# An Investigation into Interoperable End-to-end Mobile Web Services Security

Submitted in fulfilment
of the requirements of the degree
Master of Science
of Rhodes University

Thamsanqa Moyo

11th March 2008

## Abstract

The capacity to engage in web services transactions on smartphones is growing as these devices become increasingly powerful and sophisticated. This capacity for mobile web services is being realised through mobile applications that consume web services hosted on larger computing devices. This thesis investigates the effect that end-to-end web services security has on the interoperability between mobile web services requesters and traditional web services providers.

SOAP web services are the preferred web services approach for this investigation. Although WS-Security is recognised as demanding on mobile hardware and network resources, the selection of appropriate WS-Security mechanisms lessens this burden. An attempt to implement such mechanisms on smartphones is carried out via an experiment.

Smartphones are selected as the mobile device type used in the experiment. The experiment is conducted on the Java Micro Edition (Java ME) and the .NET Compact Framework (.NET CF) smartphone platforms. The experiment shows that the implementation of interoperable, end-to-end, mobile web services security on both platforms is reliant on third-party libraries. This reliance on third-party libraries results in poor developer support and exposes developers to the complexity of cryptography. The experiment also shows that there are no standard message size optimisation libraries available for both platforms. The implementation carried out on the .NET CF is also shown to rely on the underlying operating system.

It is concluded that standard WS-Security APIs must be provided on smartphone platforms to avoid the problems of poor developer support and the additional complexity of cryptography. It is recommended that these APIs include a message optimisation technique. It is further recommended that WS-Security APIs be completely operating system independent when they are implemented in managed code.

This thesis contributes by: providing a snapshot of mobile web services security; identifying the smartphone platform state of readiness for end-to-end secure web services; and providing a set of recommendations that may improve this state of readiness. These contributions are of increasing importance as mobile web services evolve from a simple point-to-point environment to the more complex enterprise environment.

**Acknowledgements**

The author would like to thank his supervisors Barry Irwin and Madeleine Wright for the time, effort and advice they contributed during the course of the research conducted for this thesis. He acknowledges the Rhodes Computer Science Department and Centre of Excellence who provided him with all the resources required to carry out the research. Thanks are also due to Lindsey Berry, Emma Drury, Rebecca Jearey,Thulani Nxasana and James Short for proof-reading chapters of this thesis.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Introduction

The field of web services has expanded to include the mobile computing paradigm [Hirsch et al., 2006]. Handheld mobile devices participate in web services transactions through mobile web services. Mobile web services are typically mobile applications that consume traditional web services. The term traditional is used in this thesis to refer to web services hosted on larger computing devices connected by fixed line networks, for example servers connected by a wired local area network (LAN).

Interoperability is a foundational feature of web services [Wright, 2005]. A web services transaction is dependent on interoperability because interoperability ensures that web services entities are able to communicate with each other. The Web Services-Interoperability Organization (WS-I) has produced the *Basic Security Profile Version 1.0* document to guide the securing of traditional web services so that secured web services messages allow for interoperability [McIntosh et al., 2007]. The need for such a document demonstrates that securing web services may impact on web services interoperability.

This thesis investigates the effect that the application of end-to-end, web services security has on the interoperability between mobile web services and traditional web services. It establishes whether current mobile web services and security technology allow mobile web services to participate in end-to-end, secured web services transactions. This thesis focuses on end-to-end security because it is relevant to web services as explained in the following section.

## 1.2 The Relevance of End-to-end Security

The "end-to-end argument" by Saltzer et al. [1984] is presented formally as:

> The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible. (Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement.).

This argument, when applied to the function of web services security, dictates that the web services endpoints should assume responsibility for securing web services messages for the entirety of a transaction. The endpoints should not delegate responsibility for securing the transaction to underlying network protocols.

Web services endpoints consist of either a web services provider or a web services requester. A web services provider is an entity that is responsible for implementing and exposing a web service and a web services requester consumes the provider's service [Booth et al., 2004]. The end-to-end argument is relevant to web services because web services messages may traverse network connections and other entities that are out of the control of web services endpoints [Hirsch et al., 2006]. End-to-end security guarantees that the web services endpoints do not lose control over the security of web services messages during a web services transaction. Chapter 3 provides further analysis of web services security and selects an appropriate end-to-end web services security standard for the investigation carried out in this thesis. The numerous combinations available for the realisation of mobile web services requires some limitation of scope as explained in the following section.

## 1.3 Scope

A multitude of web services approaches exist, for example Asynchronous Javascript and XML (AJAX) [Gehtland et al., 2006], Representational State Transfer (REST) [Fielding, 2000], and SOAP [Gudgin et al., 2007] web services. Diversity is also encountered when the types of handheld mobile devices are considered, for example personal digital assistants (PDA), pagers, mobile phones and smartphones. The convergence of mobile devices and web services results in the availability of multiple mobile web services approaches. This thesis limits its investigation to

SOAP web services on smartphones. The reasons for this limitation of scope are briefly described in the following subsections.

### 1.3.1 SOAP

SOAP messages are extensible because SOAP is as an XML based protocol [Gudgin et al., 2007]. This extensibility allows for SOAP messages to be processed by intermediary nodes lying along the path of the web services endpoints.

