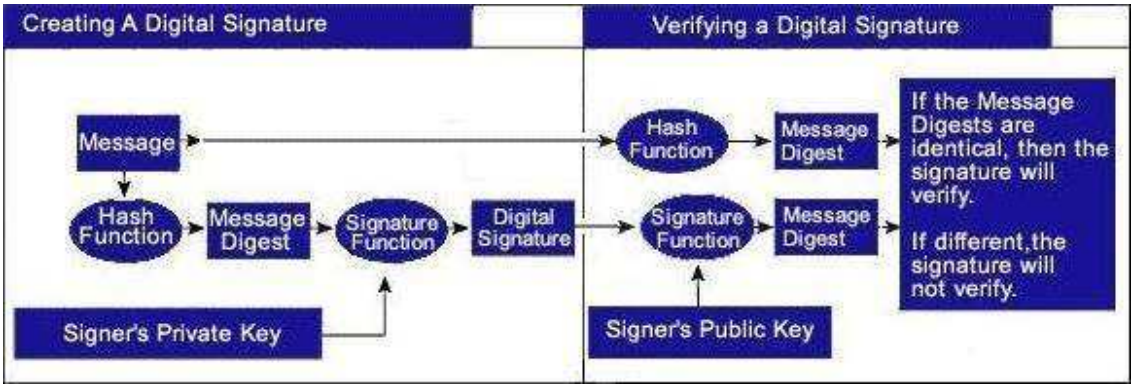


Figure 13 : Digital Signature workflow (achieves Authentication and Integrity) [61]

Figure 13 explains the workflow of a digital signature lifecycle with which user authentication and message integrity are achievable. On the sender side, message digest is calculated from the message by using hash function and then this digest is digitally signed by signature function (cryptographic signers) using the sender's private key. Both the message and the signature are sent to the receiver. On the receiver front, from the digital signature, the message digest is retrieved by the signature function using sender's

public key. This message digest when matches with the original sent message's calculated digest will be treated as authentic and not tampered with.

Based from the feedback and implementation from the message-level security realization at mobile client, the security features confidentiality and data integrity was then realized on the Mobile Host. One of the prime reasons for this modeling is that at the time of implementation, the Mobile Host was available in Personal Java where as the security analysis is being carried out in J2ME. By the time of realization of the message-level security on mobile client in J2ME, the Mobile Host is transformed from Personal Java platform to J2ME. Then the entire message-level security was realized on Mobile Host with small modifications. The Figure 14 shows the basic architecture of realizing the message-level security for the Mobile Host.

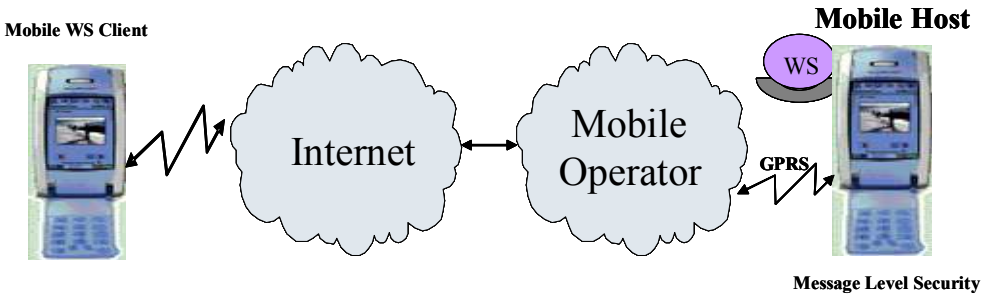


Figure 14 : Proposed Message-level Security scenario of Mobile Host

3.3 End-Point Security Design Models

Figure 15 depicts our generic architecture to realize the basic security principles for the Mobile Host including both message-level and end-point security. Once a web service is deployed on Mobile Host; any WS client can request for the service. The SOAP message along with the WS-Security information is routed across the internet to the Mobile Host. The message-level security information is extracted and addressed at the Mobile Host while the end-point access security is handled by a third party on behalf of the Mobile Host. The third party requirement depends on the possible end-point security handling at the Mobile Host. Further, the corresponding service details are extracted and the service is invoked. The SOAP response is sent back to the client across the same route.

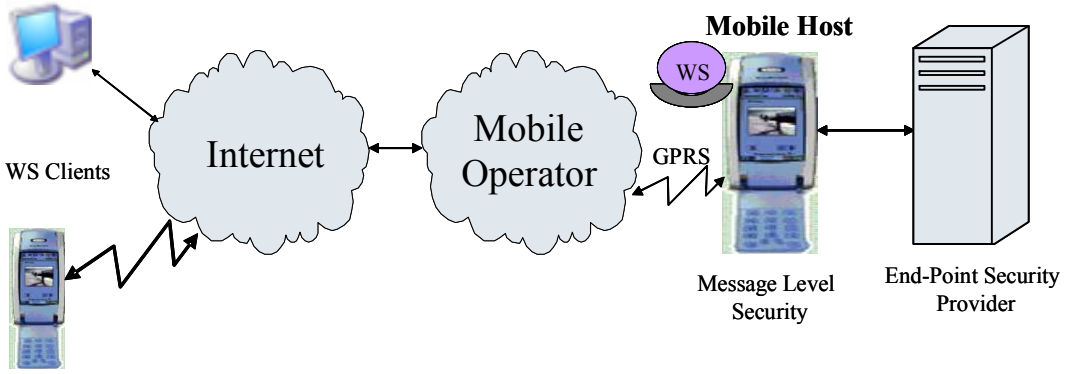


Figure 15 : Proposed WS-Security scenario of Mobile Host

As for basic authentication, it can be done using reverse Public Key Infrastructure. Here, the use who wants to authenticate will encrypt his username/message using his private key. Then, whoever wants to test the authenticity of the user will test by checking the successful decryption of the message using the user's public key. The Figure 13 described in the previous section further explains the flow of the authentication procedure.

One of the best ways to achieve the end-point security for web services is using Single Sign-on. SAML specification needs to be fulfilled while achieving Single Sign-on. SAML provides a way to embed both authentication and authorization information into the SOAP headers. LA and WS-Federation are some of the big groups which provide architectural infrastructure to use SAML. SAML implementations are in development by OpenSAML, WSS4J and SourceID projects.

SourceID group provides few toolkits to implement SAML. It is an open source project and its aim to enable identity federation and cross platform security. They are SAML1.1 Java Toolkit, ID-FF1.2 Java Toolkit (SourceID Liberty 2.0 Beta) and WS-Federation Toolkit. We considered the SourceID Liberty 2.0 Beta toolkit to realize our Sign Sign-on analysis after extensive research on all the available tools.

3.3.1 Public SourceID Liberty 2.0 Beta

SourceID Liberty 2.0 Beta [43] is a Java application developed on JBoss application server (JBoss 3.2.4). This toolkit, also referred as ID-FF1.2 Java Toolkit, allows developers build Federated Identity Management services into existing web service projects easily. Using this open source Java toolkit, we can achieve Single Sign-on and federated identity exchange by means of Liberty ID-FF v 1.2 protocols. The toolkit provides generic high-level functionality for user-friendly development by shielding the developer from the complexities of the Liberty protocol, SAML and other dependencies.

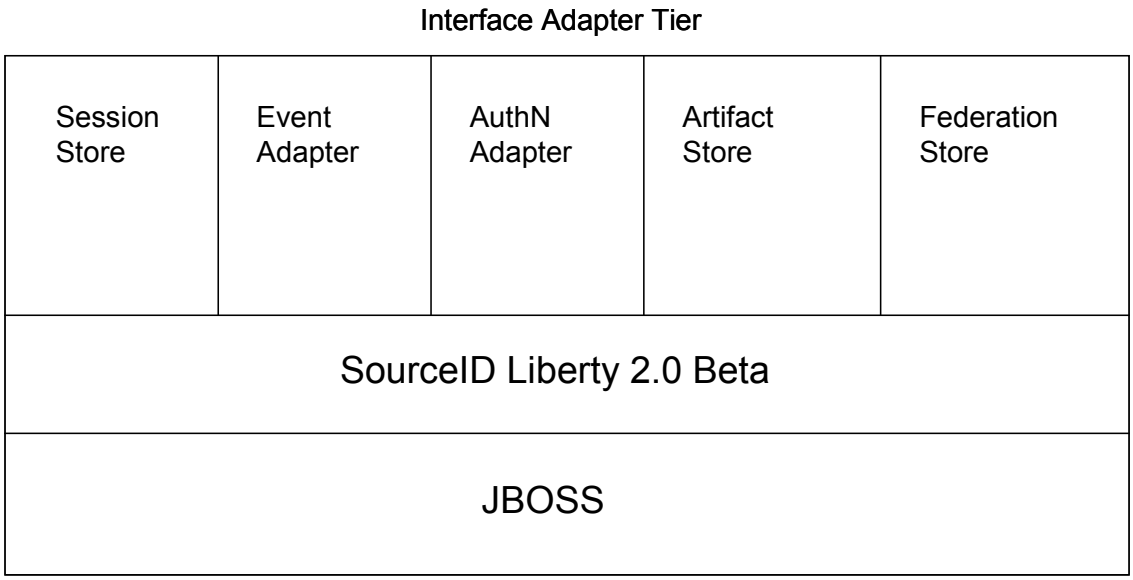


Figure 16 : SourceID Liberty 2.0 Beta High-level Architecture [43]

As shown in the Figure 16 architecture, the developer can customize the adapter tier, a set of Java interfaces, to manage data storage and interactions within and around a web application. Developers implement these interfaces to integrate SourceID into their application environments.

Out of the interfaces, in-memory implementations are available for three of them namely; Session Store, Artifact Store and Federation Store which can be readily usable by a developer without any additional configuration. Session Store maintains and provides the user information as who is logged in and logged out. Artifact Store conform the artifact profile as per Liberty specification by keeping track of an associated array of artifacts to assertions. Federation Store keeps track of information about account linkages by hiding all implementation details of mapping user account identifiers to pseudonyms. Though these above mentioned three interfaces do not need any changes to use, the developers have a choice to customize upon their need.

Event and AuthN adapters are the two interfaces which must be implemented by the developer for his specific deployment. The implementation at Event adapter must provide a notification mechanism that will be used to update the local session system when a SSO event occurs. AuthN adapter interface is used to retrieve the session identifier provided in a previous call to onSessionCreated method on the EventAdapter interface. Then this session identifier will be used by SourceID to track state information about a user's current session to make functionalities such as Single Log-out works correctly. For both Event and AuthN adapters, separate adapter instances must be created for handling Identity Provider and Service Provider side behavior.

To summarize, SourceID Liberty 2.0 Beta supports the basic core profiles such as Single Sign-on (both Artifact and POST), Single Log-out, Register Name identifier, Federation Termination Notification and Identity Provider Introduction. The current

provision from SourceID group is to utilize the framework on a standalone system which possesses the resources like Sun Java JDK 1.4.2, JBoss 3.2.4 and Ant 1.5.1 or higher.

By removing some of the high-end functionality on the Service Provider front from the toolkit, we can visualize our Mobile Host in this paradigm to achieve Single Sign-on. Basic possible JBoss compliant features specific to the toolkit needs to be considered on the Mobile Host. Below explained are the four different scenarios to achieve Single Sign-on. Upon successful implementation of the Service Provider features of the SourceID toolkit on the Mobile Host, Scenario I and Scenario II designs can be achieved. If this implementation is not feasible, then the alternate solutions would be to choose from Scenario III and Scenario IV. All the Single Sign-on scenarios are explained below one by one.

3.3.2 SSO Scenario I

In the first Single Sign-on scenario as shown in Figure 17, the workflow is quite straight-forward. The Mobile Web Service Client logs in with its generic interface. The interface back-end will request SAML token from the Identity Provider. After receiving the SAML token, the client then requests for the service access from the Mobile Web Service Provider (Mobile Host) by providing the SAML token. The Mobile Host will then requests the client token validation with the Identity Provider for its authenticity and the corresponding authorities for the client if any. Upon positive confirmation from the Identity Provider, the Mobile Host will then grants the service access to the Mobile Web Service Client.

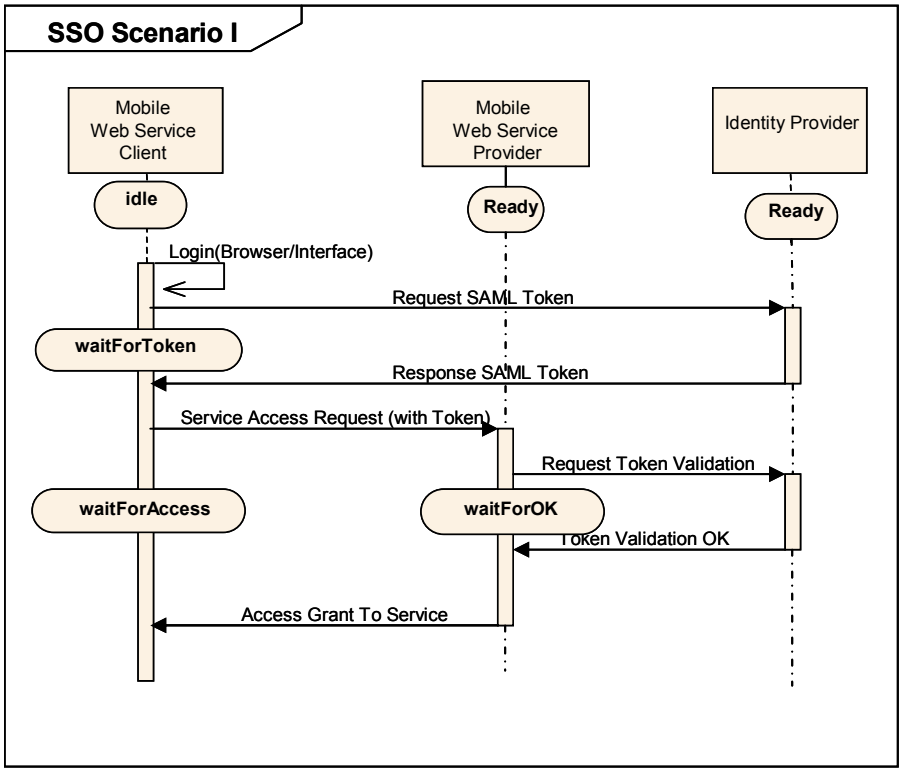


Figure 17 : Single Sign-on Design Scenario I

3.3.3 SSO Scenario II

Figure 18 depicts second Single Sign-on scenario. This workflow is required when the client does not have information about the Identity Provider. The Mobile Web Service Client logs in with its generic interface. The interface back-end will then request the service access with the Mobile Host. As the Mobile Host fails to find a SAML token, it sends the response redirecting to Identity Provider. Then the interface back-end will request SAML token from the Identity Provider. After receiving the SAML token, the client then requests for the service access from the Mobile Web Service Provider (Mobile Host) by providing the SAML token. The Mobile Host will then requests the client token validation with the Identity Provider for its authenticity and the corresponding authorities for the client if any. Upon positive confirmation from the Identity Provider, the Mobile Host will then grants the service access to the Mobile Web Service Client.