End-to-end security is sufficient but not necessary for other web services approaches that utilise point-to-point network links and do not consider intermediary processing nodes [Kearney, 2005]. Examples of such approaches include AJAX and REST. Network transport level security for these web services approaches is sufficient and preferred given its lower performance overhead. However, SOAP processing at intermediary nodes makes point-to-point security inadequate to secure SOAP web services because this security lasts only from an endpoint to its closest intermediary. End-to-end security is the only sufficient security mechanism for all SOAP web services scenarios. SOAP is selected in this thesis because it allows a distinction to be drawn between point-to-point and end-to-end security. The selection of SOAP is also advantageous because significant work on end-to-end web services security is being carried out with SOAP, for example the OASIS WS-Security standard [Nadalin et al., 2006a]. Chapter 2 provides the background to web services and a more detailed motivation for the selection of SOAP.

### 1.3.2 Smartphones

This thesis limits its investigation to handheld mobile devices commonly referred to as smartphones [Fox and Box, 2004]. These devices may be loosely described as those that combine the features of a PDA and a mobile phone. Smartphones are selected as the mobile web services application hosting device type because of their relevance to m-commerce. Mobile web services may be seen as a subset of m-commerce [Hirsch et al., 2006]. Chang and Chen [2005] advocate that the smartphone is the most technically appropriate handheld mobile device for use in m-commerce.

The writer finds that other compelling reasons to select the smartphone for this research are found when the smartphone is considered within the context of the global development of m-commerce. Firstly, developed countries are in the phase of further developing the m-commerce market from its “infancy” to one that resembles the larger e-commerce market [Takahashi, 2006]. Smartphones provide the technology required to overcome the challenges highlighted by the Or-

ganisation for Economic Co-operation and Development (OECD ) in meeting this goal [Takahashi, 2006]. The specifics of these challenges are out of the scope of this thesis but it is noted here that the OECD consider the smartphone as the device that can meet these challenges.

Secondly, Africa continues to present the fastest growing mobile market [Africa Research Bulletin, 2007]. Scott et al. [2004] suggest that smartphones will provide African people with an access point to data services. Considering that some African countries have ten times as many mobile subscribers as fixed line subscribers, it is highly probable that the smartphone may become Africa's main access point to data services [Cheneau-Loquay, 2007].

The technical ability of the smartphone coupled with its potential to be the main device driving m-commerce development, informs the decision to select it as the mobile device type discussed in this thesis. Chapter 4 describes the issues surrounding the realisation of mobile web services on smartphones and chapter 5 introduces the smartphone platforms used in this thesis. The realisation of mobile web services running on these platforms relies heavily on their ability to interoperate with traditional web services.

## 1.4 Interoperability and Mobile Web Services

The author considers interoperability of critical importance within the mobile web services context for two reasons: firstly, the primary realisation of mobile web services is in the form of a web services requester [Shu Fang Rui, 2006]. A lack of interoperability between mobile web services and traditional web services renders web service consumption by mobile applications impossible. The primary realisation of mobile web services hinges on their interoperability with traditional web services.

Secondly, while m-commerce is a subset of e-commerce, one differentiating factor between them is the device-dependent nature of m-commerce and the device-independent nature of e-commerce [Takahashi, 2006]. The device interoperability provided by web services mitigates the drawback of device-dependent service provision. Mobile web services ease m-commerce service provision by utilising device independent-interfaces.

The provision of end-to-end SOAP security leverages the extensibility of SOAP messages and requires the modification of the messages themselves [Nadalin et al., 2006a]. Some of these modifications may lead to interoperability problems if they are not supported by all web services entities. The mobile web services dependence on interoperability means that it is possible for the application of end-to-end security on traditional web services to hinder the realisation of mobile web services. The analysis of this possible obstacle to the development of the mobile

web services field is the central issue of the investigation carried out in this thesis. The following research questions and goals were developed to aid the investigation.

## 1.5 Research Questions and Goals

The following research questions are answered in this thesis:

1. Does the deployment of interoperable, end-to-end, web services security on a traditional web services provider hinder a mobile web services requester from participating in an end-to-end, secured web services transaction with the provider?

2. Which smartphone aspects require improvement so that mobile web services may better interoperate with end-to-end, secured traditional web services providers?

These questions are answered within the context of the following specific research goals:

1. to determine appropriate mechanisms that achieve interoperable, end-to-end, mobile web services security;

2. to ascertain whether interoperable, end-to-end, mobile web services security can be feasibly implemented using current mobile software technology;

3. to identify the areas that require improvement for the provision of interoperable, end-to-end, mobile web services security;

The research goals mentioned in this section form the direction that this thesis takes. A more detailed discussion of these goals appears in the following sub-sections.

### 1.5.1 Goal 1: Determination of an Appropriate Mechanism

The selection of SOAP, mentioned earlier, allows a distinction to be drawn between point-to-point and end-to-end web services security. This distinction allows mechanisms that provide true end-to-end web services security to be identified. Chapter 3 identifies such mechanisms that are also interoperable.

The appropriateness of the mechanisms identified in chapter 3 is determined by the degree to which they are suitable for smartphone platforms. This suitability depends on how the mechanisms operate in respect to the constraints faced by smartphones. Chapter 4 presents a set of appropriate mechanisms that provide interoperable, end-to-end, mobile, web services security. This set is implemented on smartphone platforms to meet the next research goal.

### 1.5.2 Goal 2: Ascertation of Implementation Feasibility

The smartphone capability to support interoperable, end-to-end, mobile web services security is ascertained by implementing the set of mechanisms detailed in chapter 4. This implementation highlights shortcomings that prevent smartphones from hosting mobile web services that interoperate with end-to-end, secured traditional providers. Chapter 5 describes the experiment used to implement the set of mechanisms. The platform shortcomings that hinder this implementation are used to the meet the next research goal.

### 1.5.3 Goal 3: Provision of Recommendations

The shortcomings gleaned from the experiment allow for the identification of smartphone platform aspects that require improvement. Suggestions to improve smartphone platforms such that they can be used to implement interoperable, end-to-end, mobile web services security are provided in chapter 6. These recommendations are issued after smartphone platform shortcomings are discussed in the same chapter. The nature of the content of this thesis is detailed in the following section.

## 1.6 Thesis Content

The research reported in this thesis occurs within the dynamic fields of web services and mobile computing. Work and interest in end-to-end mobile web services security is increasing as evidenced by the work of Kangasharju [2007] and Narayana et al. [2007], discovered by the author at the time of writing. However, there are instances where books and journal articles do not provide a deep enough analysis of the issues or present outdated material. To this end, technical specifications, technical articles and web material is referenced in this thesis along with books and journal articles. Technical specifications and articles such as the *NIST Guide to Secure Web Services* provide a deeper analysis of some issues than books or journals. These technical citation sources also provide details on the latest standards because some are written by the standardisation bodies themselves. Web references are used when no alternative citation source is found and the reference explains an issue that the thesis needs to address.

The mobile computing field is also rapidly changing and this may result in the discussions presented in this thesis referring to a technology that has been superseded. There is a six month period between the completion of some aspects of the experiment and the conclusion of writing. An example of technology that has changed during the course of writing is the Windows Mobile

operating system. Devices running Windows Mobile 6 became available after the part of the experiment using a Windows Mobile device was concluded. This thesis reports work conducted on a Windows Mobile 5 device because Windows Mobile 5 was the latest edition running on Windows Mobile devices available at time of experimentation. The recommendations and conclusions presented in this thesis have been kept general to address this rapid change. This allows the recommendations and conclusions to be applied to the latest technology, if it should suffer from the same shortcomings identified in this thesis.

A note is also made here about the spelling of terms within this thesis. The thesis adopts British English for most of its content. However, some terms and quotations are spelt with American English, for example the Message Transmission *Optimization* Mechanism. The spelling used in these terms and quotations is left unchanged. The content described in this section is laid out as described in the following section.

## 1.7 Thesis Layout

Chapter 2 introduces web services concepts used in the rest of the thesis. Web services terms are defined in this chapter and two dominant web services approaches are compared; SOAP and REST. The selection of SOAP is further motivated in this chapter.

Chapter 3 introduces the security needs of SOAP web services. Strategies that may be used to meet these needs are discussed and those that provide end-to-end security as opposed to point-to-point security are identified. These strategies are discussed within the context of mobile web services in chapter 4.

Chapter 4 discusses the realisation of mobile web services on smartphones. Alternative strategies for this realisation are discussed and a mobile web services architecture appropriate for the experiment is selected. The mechanisms identified in chapter 3 are discussed within the context of the constraints that make the mobile computing paradigm unique. Web services security mechanisms that are considered to be the most appropriate, when these constraints are considered, are selected and used in the following chapter.

Chapter 5 details an experiment that involves implementing the mechanisms discussed in chapter 4 on two smartphone platforms: the .NET Compact Framework and the Java Micro Edition [Fox and Box, 2004] [Sun Microsystems, 2007b]. These two platforms are motivated in chapter 5 as the most appropriate for the experiment's objectives. The shortcomings encountered in both platforms when implementing interoperable, end-to-end, mobile web services security are highlighted in this chapter and form the discussion of the following chapter.

Chapter 6 considers the platform-specific shortcomings presented in chapter 5 as lessons that are applicable to smartphone platforms in general. These lessons are discussed and they are used as the basis on which the recommendations presented in the chapter are made. These recommendations suggest improvements for smartphone platforms so that they may host mobile web services that participate in end-to-end secured web services transactions.

Chapter 7 concludes this thesis by discussing the answers to the research questions posed in this introduction chapter. The contribution made by the research is outlined and possible future work is suggested.

# Chapter 2

# Web Services

## 2.1 Introduction

This chapter introduces web services and concepts required for the discussion of web services security in the next chapter. The aim of this chapter is to survey the web services domain for an appropriate web services approach that facilitates a discussion on end-to-end web services security.

This chapter identifies common web services characteristics. These characteristics define the terms utilised in the survey of two popular web services approaches: SOAP and REST. These two approaches are compared and SOAP is selected as an appropriate approach to examine end-to-end web services security.

## 2.2 The Web Services Landscape

The web services domain is characterised by a wide scope comprising a multitude of implementation approaches. This is illustrated by the overview by Adams et al. which defines three broad categories of web services: Enterprise web services, Internet web services and XML web services.