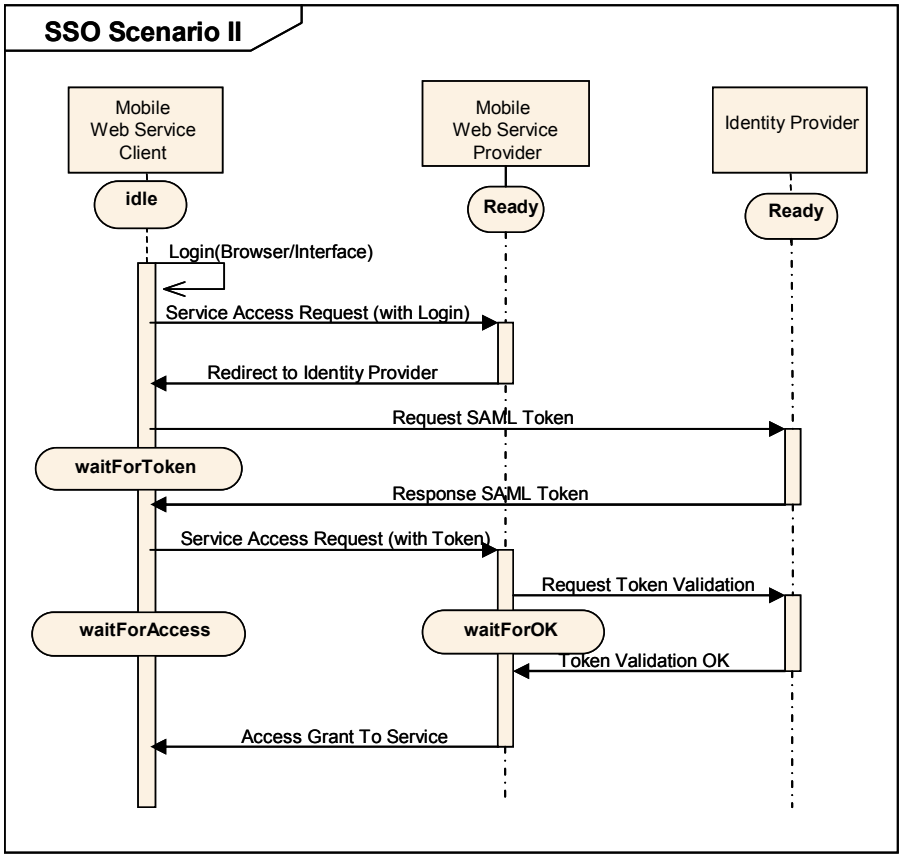


Figure 18 : Single Sign-on Design Scenario II

When the SourceID toolkit's Service Provider features are not feasible on the Mobile Host, the following scenarios will be realizable. In these scenarios, an additional third party standalone component is required. The primary purpose of this component would be to validate the client's SAML token sent from the Mobile Host. There will be a

mutual trust between the Mobile Host and the third party standalone component thereby helping the resource constrained Mobile Host in validating the client SAML tokens.

3.3.4 SSO Scenario III

In this third Single Sign-on scenario as shown in Figure 19, the workflow is quite straight-forward too but with an additional component. The Mobile Web Service Client logs in with its generic interface. The interface back-end will request SAML token from the Identity Provider. After receiving the SAML token, the client then requests for the service access from the Mobile Web Service Provider (Mobile Host) by providing the SAML token. The Mobile Host simply forwards the token to its mutually trusted third party component. This third party standalone component will then request the client token validation with the Identity Provider for its authenticity. Upon positive confirmation from the Identity Provider, the component sends an OK to the Mobile Host. The Mobile Host will then grant the service access to the Mobile Web Service Client.

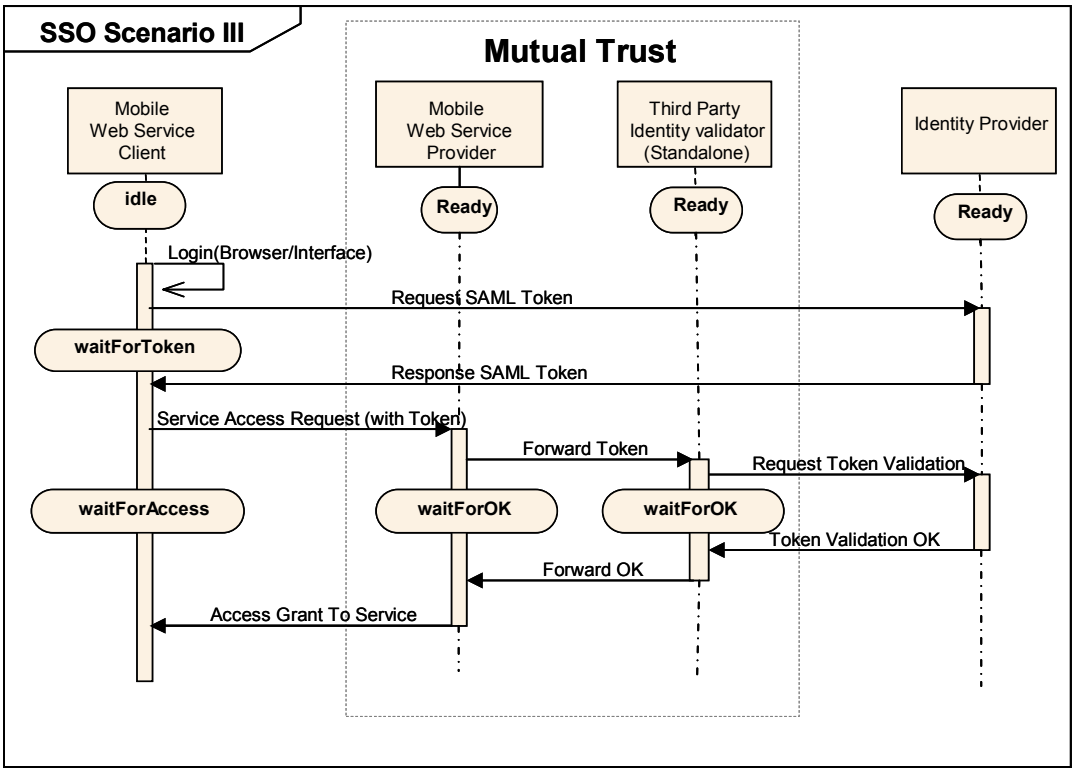


Figure 19 : Single Sign-on Design Scenario III

3.3.5 SSO Scenario IV

Figure 20 depicts the fourth Single Sign-on scenario. This workflow with the additional component is required when the client does not have information about the Identity Provider. The Mobile Web Service Client logs in with its generic interface. The

interface back-end will then request the service access with the Mobile Host. As the Mobile Host fails to find a SAML token, it sends the response redirecting to Identity Provider. Then the interface back-end will request SAML token from the Identity Provider. After receiving the SAML token, the client then requests for the service access from the Mobile Web Service Provider (Mobile Host) by providing the SAML token. The Mobile Host simply forwards the token to its mutually trusted third party component. This third party standalone component will then request the client token validation with the Identity Provider for its authenticity. Upon positive confirmation from the Identity Provider, the component sends an OK to the Mobile Host. The Mobile Host will then grant the service access to the Mobile Web Service Client.

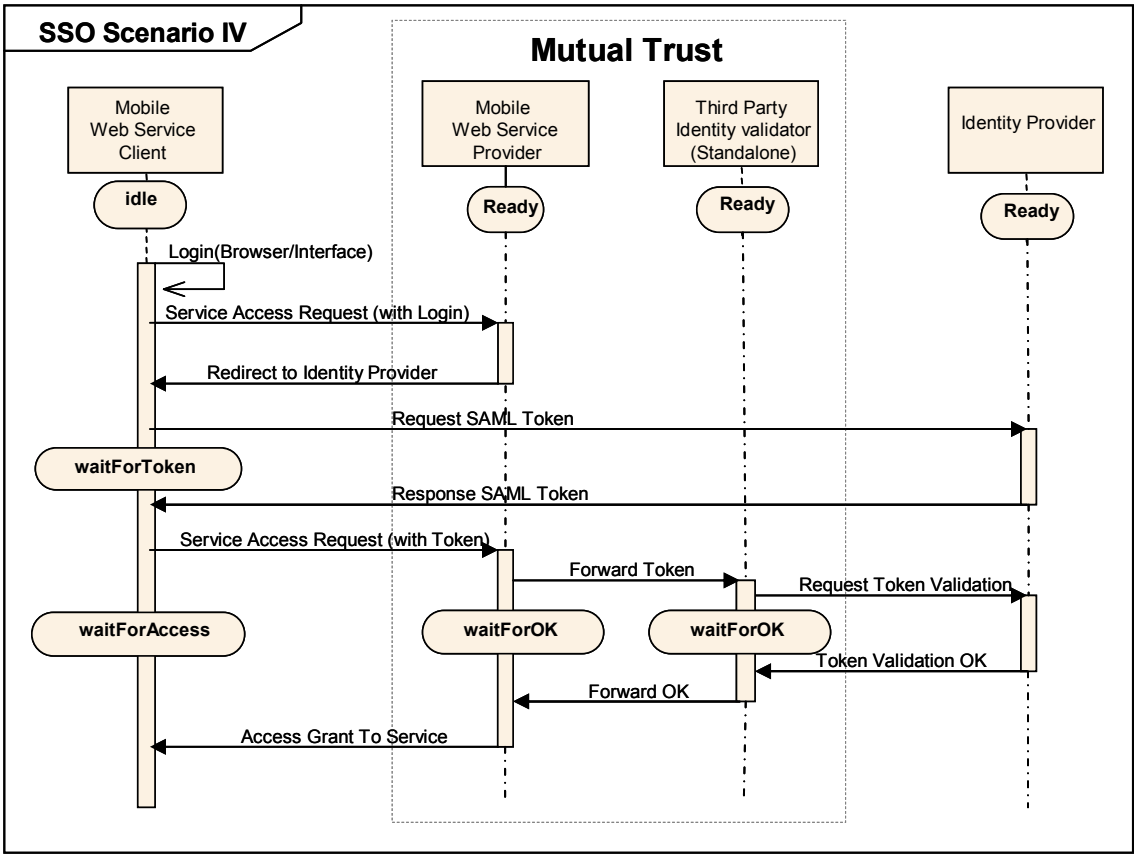


Figure 20 : Single Sign-on Design Scenario IV

Summary:

This section, in summary, addressed the basic security requirements for mobile web services and then discussed both message-level security design models and end-point security design models. It also explained some of the used existing technologies such as Public Key Infrastructure, Public Key Encryption and Digital signature workflows, and SourceID Liberty Beta toolkit.

4 Mobile WS-Security Implementation

This section runs through explaining the thesis related development tools and platforms and WS message-level security implementation details with relevant implementation model and details with a class diagram. The section further explained the end-point security implementation model with a Single Sign-on scenario.

4.1 Development Tools/Platform

This subsection explains the J2ME platform in detail and describes the Sun Java Wireless toolkit and Lightweight Bouncycastle Cryptographic API. Further the subsection addresses the development of adapted KSOAP2 API from existing KSOAP2 explaining the KSOAP2 and KXML2.

4.1.1 J2ME – Java 2 Platform, Micro Edition

Java 2 Micro Edition is the newest and smallest addition to the Java family. J2ME is intended for small and constrained devices and it provides an application environment for applications on consumer and embedded devices, like mobile phones and Personal Digital Assistants (PDAs) [45]. J2ME contains a set of standard Java APIs focusing on these devices. J2ME consists of configurations, profiles and optional packages [45], as shown in Figure 21, where each combination is optimized for the memory, processing power, and I/O capabilities. The configurations and the profiles provide information about APIs and different families of devices [46].

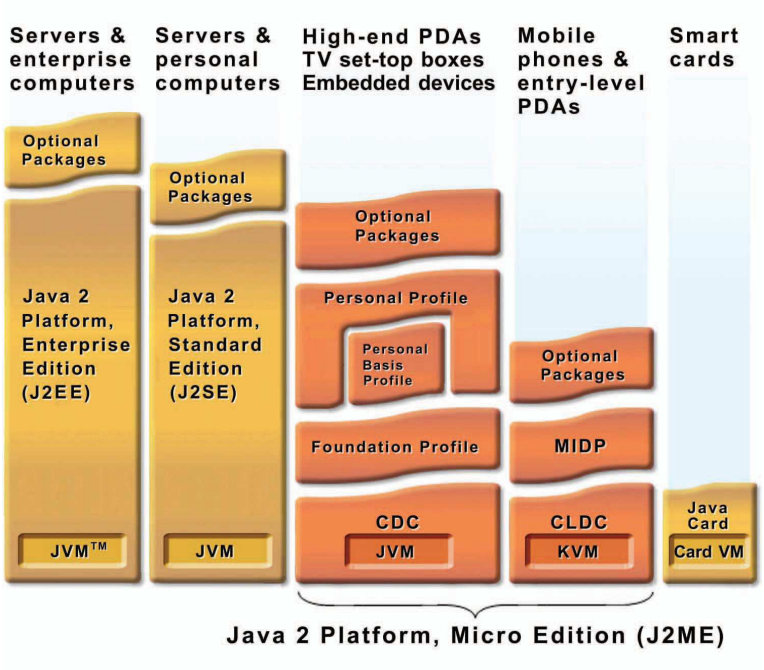


Figure 21 : The J2ME Architecture [45]

A configuration consists of a virtual machine and a minimal set of class libraries designed to provide the base functionality for a distinct set of devices with similar characteristics, such as network connectivity, processor power and memory [44]. Profiles are a set of higher level APIs, and they must be combined with the configurations, in order provide a complete runtime environment. The configurations from J2ME platform are Connected Limited Device Configuration (CLDC) and Connected Device Configuration (CDC) as shown in Figure 21. CLDC is the smaller configuration, and it is designed for devices with intermittent network connections, slow processor and limited memory. Mobile phones, two way pagers and PDAs are typical devices. It defines the minimum required Java technology consisting of libraries and components for small-connected devices [44]. CDC is designed for devices like high-end PDAs with more memory, faster processors and greater network bandwidth. CDC includes a more complete JVM and a larger subset of the Java 2 Standard Edition (J2SE) platform.

As our Mobile Host realization is on Sony Ericsson P910i terminal, our interest will be focused on CLDC 1.0 configuration and Mobile Information Device Profile (MIDP 2.0) profile which are P910i compatible. MIDP profile is designed for mobile phones and entry-level PDAs. It contains the core functionality needed by mobile applications, like the user interface, local data storage, network connectivity, and application management.