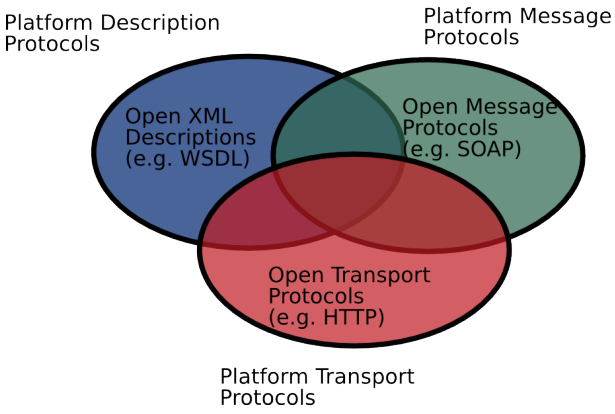


Figure 2.1: Venn diagram of web services platform protocols, adapted from Adams et al..

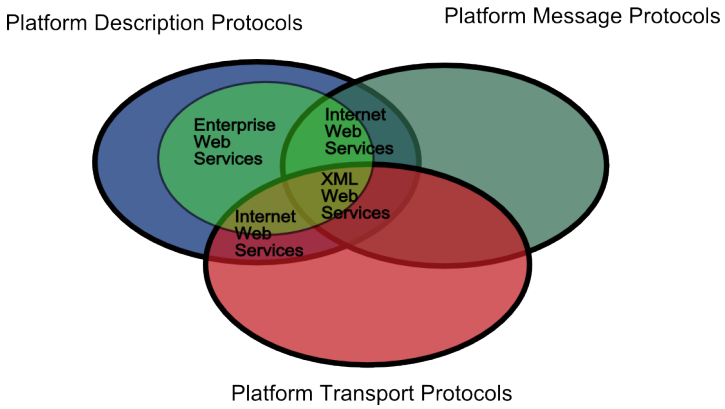


Figure 2.2: Venn diagram of web services categories, adapted from Adams et al..

Figure 2.1 shows three classes of protocols that may be used to implement these web services. The description protocols are used for communicating the functionality exposed by a web service and the requirements for invoking this functionality. This description determines the content of messages formatted according to messaging protocols. These messages are sent over a network using transport protocols.

Figure 2.2 maps the three web services categories onto the three protocols [Adams et al.]. Enterprise web services are those that utilise open protocols for description and messaging but may use proprietary transport protocols. Internet web services use either open messaging or open transport protocols and XML web services use open protocols for all categories.

This thesis does not continue to use the category definitions provided by Adams et al.. Their categorisation of web services however, does provide an insight into the numerous combinations possible in web services implementation. For example, a SOAP web services implementation may utilise SOAP for messaging, the Web Services Description Language [Chinnici et al., 2007] for description and the Simple Mail Transfer Protocol (SMTP) [Postel, 1982] for transport. A RESTful implementation [Richardson and Ruby, 2007] may utilise the Extensible Markup Language (XML) [Bray et al., 2006] for messaging, the Web Application Description Language [Hadley, 2006] for description and the Hypertext Transfer Protocol (HTTP) [Fielding et al., 1999] for transport.

Despite the lack of a standard definition for web services and the diversity of their implementations, this section provides a generic overview of web services and their nature. This overview lays the groundwork for subsequent discussions of web services implementations. The overview begins by identifying a general definition for web services.

### 2.2.1 What are Web Services?

Pandey [2006b] states that there is no universally accepted definition for web services. Adams et al. confirm this viewpoint when arguing that the existence of a plethora of web services definitions, some of which are "contradictory", poses a challenge to defining them.

This section identifies a general definition of a web service. This definition is used to discuss the common aspects of web services. A concept that is often mentioned in web services definitions is the architectural notion of a service oriented architecture.

#### 2.2.1.1 Architectural Considerations

An examination of the relationship between web services and a Service Oriented Architecture (SOA) is carried out because of the strong link asserted between the two terms [Wright, 2005]. The danger of this close association is in assuming that the terms and their security are analogous. This sub-section shows that the difference between these two terms is such that the concept of SOA is inadequate in providing a general web services definition. Papazaglou and van-den Heuvel [2004] define a service as:

> ... self-describing, platform agnostic computational elements...which provide some application logic in a distributed computing system.

This definition implies that services are an atomic element within a service oriented architecture which they continue to define as:

> ... a logical way of designing a software system to provide services to either end user applications or other services in a network through published and discoverable interfaces [Papazaglou and van-den Heuvel, 2004].

Wright [2005] asserts that the main distinction between a SOA and web services is that a "SOA is an architecture, while web services are a technology". This distinction is corroborated by web services definitions that suggest that web services are implementations of a SOA. This is evident in the following web services definition:

> Web services refer to a specific set of technologies that can be used to implement a SOA [Hirsch et al., 2006].

Further definitions provided by Ramesh Nagappan et al. [2003], Muschamp [2004] and Kearney [2005] also suggest this architecture-implementation relationship. This relationship suggests an implementation of web services must possess qualities espoused by the architectural specification of a SOA. Such qualities include the properties outlined by the World Wide Web Consortium (W3C) Web Services Architecture (WSA) Working Group [Booth et al., 2004]: message orientation; description orientation; granularity; network orientation; and platform neutrality. All these properties are encompassed by the definitions of a service and SOA provided by Papazaglou and van-den Heuvel [2004] .

These definitions imply a tight relationship between web services and a SOA, where a SOA provides the blueprint and the bricks are laid using web services. However, the W3C WSA Working Group [Booth et al., 2004] advocates a looser relationship by stating that the implementation of web services does not necessarily translate into an implementation of a SOA. This rationale extends from the W3C WSA Working Group's stance that a SOA is not a web services architecture but one of many views that a stakeholder may hold of a web services architecture. Some deployments of web services may implement a SOA but it is possible for web services to be employed in a distributed system that does not meet the properties of a SOA. The Resource Oriented Architecture introduced by Richardson and Ruby [2007] provides an example of web services deployed according to a different architecture. This architecture is appropriate for the implementation of RESTful web services.