A MIDP application is called a MIDlet. It has access to packages from both the CLDC and the MIDP, as shown in Figure 22. The CLDC defines the core APIs where most of them come from J2SE [46]:

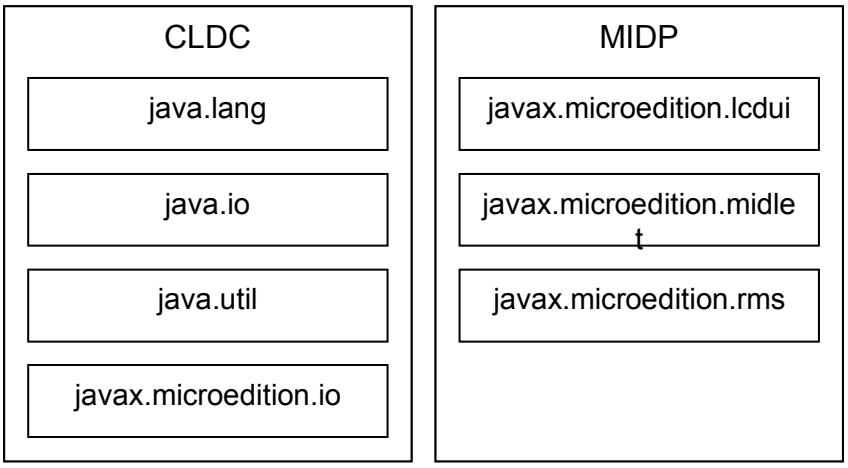


Figure 22 : MIDP Packages [46]

The characteristics of MIDP are [46]:

- 128 KB of non-volatile memory for the MIDP implementation
- 32 KB of volatile memory for persistent data
- a screen of at least 96x54 pixels
- some capacity for input, either by keypad, keyboard, or touch screen

- two-way network connection, possibly intermittent

MIDP offers portability, which is achieved through Java. An application that uses the MIDP APIs will be portable to any MIDP device. MIDP allows the execution of multiple MIDlets. The model defines how the MIDlet is packaged, what runtime environment is available, and how it should behave when resources are constrained. The model also defines how MIDlets can be packaged together in suites and how to share common resources. Each MIDlet suite has also a JAD file, which is a descriptor file that allows application management software (AMS) on the device to identify what it is about to install prior to installation. The model also defines a lifecycle for a MIDlet which allows starting, stopping and cleanup of a MIDlet [44].

The MIDlet life cycle:

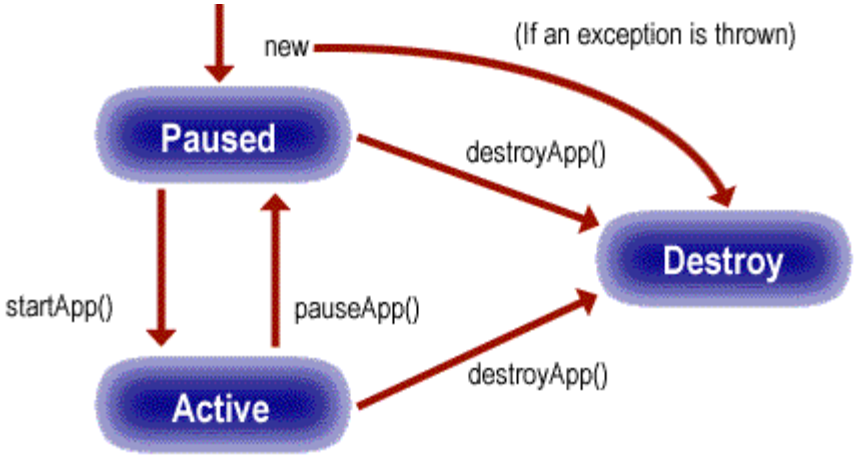


Figure 23 : The MIDlet life cycle [47]

A MIDlet is managed by the Java Application Manager, which executes the MIDlet and controls its life cycle. The MIDlet can be in one of the following states: paused, active, or destroyed. When you first create and initialize a MIDlet, it is in the paused state. If an exception occurs in the MIDlet's constructor, the MIDlet enters the destroyed state and is discarded. The MIDlet enters the active state from the paused state when its startApp() method call is completed, and the MIDlet can function normally. The MIDlet can enter the destroyed state upon completion of the destroyApp (Boolean condition) method. This method releases all held resources and performs any necessary cleanup. If the condition argument is true, the MIDlet always enters the destroyed state [47]. Figure 23 illustrates the various states of a typical MIDlet life cycle.

4.1.2 Sun Java Wireless Toolkit

The Java 2, Micro Edition Wireless Toolkit [48] is a bunch of tools for creating Java applications that run on Java Technology for the Wireless Industry specification compliant devices. It enables user-friendly development environment for programmers to design applications on J2ME devices with the help of its build tools, utilities and a device

emulator for exact offline simulation of the J2ME device to test MIDP applications. The toolkit supports both versions of CLDC (CLDC1.0 and CLDC1.1) and MIDP (MIDP1.0 and MIDP2.0). The other APIs which the toolkit supports are Wireless Messaging API, Mobile Media API, and PDA Optional Packages for the J2ME Platform which consists of File access mechanism, Bluetooth and 3D APIs.

All we got to do is write the source code and the complete building and packaging will be taken care by the toolkit which includes compilation, pre-verification of class files, and packaging the MIDlet suite. We can also run the MIDlet suite directly in the emulator and can analyze the operation of the MIDlets with the help of memory monitor and network monitor provided by the toolkit. The toolkit also contains tools to test the operations of MIDlets in protected domains by signing MIDlet suites cryptographically.

The toolkit also supports the usage of obfuscation. Obfuscation is needed to reduce the size of class files. This is essential because most of the times a MIDlet suite needs to be compact to minimize download times and to comply with the stringent limits on JAR size by the device manufacturers. Obfuscators help this requirement by keeping MIDlet suite JAR quite small when compared to original JAR without obfuscation. Pro Guard obfuscator is one of them, which can be readily configured to use. The obfuscation will be carried out during the packaging process of a development cycle.

4.1.3 Lightweight Bouncycastle Cryptographic API

The Bouncy Castle Crypto package is a Java implementation of cryptographic algorithms. The package is organized such that it contains a light-weight API suitable for use in J2ME environment with the additional infrastructure to conform the algorithms to the Java Cryptography Extension framework. The light-weight cryptographic API consisting of support for the following [49]:

- BlockCipher
- BufferedBlockCipher
- AsymmetricBlockCipher
- BufferedAsymmetricBlockCipher
- StreamCipher
- BufferedStreamCipher
- KeyAgreement
- Integrated Encryption Scheme Cipher (IESCipher)
- Digest
- Message Authentication Code (Mac)
- Password Based Encryption (PBE)

Some of the algorithms of light-weight cryptographic API which are relevant to my current research will be discussed in brief here. Out of the Symmetric Block algorithms, the basic interface of interest is BlockCipher and has various implementation modes. The mode we are interested in is BufferedBlockCipher with default PKCS5/7 padding termed as PaddedBufferedBlockCipher. The cipher engines that we have implement using this mode are AESEngine, DESEngine, DESedeEngine, and IDEAEngine with keysizes varying from 64 bit to 256 bit subjectively.

We have used BufferedAsymmetricBlockCipher mode from Asymmetric Block algorithms whose basic interface is AsymmetricBlockCipher. RSAEngine with 1024 bit and 2048 bit implemented using this mode is the engine used for public key encryption. In addition, we have used SHA1 digest algorithm which is implemented by basic interface Digest. Out of supported Signers from the package, we made use of DSA signers and RSA signers for signing purposes.

To conclude this section, the usage of this light-weight API along with KSOAP2 API to implement WS-message-level security in J2ME has resulted in Adapted KSOAP2 API which is explained in detail in the next subsection.

4.1.4 Adapted KSOAP2 API

As mentioned in the earlier sections, SOAP structure is based on XML and includes the most of the security information in its header. For standalone web services, there are a lot of tools and APIs available such as WSS4J by Apache group, JWSDP from SUN networks etc. in Java oriented development of secure web services supporting WS-Security specifications. The point to note is that they are still not yet commercialized.

On mobile front, there are some toolkits to support the normal web services but without security. The most noted ones are KSOAP and SUN related toolkits. As of SUN, the specifications like JSR 172 has been introduced to support web services.

This JSR is designed to provide an infrastructure as two optional packages for J2ME to [51]:-

- provide basic XML processing capabilities
- enable reuse of web service concepts when designing J2ME clients to enterprise services
- provide APIs and conventions for programming J2ME clients of enterprise services
- adhere to web service standards and conventions around which the web services and Java developer community is consolidating
- enable interoperability of J2ME clients with web services
- provide a programming model for J2ME client communication with web services, consistent with that for other Java clients such as J2SE

The Sun Java Wireless Toolkit which we used for the development ease supports this web service specification. But, as our mobile web server is based on KSOAP which is based on KXML in personal java environment coupled with the problem that jsr 172's primary provision is towards J2ME clients, we chose KSOAP2. As personal java is no more in use, we transformed the Mobile Host using J2ME. Detailed information about the transformation will be discussed in later sections. As of toolkits to handle SOAP structure and its security specification according to WS-Security and SAML specifications, neither KSOAP nor SUN existing toolkits are sufficient. After intensive research in this domain, we decided to adapt KSOAP2 which is an extended version of KSOAP API. It is based on KXML2.

4.1.4.1 KSOAP2

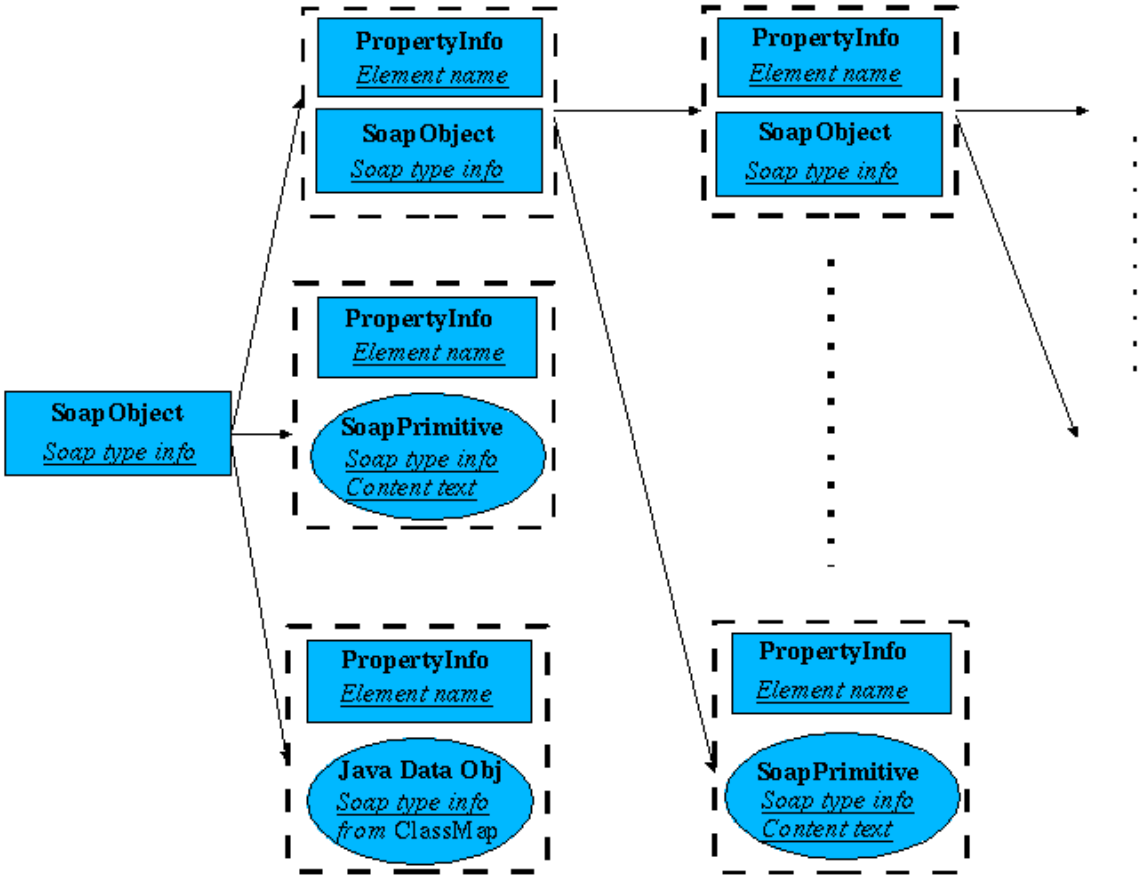


Figure 24 : A Typical KSOAP2 body Structure [50]

The above Figure 24 represents a typical KSOAP2 body structure. Similar to KSOAP [50], it is an open source API for SOAP parsing. Its primary motive is to convert the Java request objects into XML based messages with SOAP structure compliance and revert back the response SOAP complied XML based messages to Java response objects inline to normal SOAP processing. The advanced toolkit also now supports literal encoding. The provision of SOAP serialization support is made optional and integrated several classes into SOAPSerializationEnvelope class whose base class is SOAPEnvelope.

4.1.4.2 KXML2

KXML2 is a small XML pull parser specially designed for constrained environments, to access, parse, and display XML files for J2ME devices. As stated on xmlpull.org, the Common API for XML Pull Parsing (XmlPull) is an effort to define a simple and elegant pull parsing API that will provide a standardized method to do XML pull parsing from J2ME to J2EE. It is a minimal API, one that is easy to implement as a

stand-alone API or on top of an existing parser. XmlPull allows both fast, high-level iteration (using next() method) and low-level tokenizing (using the nextToken() token). It is designed for easy building on top of SAX, XML pull parsers that use iterators with event objects, and even DOM implementations.

General XML content can be parsed with the XML pull API using a loop advancing to the next event and a switch statement that depends on the event type. However, when using XML for data transfer (in contrast to text documents), most XML elements contain either only text or only other elements (possibly with further sub-elements). For those common cases, the parsing process can be simplified significantly by using the XmlPull API methods *nextTag* and *nextText*. Additionally, the method *require()* may optionally be used to assert a certain parser state. The following sample illustrates both situations and methods. The outer element *elements* has element-only content; the contained *text*-elements have text-only content:

```
<elements>
  <text>text1</text>
  <text>text2</text>
</elements>
```

The relevant parser methods here are:

```
nextTag()
nextText()
require()
```

nextTag() advances to the next start or end tag, skipping insignificant events such as white space, comments and PIs. *nextText()* requires that the current position is a start tag. It returns the text content of the corresponding element. The post condition is that the current position is an end tag. Please note that the calls *require()* are optional assertions, they may be left out completely.

4.1.4.3 Custom SOAP Envelope

As our current domain is mobile web service security, there is no literal support from KSOAP2. So, for our research purposes, we used only SOAPEnvelope class from KSOAP2 which contains core Write and Parse methods using XMLPullParser. Also note that in KSOAP2, SOAP headers are handled as elements and SOAP body as objects. For SOAP WS-Security compliance, we handled SOAP body in elements as well. This is the only class we have adapted into a new class called CustomSoapEnvelope to serve the purpose of serializing and de-serializing the Java Objects and SOAP messages accordingly.

The methods of CustomSoapEnvelope are simple:

```
Write()
WriteHeader()
WriteBody()

Parse()
ParseHeader()
```

`ParseBody()`

The Write() and its sub functions convert the Java Objects into XML data to streaming via HTTP protocol. Parse() and its relevant functions transforms the received XML stream in to Java Objects. As the mentioned Java Objects in here are Java Element Objects, the setup effectively helped us to setup SOAP message structure very much according to WS-Security and SAML specifications. This will leave the complete Java Object Interpretation wide open. The Java Object creation and interpretation according to WS-Security specifications is handled in our mobile web service security API which will be explained in the forthcoming sections.

4.2 WS Message-level Security Implementation

As there are not any changes in web service message-level security implementation as described in message-level security design model section when shifted the implementation from mobile client to Mobile Host, the implementation details in this subsection will refer to Mobile Host while describing the coding process. The following subsections will explain the implementation model and the abstract coding details with a class diagram.

4.2.1 Implementation Model

To realise WS message-level security for Mobile Host, on Sony Ericsson P910i, as mentioned in earlier sections, we have used J2ME MIDP2.0 for implementation. The device supports MIDP2.0 with CLDC1.0 configuration. For cryptographic algorithms and digital signers, java based light weight cryptographic API from Bouncy Castle crypto package is used. KSoap2, the java API based on KXml2, is adapted by us according to WS-Security standard and utilized to create the request/response web service messages.

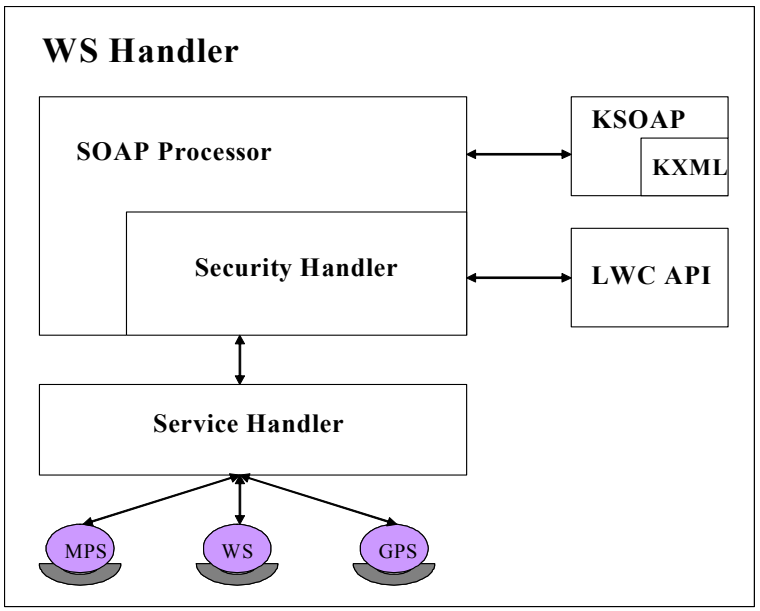


Figure 25 : Web Service Handler of the Mobile Host

The web service security enabled WS Handler of the Mobile Host is shown in Figure 25. The SOAP Processor extracts the SOAP messages from web service requests. The Security Handler does the respective security tasks/checks over the message and transfers decrypted message to the service handler, which extracts the service details and invokes the respective service. Effectively, the handler manages the full message-level security.

To realize confidentiality, the message was ciphered with symmetric encryption algorithm and the generated symmetric key is exchanged by means of asymmetric encryption method. The message was tested against various symmetric encryption algorithms including the WS-Security mandatory algorithms, namely, TRIPLEDES [52], AES-128, AES-192 and AES-256 [53]. The PKI algorithm used for key exchange was RSA-V1.5 [54] with 1024 and 2048 bit keys. Upon successful deployment and testing of confidentiality, we considered data integrity on top of confidentiality. The messages were digitally signed and tested against two signer algorithms, namely, DSAwithSHA1 (DSS) [55] and RSAwithSHA1 signature algorithms. Note that, as said earlier, all the algorithms mentioned above have been implemented using java based light weight bouncy castle cryptographic API.

4.2.2 Implementation Details

This subsection explains the implementation details of the complete message-level security whose class hierarchy is bundled in to a single package including adapted ksoap2, security API and the test bed to fetch the experimental results and presents the resultant SOAP message structure when fed with message security.

4.2.2.1 General provisions of the Security API

The Security API provides the following features for the Mobile host to handle message-level security:

- Conforms the soap message according to the web service standards
- Capable of handling all the symmetric encryption algorithms as specified by WS-standards.
- Supports symmetric key generation
- Provides symmetric key exchange using Public Key Infrastructure
- Supports both RSA and DSA signature for message integrity domain.
- Provides file access capability for fetching digital certificates or PKI key-pair values.

4.2.2.2 The Security API Package Analysis

As said earlier, the entire message-level security functionality of the Mobile Host is implemented in a single package. The class diagram of the package is shown in figure 26 which provides overview of all the classes and their functionality.

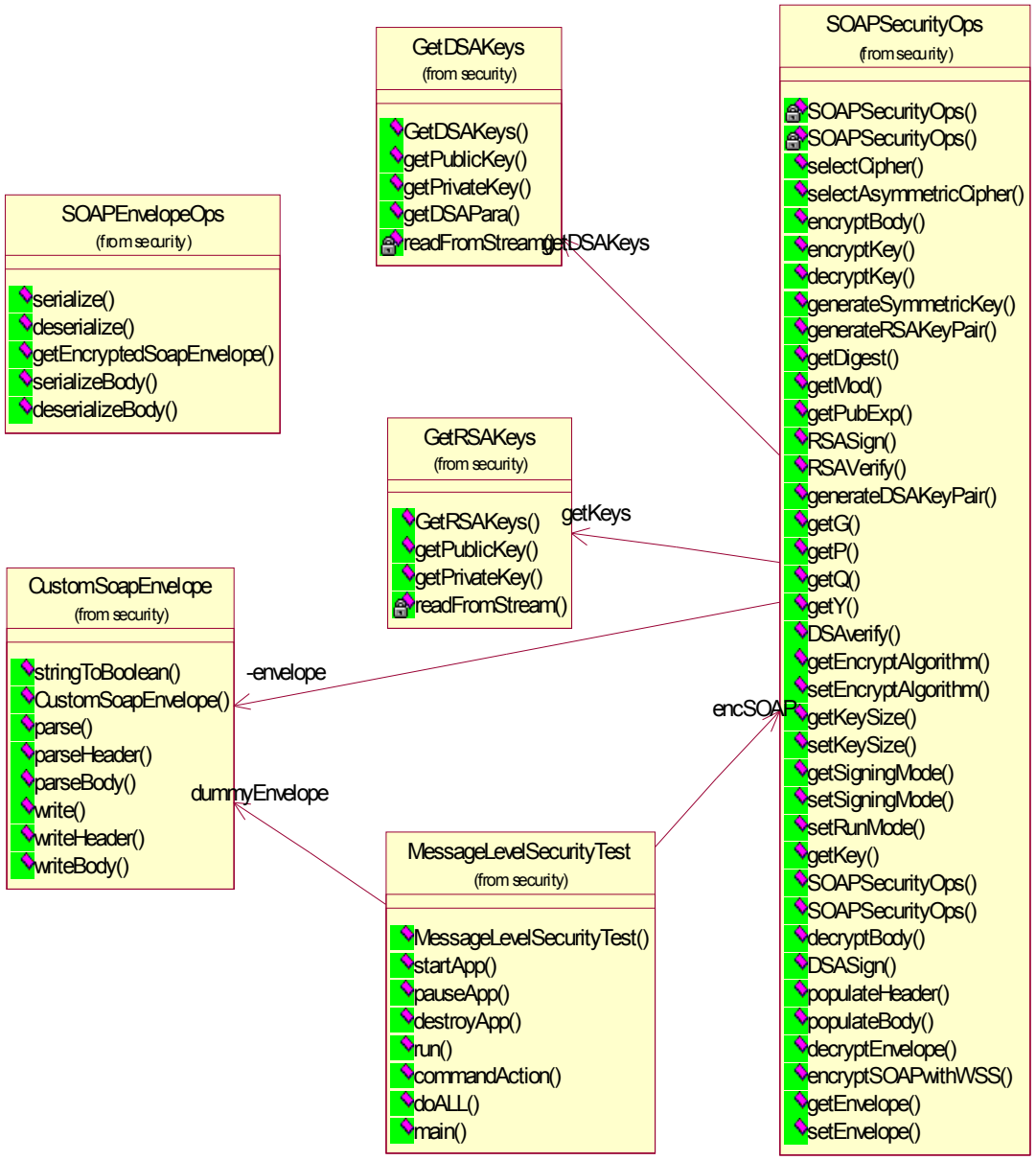


Figure 26 : Class Diagram of Security API Package

SOAPSecurityOps:

SOAPSecurityOps is the main class which reflects the core message-level security API. The class has two primary provisions. One is to take a normal soap message, adds the WS message-level security information and will return back the result WS message-level security enabled soap message. Two is receive message-level security enabled soap message, process the security information from the message and send out the result as normal soap message.

In order to work with this class, some initial class variables which are assigned to default values need to be configured for custom purposes. They are symmetric algorithm, its key size, and signing mode. One has to also configure the path to access public and private keys for asymmetric encryption.

The complete process of the class is explained here. Note that one can also access individual operations to have custom security depending on their application and domain scenario. In case of adding security information to soap message, the class first extracts the normal soap body and populates the security enabled soap body. To carry out this, first a symmetric key will be generated according the specified key size and then the normal soap body is encrypted using the generated symmetric key with the specified symmetric algorithm. After this step, the class populates security enabled soap header. In the resultant soap header, the following will be added. First the symmetric key is encrypted with default asymmetric algorithm using private key. Then the encrypted body will be signed using one of the signers from DSA and RSA signers. To achieve the signature, first the body digest will be calculated. In case of RSA signature, the entire body will be signed using RSA signer while only body digest will be signed using DSA signer for DSA signature with their respective public key. The resultant signature value along with encrypted symmetric key and digest value will be filled into the soap header thereby producing the entire WS message-level security enabled soap message.

The de-securitization of the message-level security enabled soap message is as follows. The class first depopulates the soap header. First it verifies the digest value by comparing the sent digest value from the soap header with the calculated soap body digest. Then it fetches the signature value to verify the integrity of the message. In case of RSA signature, the RSA verifier verifies the soap message by using signature value, the recipient's RSA private key and the soap body. For DSA signature, the message is verified with DSA verifier with the help of signature value, the recipient's DSA private key and soap body digest value. Upon successful integrity check, the decryption of encrypted symmetric key takes place using recipient's private key. Using the symmetric key, the encrypted soap body is decrypted using the relevant symmetric algorithm resulting in the normal request/response soap message.

SOAPEnvelopeOps:

SOAPEnvelopeOps class is quite simple and provides two generic functionalities. One is to receive soap envelope and serialize it to byte stream for SOAP/HTTP transfer. The provision of serializing either the complete envelope or only the SOAP body is available for specific usage. The second functionality de-serializes a byte stream received into a soap envelope or soap body depending on the subject.

GetRSAKeys and GetDSAKeys:

These two classes are basically required to fetch the public and private keys from a location on the mobile for achieving asymmetric encryption or decryption and RSA or DSA signature evaluation. As file access in not supported on our current test bed, Sony Ericsson p910i mobile, the classes generate RSA and DSA key-pair. Since, this problem would not arise in commercial picture, the timestamps took to generate these key-pairs are ignored. These are further explained in forthcoming evaluation sections' test-bed and test-case subsection.

MessageLevelSecurityTest:

MessageLevelSecurityTest class is the actual test bed to evaluate the message-level security scenario on the Mobile Host. The test application is a simple sum function. For simplicity, both client and service provider are considered on the Mobile Host itself ignoring network delay which is already realized [1]. The class takes care of generating normal soap message, enabling message security using SOAPSecurityOps and serializing it as part of mobile client. It then, assuming as Mobile Host, de-serializes the serialized content, verifies message security using SOAPSecurityOps and reads the resultant normal soap envelope, processes the result, forms the normal result soap message, adds security again and serializes it.

For optimized testing purpose, the class is organized such that the entire cycle mentioned above will run through all specified symmetric algorithms and its varied key sizes and produces a completed result table as output. This helped in alleviating the cost for fetching the results.

4.2.2.3 Resultant SOAP Message Structures

A normal J2ME SOAP message before adding WS-Security information according to advanced KSOAP2 API looks like the following message structure:

Normal SOAP Message:

```
<v:Envelope xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns:d="http://www.w3.org/2001/XMLSchema"
xmlns:c="http://www.w3.org/2001/12/soap-encoding"
xmlns:v="http://www.w3.org/2001/12/soap-envelope">
<v:Header />

<v:Body>
<n0:input xmlns:n0="CustomMethodURL">
<input1>44</input1>
<input2>55</input2>
<bodyPadding>As pervasive mobile data applications are becoming
ubiquitous in parallel with the fast developing and easily readable web
services (WS), the ability to provide secure and reliable communication
across mobile web service applications became utmost important. Even
though a lot of standardized security specifications and implementations
exist for web services in the wired networks, not much has been
standardized in the wireless environments. This thesis report addresses
some of the critical challenges in providing WS-S...</bodyPadding>
</n0:input>
</v:Body>
</v:Envelope>
```

Example 7 : Normal SOAP message structure with Adapted KSOAP2 API

A J2ME SOAP request message structure after encrypting body with AES 256 bit algorithm, RSA 1024 bit symmetric key exchange and signed with RSA signature using the above mentioned security API is represented below:

Message-Level Security Enabled SOAP Message:

```
<v:Envelope xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns:d="http://www.w3.org/2001/XMLSchema"
xmlns:c="http://www.w3.org/2001/12/soap-encoding"
xmlns:v="http://www.w3.org/2001/12/soap-envelope">
<v:Header>
<Security>
<n1:EncryptedKey xmlns:n1="http://www.w3.org/2001/04/xmlenc#">
<EncryptedMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
<CipherData>
<CipherValue>

UlhOLVkNgWdvP/7r8upPYGSwHzKwG/8hTS9i47NebGmswAXNidZC6GXSN8eaG1jWJNgV6F71
vyeUuHOLxhs2EtiohQLstKB9iqSHxT4Jzqy8SFxxOZgjWRBQxsml8aljLlJ96L7pHQijR/CB
rGF1S97haGx4u8fXeQNY+j87cTg=

</CipherValue>
</CipherData>
<ReferenceList>
<DataReference URI="#441252522" />
</ReferenceList>
</n1:EncryptedKey>
<n2:Signature xmlns:n2="http://www.w3.org/2000/09/xmldsig#">
<SignedInfo>
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"
/>
<Reference>
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
<DigestValue>

OG9p3JPjFdtLu2ATbXMAYA1DTzQ=

</DigestValue>
</Reference>
</SignedInfo>
<SignatureValue>

lPfksxdp53VSscTAhHHJYYfILT+v6xidBVZ0eYPEapXUO8S5XBxvhfXP8GdQoKXv70PA4CHQ
/c5mUhfxPVqzSk2B5GaXqxeLaRnlE3KktNT/Mbo6329MNPblyLbuDIwr+9/Y2V9QfeT54FD0
BxXZ2VZ8IFft4x63xBgmzEjNWrw=

</SignatureValue>
<KeyInfo>
<KeyValue>
<RSAKeyValue>
<Modulus>
AJ64wyiDicWPvy8jfAShT1/pPg6izOMTqiBUHGR5248nU+z0wSObZyKik25j6vwG0dnKb9mm
WVLP/AzHDl3iT8vlOcJp6dGANU4GnOyqEQ9Oy+2pvvGmOHLUedvSKIUKctxbT7UIkVYtkgnw
Ja3VfKXz7oIyXRaa8AXOdhZ1QopR
</Modulus>
<Exponent>AQAB</Exponent>
</RSAKeyValue>
</KeyValue>
</KeyInfo>
</n2:Signature>
```

```
</Security>
</v:Header>

<v:Body>
<n0:EncryptedData Id="223940028"
xmlns:n0="http://www.w3.org/2001/04/xmlenc#">
<EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#AESEngine"
/>
<CipherData>
<CipherValue>

Ye/qF76fWiDXQql54OvjmGar4t9Kdyn5aINxqnyxeIv+UPwlzGusvFoe2lpvw0YjA74y+ei4
dGyDXjREOlvqcWJjDW7N6fTjgVGM/r+5LTcTPBk00QwdwEX0uJpkhp7assGPWTLQPenHqwGU
WfkX3czfzo5GNrxwEgRBGqsw4cQJplmQIViijSB8vNBzUnYUHKmGrV56IUrnPM3yWHDnklrc
aiqqtVq7Kgl+S8jDQTeJ1AYOKhgNtZt0iozsh+mM7zIr1cVf1OM9RWpBc4kycL/sl0bLXOMA
N2mJ+eQxnkjwtSdUS8F1SUe2mbZMFdPz2PbPKqhHaq6nUd6VVC2a+MOlLwIghrWCfIdK1zlT
6luRs9uY2/biZdjQF+Qzf2iQMdbNLbshtQ/VhNOKtPIHYiZeNE4WP5b+GqISSXCIMLJFtC1E
xCVnQfeF1tfnlGNpAldTDmNMCydXD+FQdPLXwBJL6bjtqvth9fLvR+zLtViufGiI0RnGuVii
2eTUZcxykT+HVGn7Xcg8Qa2yEp+Dv6GO6fi1Lqx4YTpPGc0qb+Qaf6h7cTDaHz7aFdYBz6JG
6bsZh0BG/LE/+IfSFP+LiyroXR0SHSqmd1WzJ8k2rWJLSH4T1jEl/MAv7TDRRjJWrMNciPER
cY6bKQFGVtH1pNPWXRRmflxlcmqczfLqmjA/pG+mBDVzCdGA/DLg02wDhyNn8vl4zCj+NxyI
+ht4+802Rf2eKyHv5Gv8nTdv3UqdKSDlHSLDO02H4clsjoRwTFjpJ7/Q1Vn5DfJIdJXjPkuO
qn/6bkcwHVtkgTdeQn94SwhSjLFZ41BlAPmiuVb/RMESnqklKDtPQ0Pw3u1eVw==

</CipherValue>
</CipherData>
</n0:EncryptedData>
</v:Body>
</v:Envelope>
```

Example 8 : A Typical SOAP message generated by Security API Package

4.3 WS End-point Security Implementation Model

This subsection tries to model the WS end-point security in our domain. The primary design was to analyse to Single Sign-on to provide both authentication and the possible authorization activities. We considered the SSO scenario III and IV as explained in end-point security design models subsection. The SourceID Liberty Beta toolkit is used for the communication between third party identity validation component and the identity provider.

The basic components, as shown in Figure 27, required to analyze the Single Sign-on scenario are the Mobile WS client, Mobile Host (Mobile WS provider), Third-party Identity validator, and Identity provider. The initial federation between the WS client and the identity provider are not explained in the analysis shown in Figure 27. The federation process in brief is as follows. WS client authenticates with third party identity validator and requests for federation with the identity provider which it supports. Then the identity provider requests the WS client to validate its credentials again to federate into the domain. Upon the respective successful handshakes, the identity provider maintains the federated identity of the WS client.

As far as actual Single Sign-on scenario is concerned, two kinds of communication are evident as shown in the Figure 27. One is WS client SOAP request

initiation with the Mobile Host without security token and the other is to initiate the SOAP request with security token. When a WS client, already with the identity provider, requests a web service from the Mobile Host, the request will be parsed to check the security token for authentication. In case of security token not available, the Mobile Host will then requests third party identity validator, a standalone component which as mutual trust with Mobile Host, to authenticate the WS client using Single Sign-on. The third party component will then ask the WS client to choose the relevant IDP with which it is federated. Upon receiving the artifact from the WS client, the third party component then forwards the artifact to the identity provider. The identity provider then requests the Single Sign-on credentials by providing login page to the WS client. After successful verification of client credentials at identity provider, an assertion token will be sent back both to WS client and third party component which forwarded artifact. The WS client uses the received assertion token along with further service requests which conforms Single Sign-on. Meanwhile, the third party component sends an OK to the Mobile Host about the authenticity of the WS client SOAP request. The Mobile Host then grants access permission to WS client for its concerned web service.

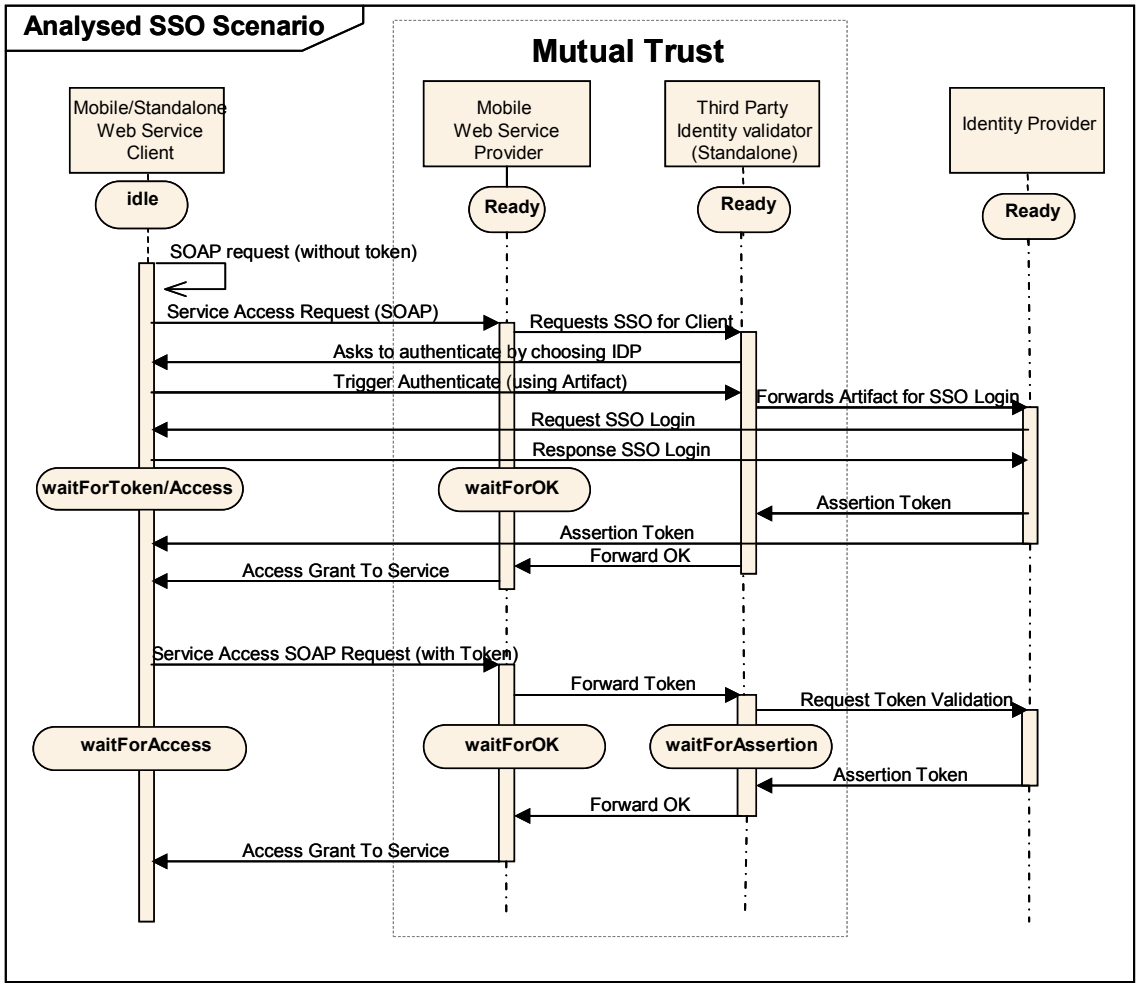


Figure 27 : Analyzed Single Sign-on scenario in Mobile Web Service Provisioning Domain

Once WS client performed Single Sign-on, it can use the assertion token for further web service requests with in the domain. The second kind of communication starts here and is fairly simple. The WS client requests the service by attaching token to the SOAP request. The Mobile Host then forwards the token to third party identity validator for token validation. At third party component, the token is validated with the identity provider for its authenticity. The component then sends an OK upon successful validation to Mobile Host which in turn grants access to the web service to WS client.

To sum up, the Single Sign-on scenario depicted in Figure 27 was partially realized drawing out the feasibility of its full throttle usage. The mutual trust between Mobile Host and the third party component are assumed. Of course, it is even possible to exchange the information between these two mutually trusted components in a secure mode using such as encryption and digital signatures to ensure confidentiality and integrity. And as session maintenance is not possible at Mobile Host, each service request with token from WS client will be validated for the clients' authenticity with the second type of communication mentioned in Figure 27.

Summary:

This implementation section, in summary, covered the implementation model to achieve the message-level security and the explanation of the Security API package with the help of a class diagram and then showed the SOAP message structure with and without security information. Further, the section covered the analyzed Single Sign-on scenario to cover end-point security implementation. Note that authentication is also achievable in much the same way as integrity by sending the username encrypted with private key of the sender as explained in the end-point security design model section.

5 Mobile WS-Security Evaluation

The WS-Security evaluation section evaluates and analysis the experimental results. First, performance model with various timestamp definitions in the test bed will be explained. Then the section describes the generic evaluation model with whole web service cycle including Mobile Host and the client. Further the section ends with experimental results and performance analysis of the Mobile Host.

5.1 Performance Model

To analyze the performance of the Mobile Host with the security load, the durations of different activities during the web service invocation cycle are observed. The client initiates the call for the web service and the Mobile Host processes the request, populates the response, and sends response back to the client.

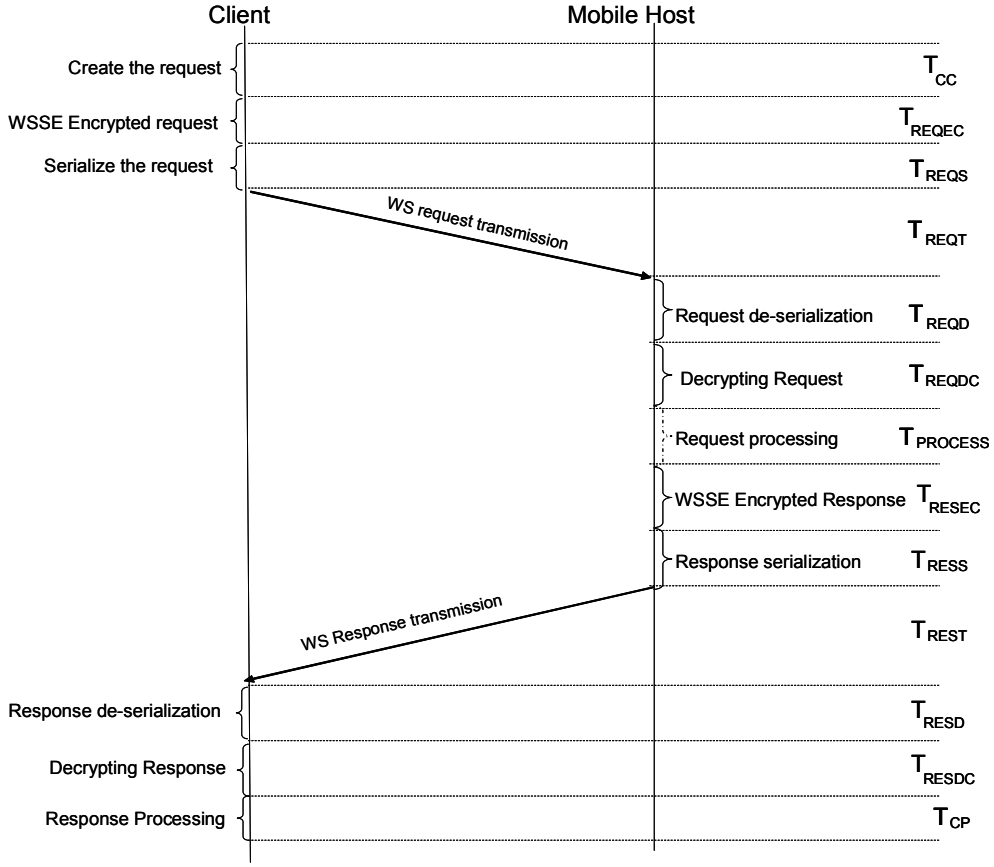


Figure 28 : Secure Mobile Web Service Invocation – Operations & Timestamps

The total time taken for this mobile web service invocation (Tmwsp) constitutes, the time taken by client for constructing valid SOAP message (Tcc), the time taken to encrypt the message with security information according to WS-Security standard (Treqec), the time taken to serialize the encrypted message (Treqs), the time taken to

transmit the SOAP request to Mobile Host (Treqt), the time taken for de-serializing the XML based SOAP request message (Treqd), the time taken to decrypt the request message (Treqdc), the time taken by the Mobile Host to execute the respective business logic and to populate the response (Tprocess), the time taken to encrypt the response message with security information (Tresec), the time taken for serializing the encrypted response message back to XML data streams (Tress), the time taken to transmit the SOAP response back to the client (Trest), the time taken to de-serialize the response at the client (Tresd), the time taken by the client to decrypt the response message (Tresdc), and lastly the time taken by the client to process the response (Tcp). The invocation process is shown in Figure 28 and the total time taken for the mobile web service invocation is given in the following equation 1.

$$Tmwsp = Tcc + Treqec + Treqs + Treqt + Treqd + Treqdc + Tprocess + Tresec + Tress + Trest + Tresd + Tresdc + Tcp$$

Equation 1 : Total Invocation Cycle

The exact estimation of the Treqt and Trest time is not possible as the process needs the synchronization of time stamps of both Mobile Host and client. Moreover these transmission times were observed during previous analysis in Mobile web service provisioning project [58]. Those results showed 90% of total invocation cycle is transmission time. So to analyze the minute extra delays due to security load, the whole invocation cycle is observed with both the invocation and processing of the WS request at the Mobile Host itself, thus eliminating the transmission aspects.