Therefore, two conclusions, considered within the context of this thesis, may be drawn about the relationship between a SOA and web services: Firstly web services and SOA should be treated as separate concepts. Secure web services do not automatically mean a secure SOA because web services implementations may not always meet the properties of a SOA. Secondly in the case where web services do meet the properties of a SOA, secure web services cannot be

extrapolated to mean a secure SOA. The W3C WSA Working group [Booth et al., 2004] state that depending on infrastructure and requirements, other technologies such as CORBA and COM may be more appropriate for the implementation of a SOA. However, these middleware technologies have differences from web services that demand a different security approach from that taken in web services. For example web services rely on XML but CORBA and COM implementations need not use XML in their interactions [Wright, 2005]. To this end, XML security mechanisms may be successfully deployed to secure web services but not other SOA related technologies, such as CORBA and COM. This thesis is limited to discussing web services security and an analysis of SOA security is out of scope.

Given the elimination of the concept of a SOA, a more general definition is provided in the following section.

#### 2.2.1.2 General Definition

Wright [2005] also attempts to identify a general web services definition. In determining this definition, Wright [2005] identifies XML as a common denominator amongst web services and selects a definition that places an emphasis on XML. The author agrees with the emphasis on XML since web services cannot exist without XML, as demonstrated by the category definitions of Adams et al.. The following definition from Adams et al. is utilised in this thesis:

> Web services are software components that are developed using specific technologies from three primary technology categories: An XML-based description format (for example, WSDL); An application messaging protocol (for example, SOAP); A collection or transport protocol (for example, HTTP).

This definition alludes to three extra common properties of web services that a definition based on XML alone does not not fully encompass. Firstly communication between web services role-players is facilitated by a network linking the role-players. Secondly, web services need an XML means to describe themselves such that they may be consumed. Thirdly, a message protocol is required to facilitate communication amongst web services entities.

The category lines may be blurred as the categories mentioned in the definition may use the same protocol. An example is the RESTful approach that may utilise HTTP as both its messaging and transport protocol [Richardson and Ruby, 2007]. This possibility of category overlap is not problematic to the definition, as long as an overlapping protocol is capable of performing the functionality required within each category. Web services entities in respect to this general definition are discussed next.

### 2.2.2 Web Services Entities

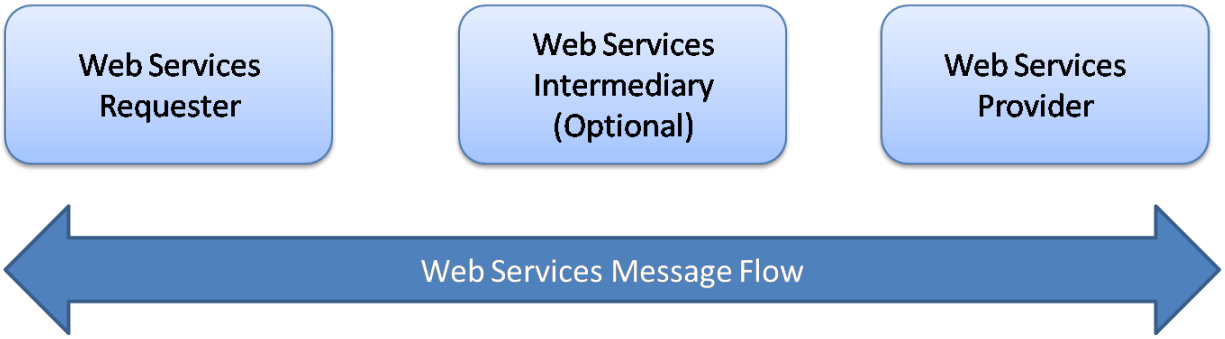


Figure 2.3: Three components of web services [Singhal et al., 2007].

Figure 2.3 is composed of three types of entities that may be defined in a web services transaction: a web services provider; a web services requester and a web services intermediary [Singhal et al., 2007]. As with the web services definition, these names should not be taken as standard. The entities they represent however, are central to the research of this thesis. These entities are named from the terms used by the Open Mobile Alliance (OMA) [Open Mobile Alliance, 2006], the US National Institute of Standards and Technology (NIST) [Singhal et al., 2007] and the W3C WSA Working Group.

The W3C WSA Working Group draws a distinction between agents and entities when defining a requester and a provider[Booth et al., 2004]. An agent is an implementation of a web service and the entity is a "person or organisation" that owns the agent [Booth et al., 2004]. No such distinction is necessary in this thesis as it is concerned with agents, that is, the implementation of web services. The terms entity, provider and requester are considered to refer to the implementation of a web service. This same approach is taken by OMA and NIST in their definitions [Open Mobile Alliance, 2006, Singhal et al., 2007].

Figure 2.3 illustrates the relationship between the roles. By definition all web services implementations require a web services provider and a web services requester since a network connects at least two endpoints. A web services message will be created at one endpoint and be consumed at another.