We can derive the pure message-level secured Mobile Host WS effort (Tmhwse) from the invocation cycle explained above. This effort only concerns with de-serializing the incoming request, de-securitizing it, processing the request, creating soap response with MWS-Security, and serializing the response for bit-stream transfer. The resultant performance equation is as follows:

$$Tmhwse \sim= Treqd + Treqdc + Tprocess + Tresec + Tress$$

Equation 2 : Web service effort on Mobile Host

From the invocation cycle, we can also derive the mobile WS message security effort (Tmwsse), consisting of only the MWS-Security handling timestamps which includes creating message with security and processing them at relevant end points, as follows:

$$Tmwsse \sim= Treqec + Treqdc + Tresec + Tresdc$$

Equation 3 : Message-Level Security effort of Mobile WS cycle

The complete experiment results and analysis that will be explained in the forthcoming subsections are entirely dependent on these timestamps explained in this subsection with the invocation cycle.

5.2 Test-bed and Test-case Model

This subsection will discuss the test-bed we used for MWS-Security realization followed by the test-cases we designed to fetch the experimental results which are based on the performance model. The subsection also addresses the problems and unsuccessful methods during the thesis and the temporary solutions to cover them.

Test-bed modules are:

- Sony Ericsson P910i mobile – acted as both Mobile Host and Mobile Client. Its provisions such as touch-screen with a pen, screen width enabled us to deal with test cases faster; while on the contrary, its lack of capability to hand file access (JSR 75) hindered our realization to a certain extent. The device has 64 MB inbuilt internal memory (RAM) and supports GPRS from the inbuilt web browser. 32 MB to 1GB external memory plug-ins' are also possible. The supported cellular bandwidths are 900, 1800 and 1900 MHz.
- Pro Duo mobile memory card – This memory card is used for the manual installation of our applications, on to the mobile, developed on our emulator platform. It also served us to fetch the files containing results from the mobile.
- T-mobile SIM card – for data access via internet network, if any.
- Redirector Application – The Redirector application on the test-bed mobile helped us to print the results either in the console or in to a file located on the mobile itself.

One of the facts is that our smart phone/mobile did not support file access mechanism i.e. JSR 75 specification. This is required to access the key-pairs, stored in a specified location on the mobile, for public key encryption and digital signatures. To overcome this, though it is not practical in commercial applications, we generated the key-pair each time we took a test case.

As we already mentioned in previous sections, not many security implementations were available for mobile web services to readily test our Mobile Host WS-Security performance. So, we decided to test it on a case-by-case scenario. Five test cases have been taken for each scenario. The mean values of these test cases were considered as that scenarios' average performance. Our each test-case general features are:

- As our Mobile Hosts' performance concerning network latency was already realized, we ignored it by considering both Mobile Host and the client on the same test-bed. This also saved our network costs.
- As the test-bed lacked file access feature, we generated the key-pairs required for the public key encryption and signing on the test-bed itself, if any.
- Considers all timestamps mentioned in the invocation cycle mentioned in the performance model section

- Covers all the symmetric algorithms and its key sizes by running that many web service request/response cycles.

The SOAP request/response message sizes considered are 1KB, 2KB, 5KB, and 10KB. Note that these are message sizes before adding WS-Security in to them. Note also that we used padding elements within the request to get the exact size which is also visible in normal SOAP message structure in implementation details subsection [Example 1].

First, we started testing the Mobile Host against confidentiality using symmetric encryption of the SOAP body request. The symmetric algorithms, including the WS-Security mandated ones, with various key sizes we tested are:

- AES with 128, 192 and 256 bit key lengths
- DES with 64 bit key length
- TRIPLEDES with 192 bit key length
- IDEA with 128 and 256 bit key lengths

After successful test-cases against symmetric algorithms, the symmetric key exchange along with symmetric algorithms and different message sizes was considered as the second scenario. To exchange the symmetric key, RSA public key encryption is used with both 1024 and 2048 bits. Both were successful in deployment. But the effort took for 2048 bit RSA encryption is approximately three times when compared to 1024 bit RSA encrytpion as per our initial test-case reports. So we ignored RSA-2048 bit after one test-case as it proved too costly. Note also that, for each message size, a separate test case scenario (mean values of 5 test cases) was taken.

Complete confidentiality feature successful deployment led us to third scenario which tested the integrity including confidentiality. To accomplish this, we used both RSA signature and DSA signature with 1024 bit key length. The test-case configuration was same as the second scenario which includes message sizes, number of test cases etc. Some of the test-bed/case images are available at

Appendix – Test Bed Images.

5.3 Mobile Host Performance Analysis and Evaluation with Message-level Security

This subsection evaluates the message-level security on a complete web service cycle as mentioned in the performance model. For achieving this, different encryption algorithms, signer algorithms and authentication principles were analyzed in the Mobile Host domain. The performance of the Mobile Host was observed during the feasibility analysis, for reasonable quality of service. The parameters of interest were extra delay and variation in stability of the Mobile Host with the introduction of the security overhead. The implemented case-by-case solutions were evaluated recursively. Some of the results are discussed here:

The following feasibility report emphasizes the message-level security analysis against various symmetric algorithms for an entire request-response web service cycle.

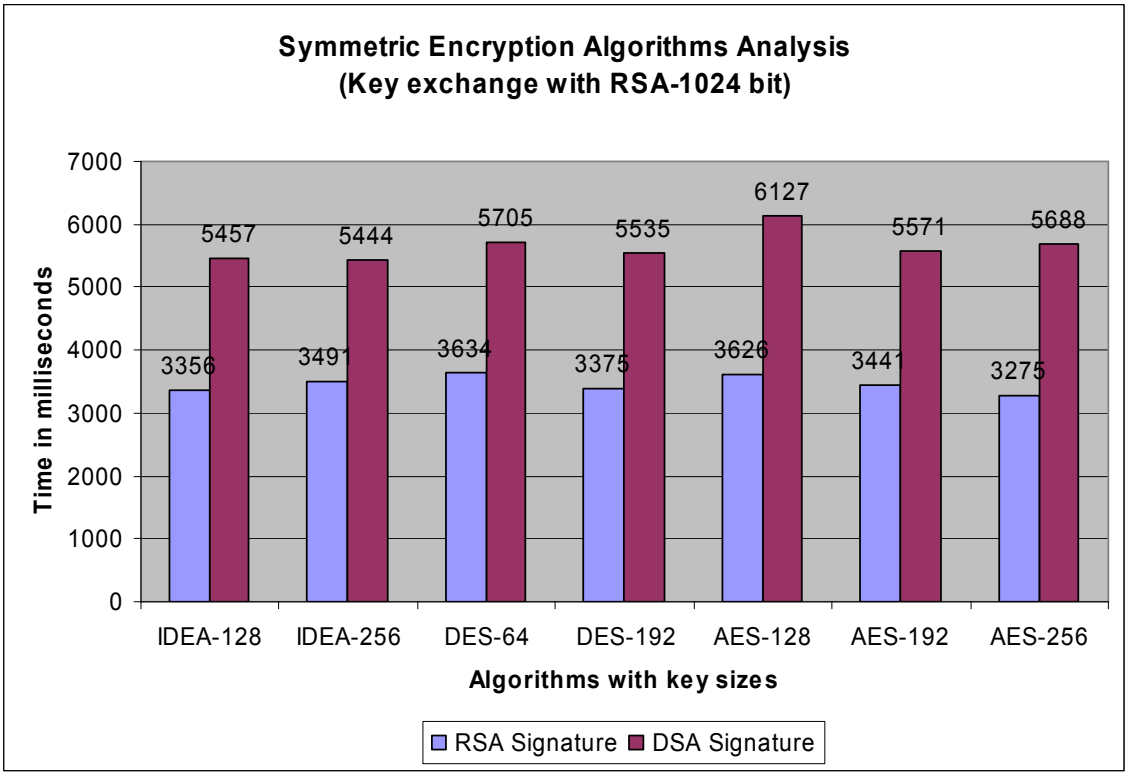


Figure 29 : Comparison of timestamps with various symmetric key algorithms

The analysis shown in Figure 29 emphasizes that not much effort difference exists on security front, out of all symmetric encryption algorithms including WS specific mandatory ones. We can interpret that the best way of securing messages in mobile web service provisioning is to use AES symmetric encryption with 256 bit key, RSA 1024 bit key exchange mechanism and RSAwithSHA1 signature. From the analysis of Figure 29 scenario, we further tried to analyse the individual timestamps of message-level security of a complete web service invocation cycle using the best symmetric encryption algorithm, AES with 256 bit key.

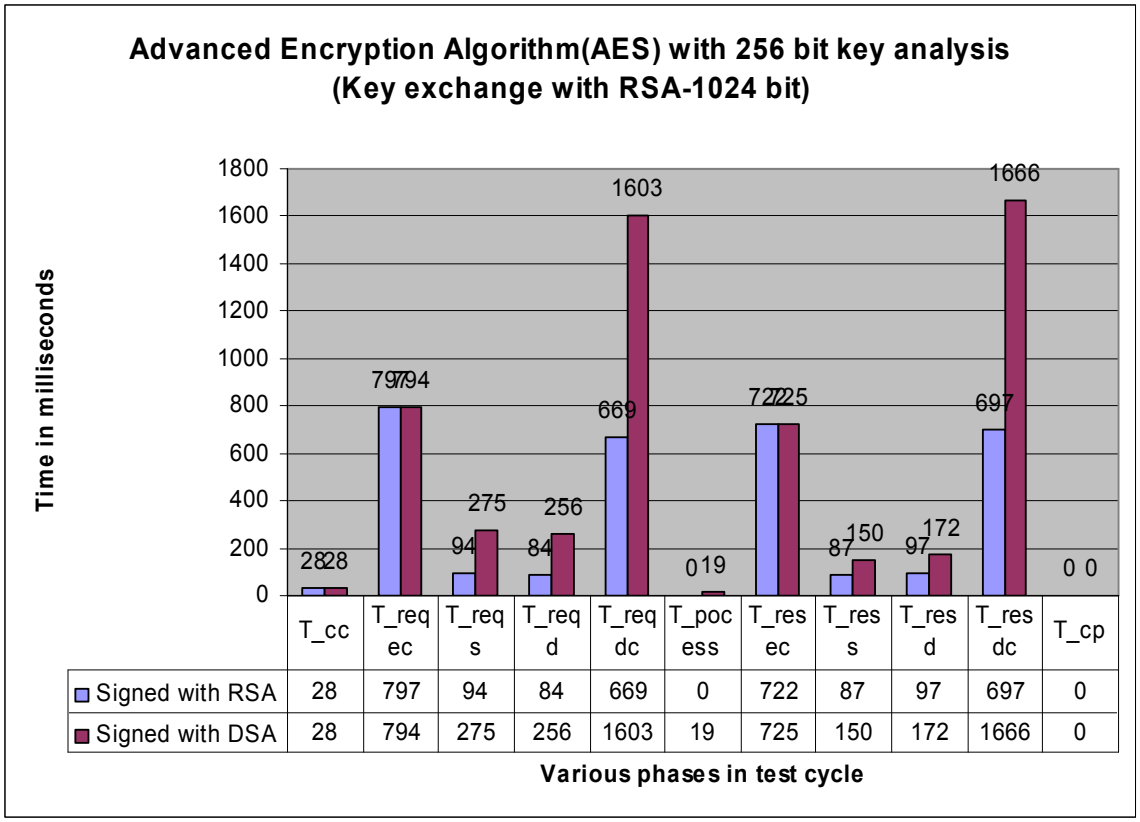


Figure 30 : Timestamps of various phases of a Message-level secured web service cycle

Figure 30 depicts times taken for various phases of a message level secured web service request/response cycle. The original message was ciphered with AES-256 algorithm and its key is exchanged with RSA-1024 PKI algorithm. To summarize further, the request message was 1 KB and response message was 2 KB. The total cycle for highly secured communication, AES-256 bit ciphered, cost around ~3 sec with RSAwithSHA1 signature and ~5.5 sec for DSAwithSHA1 signature. [2]

Further in this subsection, the Mobile Host performance deviations due to message-level security overhead is analyzed. The experimental process contains five stages that are de-serializing the incoming request, verifying the security of the message, processing the soap request and creating response, adding the security back to the soap response, and serializing the soap response.

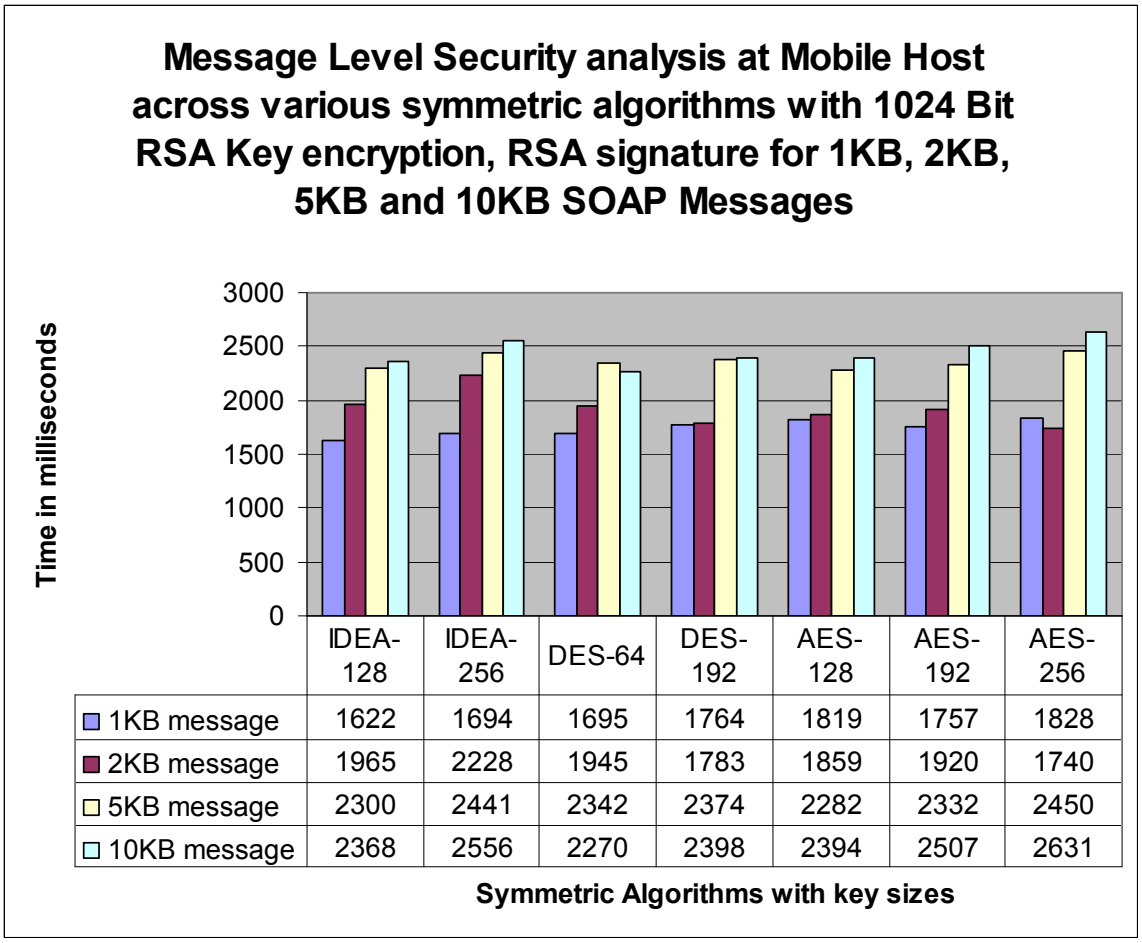


Figure 31 : Timestamps for various message sizes using RSA signature

The following Figure 31 explains various timestamps for different symmetric algorithms on Mobile Host. The test configuration considered here is RSA-1024 key exchange and RSA signature. This test is conducted against varied soap message sizes ranging from 1 to 10 KB. From the outset interpretation of the results, one can visualise more or less linearly dependency of time-cost against soap message size. The time cost up to 2 KB soap message exchange with security looks very much possible.

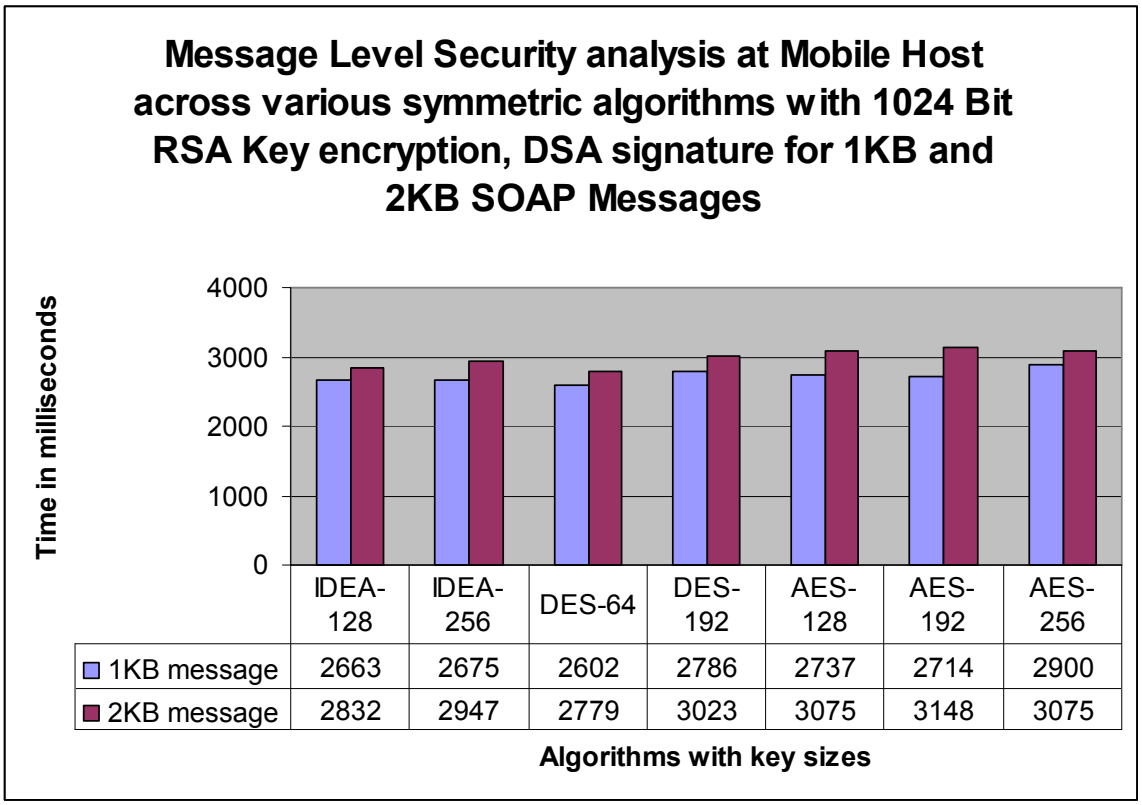
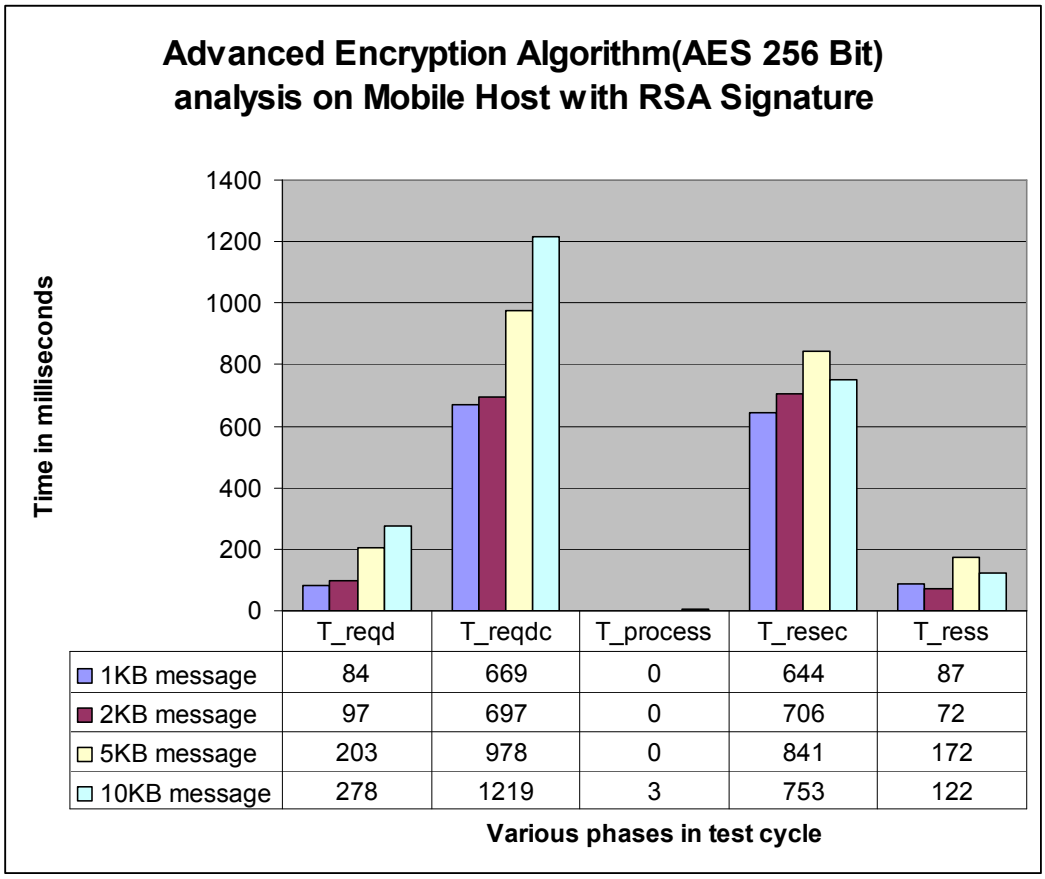


Figure 32 : Timestamps for various message sizes using DSA signature

The Figure 32 reflects the timestamps on Mobile Host when DSA signature mechanism is used. Rest of the configuration is very much the same as the previous configuration details of Figure 31. The observation at these results suggests that the cost of using DSA signature instead of RSA signature is comparatively higher. Further interpretation suggests the cost to achieve to message-level security using DSA signature is approximately 60% more than that of cost using RSA signature. This analysis thus recommends to RSA signature, especially in resource constrained devices such as in Mobile Host domain.



	T_reqd	T_reqdc	T_process	T_resec	T_ress
1KB message	84	669	0	644	87
2KB message	97	697	0	706	72
5KB message	203	978	0	841	172
10KB message	278	1219	3	753	122

Figure 33 : Timestamps for various message sizes using AES-256 encryption and RSA signature

The individual process timestamps of message security effort on Mobile Host are shown in Figure 33. Symmetric algorithm chosen is AES with 256 bit key which is by far the best security provider according to the standards. Rest configuration is RSA 1024 bit public key exchange and signature using RSA signer. To end the message-level security analysis, we can conclude again that the best way of securing messages in mobile web service provisioning is to use AES symmetric encryption with 256 bit key, RSA 1024 bit key exchange mechanism and RSAwithSHA1 signature.

From the evaluation analysis, the extra load to the message size caused by the added security information and the extra delay thus obtained are not of the main concern as this all adds to the transmission delay. With the advent of the interim-generation technologies like GPRS [56] and EDGE [23], and third-generation technologies like UMTS 21, still higher data transmission rates are achieved in the wireless domain, in the order of few hundreds of Kbs to 2 Mbs. Most recently with the advent of 4G technologies and their deployment in south Asian countries suggests that mobile data transmissions of the rate of few GB is also possible [57]. These developments should drastically reduce the transmission delays and thus make the Mobile Host soon realizable in commercial environments also. Based on this, we can say that the additional efforts, as shown in this evaluation model section, in achieving the highest possible secured web service communication, are reasonable.

Summary:

This section can be summarized as follows. The WS-Security evaluation section evaluated and analyzed the experimental results. First, performance model with various timestamp definitions in the test bed are explained. Further, the test bed and test cases scenario are described. Then the section described the generic evaluation model with whole web service cycle including Mobile Host and the client. Further the section ends with experimental results and performance analysis of the Mobile Host.

6 Conclusion

The thesis, as a whole, can be summarized as follows. First we have introduced the base project 'mobile web service provisioning'. The thesis then discussed and analyzed the security challenges of mobile web services and wireless environments. Then the thesis addressed the existing standards and technologies in both web services and mobile domains relevant to WS-Security.

Further, the thesis addressed the basic security requirements to be realised in mobile web service provisioning. The thesis then proposed WS-Security design models including message-level security and end-point security models with various architecures and scenarios. Next, the implementation models are addressed where message-level security implementation is carried out by developing custom Mobile WS-Security API. Further, one Single Sign-on is partially analyzed, due to time constraints, in the mobile web service provisioning domain as part of implementation of the end-point security.

The mobile WS-Security performance evaluation suggested that very basic message-level security in mobile web service domain is very much achievable with little overhead. The study also has drawn out the best way of achieving the confidentiality and integrity of a web service message in the mobile web service domain. The thesis also suggested that basic authentication can also be achieved using Public Key Infrastructure as explained in end-point security design model.

The thesis concludes by emphasizing that the feasibility of Mobile Host performance in handling the WS message-level security is reasonable and in achieving the WS end-point security is possible.

Thus, based on our till-date realization on security awareness, we conclude that secure web service provisioning in mobile networks is a great challenge. And as the mechanisms developed for traditional networks are not always appropriate for the mobile environment, this area still holds ample room for further research.

7 Future Work

The future research in mobile web service provisioning domain includes providing complete end-point security for the Mobile Host using Single Sign-on with SAML and LA standards. We have studied most of the existing and developing standalone web service Single Sign-on mechanisms, out of which, we selected the optimistic SourceID toolkit to achieve Single Sign-on in our domain. We designed four scenarios to achieve Single Sign-on in this thesis. But, in the implementation phase, we could analyze only one scenario partially due to time constraints. The complete implementation and evaluation of all the Single Sign-on scenarios are to be analyzed further. After achieving Single Sign-on, full fledged detailed performance analysis of the Mobile Host with full security features through real-time applications is achievable.

One of the striking factors which affect mobile web services is scalability. In scalability terms, mainly the web service overhead aspect should be considered in future work. Since both SOAP and WSDL are XML-based, a verbose protocol, XML messages have to be parsed on both the server and the client side and proxies have to be generated on the client side before any communication can take place. The XML parsing at runtime requires additional processing time, which may result in longer response time of the server in case of a web service server.

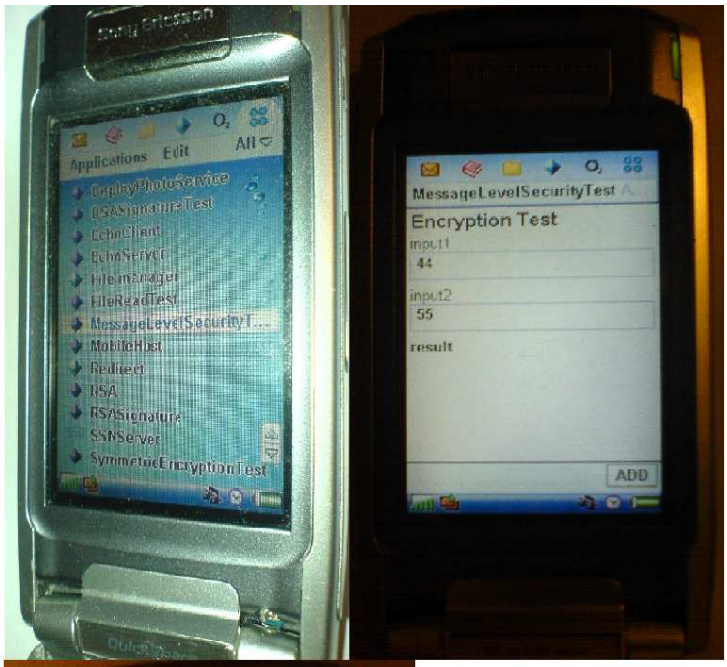
The growth of the web service message size, which results in higher data transmission time, creates a critical problem for delay sensitive applications. One way to achieve a compact and efficient representation is to compress XML − especially when the CPU overhead required for compression is less than the network latency. Compression is both useful for mobile devices that are poorly connected as well as for devices that are charged by volume and not by connection time by their providers. The latter group contains mobile users connected with handheld devices such as people accessing a service via GPRS. This group of users is expected to increase rapidly in the next years. However, the web service application on the server does not have any information about the delay, for example the current round trip time estimated by TCP, and about the available bandwidth between client and server. These overcomings need to be addressed for having better scenarios in real-time applications in mobile web service provisioning domain.