Not all web services approaches support a web services intermediary but this entity is mentioned because it necessitates end-to-end web services security. Figure 2.3 suggests that the intermediary lies along the flow of web services messages between the two endpoints. An intermediary introduces a situation warranting end-to-end security, especially if it is untrusted. This scenario is revisited in greater detail in the following chapter.

#### 2.2.2.1 Web Services Provider

A web services provider (provider) is a web services endpoint that is responsible for hosting and exposing the web service [Booth et al., 2004, Open Mobile Alliance, 2006]. In a client-server architecture the provider would be called the server. This analogy is limited because depending on the nature of a web services transaction, an endpoint may act as both a client and a server [Singhal et al., 2007]. The analogy however, illustrates that a provider hosts a resource that is consumed by a web services requester.

#### 2.2.2.2 Web Services Requester

The web services requester (requester) is a web services endpoint that consumes the web service offered by the provider [Booth et al., 2004, Open Mobile Alliance, 2006]. Some authors may refer to this as a web services consumer [Hirsch et al., 2006]. Although both a provider and requester may initiate a transaction, in most practical applications the requester initiates the transaction [Booth et al., 2004]. The messages that flow between a requester and provider may be modified by a web services intermediary.

#### 2.2.2.3 Web Services Intermediary

The web services intermediary has been encountered by the author in the SOAP web services implementation approach only. The W3C Web Services Architecture group [Booth et al., 2004] describe a web services intermediary as:

> ... a Web service whose main role is to transform messages in a value-added way.

This definition implies that the intermediary lies along the path of a message as it flows between the requester and provider. Multiple intermediaries may exist along this path and they are invoked in a sequential manner [Singhal et al., 2007]. An example of an intermediary is an XML gateway that applies security enhancements to a web services message at it travels between endpoints. This example demonstrates that the meaning of the message is never changed when value is added to it by an intermediary[Booth et al., 2004].

### 2.2.3 Web Services Transaction Process

A web services transaction may be broken into three stages: messaging, description and discovery [Hirsch et al., 2006]. Figure 2.4 illustrates the relationship between these components. The


Figure 2.4: Three stages of a web services transaction [Hirsch et al., 2006].

discovery stage permits a requester to identify the provider hosting the web service it wishes to consume. This discovery exposes the description of the web service or points to a location where the description resides.

The description phase defines the interface exposed by a web service [Booth et al., 2004]. The requester uses this description to establish what messaging and transport protocols to use and how to use them to communicate with the provider.

Messages are created and formatted according to these descriptions at the messaging stage [Hirsch et al., 2006]. The messages are placed on the network utilising the transport protocols mentioned by the description stage.

Figure 2.4 may be seen to suggest a tightly coupled relationship. This is not the case because messages can be created without the need for a description phase if the application developer already knows the nature of the web service. A discovery phase is also not required if the application developer has prior knowledge of the location of the service. However, the stages provide a guide as to how a web services transaction may be approached when no prior knowledge of the location or nature of a web service is held.

Two of the stages mentioned in this section can be directly mapped onto the general web services definition provided in section 2.2.1.2 because the definition makes explicit mention of

messaging and description protocols. The messaging protocols mentioned in the definition are utilised in the messaging stage and the description protocols are utilised in the description stage. The discovery stage is not dealt with in the definition but this omission has no bearing on the research presented.

#### 2.2.3.1 Discovery

The web services discovery component provides a requester with the location information of a provider [Hirsch et al., 2006]. Hirsch et al. [2006] suggest the following four ways this may be done:

1. The provision of information about the service location via "out-of-bands" mechanisms such as email.
2. The extraction of location information from a previously known source such as a web site.
3. The use of a repository such as UDDI.
4. Identity-based discovery services that provide location information based on the identity of the requester.

The first two require human intervention and fall out of scope because this thesis is concerned with web services implementations and not the human users of web services. The use of a repository and discovery service allows for machine driven discovery [Hirsch et al., 2006]. However, the adoption of machine driven discovery such as UDDI is hindered by issues such as trust [Wright, 2005].

This thesis is concerned with end-to-end security and that of web services messages in particular. The discovery stage provides information concerning the location of a web service and has no bearing on the securing of web services messages themselves. Therefore, this thesis does not deal with the discovery stage in its investigation. A brief mention of this aspect is given here for the sake of providing a complete overview of web services. It is assumed that if discovery takes place it is via one of the out-of-band methods such at the first two discovery methods listed by Hirsch et al. [2006].

### 2.2.4 Summary

Web services may be generically described as a product of four components: XML; a description protocol; a messaging protocol and an application protocol. These four components constitute

the web services that are the focus of further discussions presented in this thesis.

The web services discussed in the thesis have two common entities: a provider and requester. Optional intermediaries may also exist along the message path between these common entities, although this is dependent on the web services implementation. The entities engage in transactions through the stages of discovery, description and messaging. The discovery stage is irrelevant to this thesis as it does not affect the end-to-end security of web services messages.

This section's discussion is limited to a generic analysis; specifics are avoided. The following sections discuss two common implementations of web services: SOAP and REST. A simple calculator example is utilised to demonstrate the aspects of these implementations.

## 2.3 Web Services Demonstration

A simple calculator web service is provided to demonstrate the messaging and description stages of both SOAP and REST web services. The requester sends two integers to the provider and calls either the add or subtract functionality offered at the provider. This example is similar to that provided by the Web Services Interoperability Technologies (WSIT) tutorial [Sun Microsystems, 2007h] and it is built following the instructions in the tutorial. SOAP web services are the first approach that this demonstration is applied to.