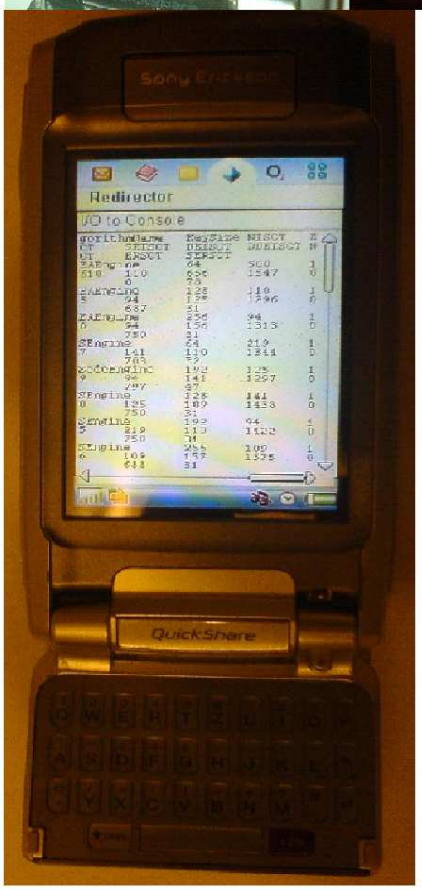
List of Figures

List of Equations

List of Examples

Appendix – Test Bed Images





NISCT - Normal Input SOAP Creation Time
EISCT - Encrypted Input SOAP Creation Time
SEISCT - Serialized Encrypted Input SOAP Creation Time
DEISCT - Deserialized Encrypted Input SOAP Creation Time
DDEISCT - Decrypting Deserialized Input SOAP Creation Time
NRSCT - Normal Result SOAP Creation Time
ERSCT - Encrypted Result SOAP Creation Time
SERSCT - Serialized Encrypted Result SOAP Creation Time

Test Case 1

AlgorithmName	KeySize	NISCT (In millisec)	EISCT (In millisec)	SEISCT (In millisec)	DEISCT (In millisec)	DDEISCT (In millisec)	NRSCT (In millisec)	ERSCT (In millisec)	SERSCT (In millisec)	Total time
IDEAEngine	64	62	135422	453	500	1094	15	50156	141	187907
IDEAEngine	128	0	750	125	140	735	0	657	125	2660
IDEAEngine	256	0	797	125	156	891	0	719	125	3069
DESEngine	64	0	844	125	125	703	0	672	109	2642
DESedeEngine	192	0	750	109	157	704	0	672	109	2693
AESEngine	128	0	891	109	125	766	0	687	125	2831
AESEngine	192	16	719	250	125	688	0	687	125	2802
AESEngine	256	0	750	125	125	688	0	687	109	2740

Test Case 2

AlgorithmName	KeySize	NISCT (In millisec)	EISCT (In millisec)	SEISCT (In millisec)	DEISCT (In millisec)	DDEISCT (In millisec)	NRSCT (In millisec)	ERSCT (In millisec)	SERSCT (In millisec)	Total time
IDEAEngine	64	47	23328	79	250	953	0	243109	78	267908
IDEAEngine	128	0	688	94	93	672	0	594	47	2316
IDEAEngine	256	0	625	47	78	813	0	594	63	2476
DESEngine	64	0	719	47	47	657	0	593	32	2159
DESedeEngine	192	0	640	31	94	656	0	609	78	2300
AESEngine	128	0	782	46	63	844	0	656	47	2566
AESEngine	192	0	703	47	62	750	0	610	47	2411
AESEngine	256	0	640	94	31	625	0	609	78	2333

Test Case 3

AlgorithmName	KeySize	NISCT (In millisec)	EISCT (In millisec)	SEISCT (In millisec)	DEISCT (In millisec)	DDEISCT (In millisec)	NRSCT (In millisec)	ERSCT (In millisec)	SERSCT (In millisec)	Total time
IDEAEngine	64	47	46938	281	281	734	0	173328	250	221923
IDEAEngine	128	16	875	218	235	953	0	765	219	3409
IDEAEngine	256	32	781	203	218	829	0	844	203	3366
DESEngine	64	16	938	203	235	734	0	1016	78	3284
DESedeEngine	192	0	688	31	47	625	0	656	32	2271
AESEngine	128	16	766	47	94	688	0	640	31	2410
AESEngine	192	0	657	46	203	672	0	594	47	2411
AESEngine	256	0	672	109	93	625	0	641	94	2490

Test Case 4

AlgorithmName	KeySize	NISCT (In millisec)	EISCT (In millisec)	SEISCT (In millisec)	DEISCT (In millisec)	DDEISCT (In millisec)	NRSCT (In millisec)	ERSCT (In millisec)	SERSCT (In millisec)	Total time
IDEAEngine	64	47	70156	266	516	1079	32	171953	47	244160
IDEAEngine	128	0	750	47	125	656	0	610	94	2410
IDEAEngine	256	0	688	78	110	718	0	640	78	2568
DESEngine	64	16	687	94	93	625	0	797	94	2470
DESedeEngine	192	16	672	94	78	672	0	640	78	2442
AESEngine	128	0	782	46	94	688	0	1391	250	3379
AESEngine	192	15	969	360	281	985	31	1016	265	4114
AESEngine	256	141	1250	93	94	718	0	625	78	3255

Test Case 5

AlgorithmName	KeySize	NISCT (In millisec)	EISCT (In millisec)	SEISCT (In millisec)	DEISCT (In millisec)	DDEISCT (In millisec)	NRSCT (In millisec)	ERSCT (In millisec)	SERSCT (In millisec)	Total time
IDEAEngine	64	47	171172	390	750	1109	125	78953	78	252688
IDEAEngine	128	0	672	46	109	984	0	1031	343	3313
IDEAEngine	256	0	1063	328	500	1031	0	938	313	4429
DESEngine	64	15	1062	281	359	1000	0	953	485	4219
DESedeEngine	192	62	1015	250	328	1047	0	641	31	3566
AESEngine	128	0	797	32	78	672	0	656	62	2425
AESEngine	192	15	625	79	46	750	0	609	47	2363
AESEngine	256	0	672	47	78	688	0	657	78	2476

Mean values of 5 Test Cases - Request Message Size 1KB - RSA 1024 bit Key Exchange - RSA Signature

AlgorithmName	KeySize	NISCT (In millisec)	EISCT (In millisec)	SEISCT (In millisec)	DEISCT (In millisec)	DDEISCT (In millisec)	NRSCT (In millisec)	ERSCT (In millisec)	SERSCT (In millisec)	Total time
IDEAEngine	128	3	747	106	140	800	0	731	166	2822
IDEAEngine	256	6	791	156	212	856	0	747	156	3182
DESEngine	64	9	850	150	172	744	0	806	160	2955
DESedeEngine	192	16	753	103	141	741	0	644	66	2654
AESEngine	128	3	804	56	91	732	0	806	103	2722
AESEngine	192	9	735	156	143	769	6	703	106	2820
AESEngine	256	28	797	94	84	669	0	644	87	2659
		NISCT	EISCT	SEISCT	DEISCT	DDEISCT	NRSCT	ERSCT	SERSCT	
		T_cc	T_reqec	T_reqs	T_reqd	T_reqdc	T_pocess	T_resec	T_ress	

References[1]

[1]. S. Srirama, M. Jarke, and W. Prinz. Mobile Web Service Provisioning. In *Int. Conf. on Internet and Web Applications and Services (ICIW06).* IEEE Computer Society, February 2006.

[2]. Srirama, S., Jarke, M., Prinz, W., Pendyala, K., "Security Aware Mobile Web Service Provisioning", In Shoniregan C. A. and Logvynovskiy A. (Eds.), Proceedings of the International Conference for Internet Technology and Secured Transactions, ICITST'06, Sep 2006, London, UK, ISBN 0-9546628-2-2, e-Centre for Infonomics, pp. 48-56.

[3]. HTTP, Hypertext Transfer Protocol version 1.1, IETF RFC 2616, http://www.ietf.org/rfc/rfc2616.txt

[4]. kSOAP, a open source SOAP implementation for kVM, http://ksoap.enhydra.org/

[5]. WS-Security, http://www.oasis-open.org/specs/#wssv1.0

[6]. "J2ME Web Services Specification", JSR 172 from Java community process

[7]. S. Srirama, "Concept, implementation and performance testing of a mobile Web Service provider for Smart Phones", Master Thesis, RWTH Aachen University, Jun. 2004

[8]. Werner, C., Buschmann, C. and Fischer, S. Compressing: SOAP Messages by using Differential Encoding. IEEE International Conference on Web Services, July 2004.

[9]. WBXML, Wireless Application Protocol Forum, Ltd. Binary XML Content Format Specification. WAP Forum, 2001.

[10]. Mark Jones and Paul Krill, InfoWorld: JavaOne: JavaFirst brings Web Services to mobile devices, http://www.javaworld.com/javaworld/jw-06-2003/jw-0612-idgns-mobile.html

[11]. SAML V2.0, Security Assertion Markup Language, http://www.oasis-open.org/committees/download.php/13786/sstc-saml-tech-overview-2.0-draft-07-diff.pdf

[12]. Jzlib, http://www.jcraft.com/jzlib/

[1] Note that all the Website links mentioned in this literature were accessible as of date **27-Sep-2006**.

[13]. SOAP, Simple Object Access Protocol, version 1.1, http://www.w3.org/TR/SOAP

[14]. WSDL, Web Services Description Language, version 1.1, http://www.w3.org/TR/wsdl

[15]. UDDI, The Universal Description, Discovery and Integration, http://www.uddi.org/

[16]. Frier, A., Karlton, P., Kocher, P., eds. (November 1996). Netscape Communications Corporation "The SSL 3.0 1175 Protocol,"

[17]. Wason, Thomas, eds. "Liberty ID-FF Architecture Overview," Version 1.2, Liberty Alliance 1147 Project (12 November 2003). http://www.projectliberty.org/specs

[18]. Wireless SOAP: Optimizations for Mobile Wireless Web Services - Apte, Deutsch, Jain (2005), http://www2005.org/cdrom/docs/p1178.pdf

[19]. World Wide Web Consortium, “XML-binary Optimized Packaging (XOP)”, Aug. 2004, http://www.w3.org/TR/2005/REC-xop10-20050125/

[20]. World Wide Web Consortium, “SOAP Message Transmission Optimization Mechanism (MTOM)”, Nov. 2004, http://www.w3.org/TR/2005/REC-soap12-mtom-20050125/

[21]. P. Sandoz, S. Pericas-Geertsen, K. Kawaguchi, M. Hadley, and E Pelegri-Llopart, “Fast Web Services”, Aug. 2003, http://java.sun.com/developer/technicalArticles/WebServices/fastWS/

[22]. UMTS, Universal Mobile Telecommunications System, http://www.iec.org/online/tutorials/umts/

[23]. EDGE, Enhanced Data Rates for GSM Evolution, http://www.ericsson.com/products/white_papers_pdf/edge_wp_technical.pdf

[24]. Cryptographic technologies, http://www.rsasecurity.com/rsalabs/node.asp?id=2212

[25]. W3C, XML Encryption Syntax and Processing, http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/Overview.html

[26]. W3C, XML-Signature Syntax and Processing, http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/Overview.html

[27]. Bouncy Castle lightweight cryptography API. http://www.bouncycastle.org/documentation.html

[28]. "Java support in SonyEricsson mobile phones P800 and P802", Jan. 2003 Developer guidelines from SonyEricsson Mobile CommunicationsAB, www.SonyEricssonMobile.com

[29]. Trusted Computing Group, https://www.trustedcomputinggroup.org/groups/network/

[30]. HHFR, http://grids.ucs.indiana.edu/ptliupages/publications/HHFR_ohsangy.pdf

[31]. Web Services Security: SOAP Message Security 1.0, http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

[32]. Web Services Security: Username Token Profile 1.0, http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf

[33]. Web Services Security: X.509 Certificate Token Profile 1.0, http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf

[34]. Web Services Security: SAML Token Profile, http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.0.pdf

[35]. OMA Web Services Enabler (OWSER): Core Specifications, http://www.openmobilealliance.org/release_program/owser_v1_1.html

[36]. Open Mobile Alliance, http://www.openmobilealliance.org/about_OMA/index.html

[37]. Apache WSS4J, http://ws.apache.org/wss4j/index.html

[38]. Web Services Interoperability Organization, http://www.ws-i.org/

[39]. Web Services Activity, http://www.w3.org/2002/ws/

[40]. S. Srirama, "Concept, implementation and performance testing of a mobile Web Service provider for Smart Phones", MasterThesis, RWTH Aachen University, Jun.2004

[41]. MSDN: Building Secure Web Services, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/THCMCh12.asp

[42]. OWASP Guide to Building Secure Web Applications and Web Services, http://searchappsecurity.techtarget.com/originalContent/0,289142,sid92_gci1157209,00.html

[43]. The ID-FF 1.2 Java Toolkit Overview, http://www.sourceid.org/projects/id-ff_1_2_java_toolkit

[44]. de Jode, Martin: Programming Java 2 Micro Edition on Symbian OS, A Developer's Guide to MIDP 2.0, Apress, 2004

[45]. Sun Microsystems: *Java™ 2 Platform, Micro Edition, The Java™ platform for consumer and embedded devices*, http://java.sun.com/j2me/docs/j2me_cdc.pdf

[46]. Knudsen, Jonathan: *Wireless Java™: Developing with Java™ 2, Micro Edtion*, Apress, 2001.

[47]. MIDlet Basics, https://www6.software.ibm.com/developerworks/education/wi-kxml/section3.html (registration (currently free) required to access the site)

[48]. J2ME Wireless Toolkit User's Guide, http://java.sun.com/j2me/docs/wtk2.2/docs/UserGuide-html/

[49]. Bouncy Castle Crypto package API Specifications, http://www.bouncycastle.org/specifications.html

[50]. JavaWorld : Access Web Services from Wireless Devices, http://www.javaworld.com/javaworld/jw-08-2002/jw-0823-wireless-p2.html

[51]. JSR 172: J2ME™ Web Services Specification, http://jcp.org/en/jsr/detail?id=172

[52]. TRIPLEDES, *Triple Digital Encryption Standard*, http://www.rsasecurity.com/rsalabs/node.asp?id=2231

[53]. AES, *Advanced Encryption Standard*, http://www.rsasecurity.com/rsalabs/node.asp?id=2234

[54]. Ronald Rivest, Adi Shamir and Leonard Adleman, *RSA*, http://www.rsasecurity.com/rsalabs/node.asp?id=2214

[55]. DSS, *Digital Signature Standard*, http://www.rsasecurity.com/rsalabs/node.asp?id=2239

[56]. GPRS, *General Packet Radio Service*, http://www.gsmworld.com/technology/gprs/index.shtml

[57]. 4G Press, *World's First 2.5Gbps Packet Transmission in 4G Field Experiment*, http://www.4g.co.uk/PR2006/2056.htm

[58]. Srirama S., Jarke M. and Prinz W. (2006), 'Mobile Host: a feasibility analysis of mobile web service provisioning', *4th International Workshop on Ubiquitous Mobile Information and Collaboration Systems (UMICS 2006)*, a CAiSE'06 workshop, Springer LNCS

[59]. OWASP Web Services, http://www.owasp.org/index.php/Web_Services

[60]. Stadlober, Stefan:, An Evaluation of Security Threats and Countermeasures in Distributed RFID Infrastructures (Master's Thesis) http://www.iicm.tugraz.at/thesis/sstadlober.pdf

[61]. WonderCrypt: What is PKI?, http://www.wondercrypt.com/pkifaq.htm

[62]. Trusted Computing Group, https://www.trustedcomputinggroup.org/home

[63]. B. Clifford Neuman and Theodore Ts'o. Kerberos: An Authentication Service for Computer Networks, IEEE Communications, 32(9):33-38. September 1994.

[64]. "Internet X.509 Public Key Infrastructure Certificate and CRL Profile". R. Housley, W. Ford, W. Polk,D. Solo, ftp://ftp.isi.edu/in-notes/rfc2459.txt

[65]. SSL Tutorial, http://www.franz.com/support/tutorials/ssl-tutorial.htm

[66]. IETF working group on Public-Key Infrastructure (X.509) (pkix) http://www.ietf.org/html.charters/pkix-charter.html