## 2.4 SOAP Web Services

SOAP web services are a popular W3C driven implementation of web services. Messaging is carried out by the SOAP protocol and description by the Web Services Description Language (WSDL) [Wright, 2005]. Each of these components are briefly examined in the context of the calculator web service example.

### 2.4.1 Messaging

Messaging in the SOAP web services approach is carried out by the SOAP protocol [Booth et al., 2004]. The W3C provides the following definition for the latest version of the SOAP protocol at the time of writing:

> SOAP Version 1.2 (SOAP) is a lightweight protocol intended for exchanging structured information in a decentralised, distributed environment. It uses XML technolo-

> gies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols [Gudgin et al., 2007].

SOAP's XML foundation allows it to leverage the XML advantages of extensibility and platform neutrality. Extensions may be applied to the SOAP protocol to add extra functionality such as security. SOAP messages may be transported by a variety of networking protocols such as SMTP and HTTP.

The envelope messaging structure defined by SOAP carries two types of information: the "application payload" and "control information" [Gudgin et al., 2007]. The application payload resides in the SOAP body and contains information intended for an endpoint [Gudgin et al., 2007]. Examples of this payload include information required to invoke a web service and a result returned by the web service to the requester. Control information resides in the SOAP header. Examples of control information include message security and routing information.

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header>

       <To xmlns="http://www.w3.org/2005/08/addressing">


          http://www.w3.org/2005/08/addressing/anonymous
       </To>
    </S:Header>
    <S:Body>
       <add xmlns="http://tham.org/">
          <i>1</i>
          <j>1</j>
       </add>
    </S:Body>
</S:Envelope>
```

The `<S:Header>` element in the snippet contains all the headers of this SOAP message. In this example, one optional header is shown: `<To xmlns="http://www.w3.org/2005/08/`

addressing">. This header is defined according the Web Services Addressing specification [Gudgin et al., 2006] and provides the Uniform Resource Identifier (URI) [Mealling and Denenberg, 2002] of a target endpoint. The value of http://www.w3.org/2005/08/addressing/anonymous indicates that this information is not available.

The <S:Body> element contains the application payload required to invoke the web service. In this case it calls the add operation of the web service and passes two integers to it.

The web services requester requires prior knowledge of the requirements of a web service to create an appropriate SOAP message. This knowledge is provided by the WSDL.

## 2.4.2 Description

The Web Services Description Language (WSDL) is a W3C recommendation for a language that describes a web service [Chinnici et al., 2007]. This description allows for the machine-to-machine interaction mentioned in the definition provided in section 2.2.1.2. A machine can read in a WSDL document and automatically generate the described SOAP message.

The language provides both an abstract and concrete description of a web service. The abstract description specifies the content of SOAP messages. The concrete description details how SOAP messages should be transported between the requester and provider. WSDL version 1.1 is used in this thesis [Christensen et al., 2001]. Although WSDL 2.0 [Chinnici et al., 2007] is the latest WSDL version, it was approved at the end of the testing reported in this thesis.

WSDL version 1.1 describes a WSDL document as "simply a set of definitions" [Christensen et al., 2001]. The WSDL definitions of the calculator web service example are discussed next.

### 2.4.2.1 Abstract Definition

Abstract definitions in WSDL 1.1 are composed of three sections: types; message and portType [Christensen et al., 2001]. The <types> element describes the data types utilised in SOAP messages. The <message> elements provide a "logical" structural description of these messages and the <portType> element describes the web service operations targeted by the messages [Christensen et al., 2001].

The following snippet illustrates the <types> element of the calculator web service:

```
<types>
    <xsd:schema>
        <xsd:import
```

```
            namespace="http://tham.org/"
            schemaLocation="http://146.231.121.204:8080
            /CalculatorWSService/CalculatorWS?xsd=1">
        </xsd:import>
    </xsd:schema>
</types>
```

WSDL allows for the referencing of eternal XML schemata that define the types utilised in a WSDL document [Christensen et al., 2001]. The calculator web service's WSDL document contains a reference to an external schema and the schema definition of an add type is shown in the following snippet:

```
<xs:complexType name="add">
    <xs:sequence>
        <xs:element name="i" type="xs:int"/>
        <xs:element name="j" type="xs:int"/>
    </xs:sequence>
</xs:complexType>
```

This add complex type is composed of two integers and is used to define the add element in the same schema:

```
<xs:element name="add" type="tns:add"/>
```

This element is utilised in the <message> element of the WSDL document:

```
<message name="add">
    <part name="parameters" element="tns:add"/>
</message>
```

The <part> element in a <message> element describes the contents of a SOAP message [Christensen et al., 2001] . The <message> element describes the parameters required to invoke the add operation of the calculator web service. These parameters are of the add type and this type is defined in the external schema as two integers.

The final part of abstract definition is the <portType> element:

```
<portType name="CalculatorWS">
    <operation name="add">
       <input message="tns:add"/>
       <output message=......
</operation>
...
</portType>
```

The <portType> element defines web service operations as a set of input and output messages [Christensen et al., 2001]. In the case of the calculator web service example, the add message described in the <message> element is used to define the message sent to invoke the add operation of the service. The constructed SOAP message will carry two integers that are parameters to the add operation. The concrete description part of WSDL is discussed next.

#### 2.4.2.2 Concrete Description

The concrete description of WSDL describes how messages specified in the abstract description should be transported [Nadalin et al., 2006a]. WSDL utilises three components to provide a concrete description: the <binding> , <port> and <service> elements [Christensen et al., 2001]. The <binding> element specifies the protocol that the messages will be sent with. The <port> element provides a URI of the SOAP message destination and the <service> element groups ports that have a relationship to each other.

The following snippet shows part of the <binding> element:

```
<binding name="CalculatorWSPortBinding"
type="tns:CalculatorWS">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http"
...>
....
</binding>
```

Each binding refers to a specific <portType> element and this snippet shows the binding for the portType CalculatorWS. This is the portType shown in section 2.4.2.1. The binding also specifies that HTTP is used in sending the messages. The URI of the target endpoint is held in the <port> element inside the <service> element as illustrated in the following snippet:

```
<service name="CalculatorWSService">
    <port name="CalculatorWSPort"
    binding="tns:CalculatorWSPortBinding">

       <soap:address location="http://146.231.121.204:8080/
       CalculatorWSService/CalculatorWS"/>
    </port>
</service>
```

This snippet shows that the message is to be sent to the provider at the `http://146.231.121.204:8080/CalculatorWSService/CalculatorWS` URI. Since there is only one portType and consequently one binding, only one port appears in the `<services>` element. A full version of the WSDL file used to generate the SOAP message shown in section 2.4.1 without a SOAP header, is provided in Appendix A.

### 2.4.3 Summary

SOAP web services facilitate machine-to-machine interactions through the WSDL which specifies the SOAP messages passed during machine communication. However, SOAP web services are criticised for the level of complexity they introduce [Richardson and Ruby, 2007]. Such criticism has lead to the development of alternative web services approaches, for example the RESTful approach.

## 2.5 RESTful Web Services

The acronym REST is first discussed by Fielding [2000] in his PhD dissertation on HTTP . The chief mantra behind REST web services is to put the "web back into web services" [Richardson and Ruby, 2007]. This is realised by utilising HTTP to facilitate web services transactions.

Richardson and Ruby [2007] state that REST, as discussed by Fielding [2000], is a set of architectural principles that may be adhered to in various implementations. The term RESTful denotes whether an implementation adheres to these principles. According to Richardson and Ruby [2007], two principles must be evident in a RESTful implementation. Firstly, instructions concerning what a provider must do with the data it receives must be expressed using standard

HTTP methods such as GET, POST and DELETE. Secondly, all data sent to the provider must be placed in the HTTP URI.

It is possible to use SOAP in a RESTful manner but a major difference between the SOAP approach and the RESTful approach is that the SOAP approach does not need HTTP for its transport. The web services definition provided in 2.2.1.2 makes explicit mention of HTTP but does not limit SOAP web services to utilising HTTP. Other protocols such as SMTP may also be utilised in an implementation of SOAP web services [Booth et al., 2004].

Another difference becomes apparent when web services messages are considered as documents wrapped in "envelopes" [Richardson and Ruby, 2007]. SOAP web services wrap their documents in a SOAP envelope structure and REST web services wrap their documents inside an HTTP envelope. Instead of utilising the HTTP URI to send data to a provider and the HTTP method to instruct the provider on how to process the data, all this information is bundled inside the SOAP message as shown in section 2.4.1. This further reinforces the earlier assertion that the RESTful approach is fundamentally tied into HTTP and the SOAP approach is not.

Therefore, a RESTful web service cannot exist without HTTP in much the same way as a SOAP web service cannot exist without SOAP. A more detailed description of the RESTful approach according to the messaging and description stages is provided in this section.

### 2.5.1 Messaging

Messaging in the RESTful web services approach is provided through a combination of HTTP and XML. A packet capture of a RESTful calculator web service transaction is shown in the following snippet:

```
GET /RestfulCalc/resources/add?i=1&j=5 HTTP/1.1
Host: 146.231.121.204:8080
Connection: Keep-Alive

HTTP/1.1 200 OK
X-Powered-By: Servlet/2.5 Server: Sun Java System Application
Server 9.1
Content-Type: application/xml
Content-Length: 18
Date: Thu, 27 Sep 2007 09:06:44 GMT
<result>6</result>
```

The first part of the packet capture shows a RESTful request. The HTTP GET method implies that a result must be returned by the provider. The URI details that the web service's add functionality is to be called and the two HTTP parameters in the URI specify two integers that need to be added.

The second part of the packet capture demonstrates the response sent by the provider to the requester. HTTP codes are utilised to provide feedback on the success of the operation [Richardson and Ruby, 2007]. In the example shown, the operation is successful and the response is accompanied by the HTTP success code 200 [Fielding et al., 1999]. The enveloping nature of HTTP is also demonstrated by the XML representation of the result that is carried in the response. In a World Wide Web (WWW) [Jacobs and Walsh, 2004] transaction this would typically be a Hypertext Markup Language (HTML) document [Raggett et al., 2005]. However, RESTful web services utilise XML documents to facilitate communication between the requester and provider.

The simplicity of REST may render the description phase defunct as the URI of the web service may describe the web service. For example, the URI utilised in the snippet may be used to accurately extrapolate that the web service it references provides a calculator with addition functionality. A description stage may ,however be required when the URI does not accurately represent the nature of a RESTful web service.

### 2.5.2 Description

The general web services definition provided in section 2.2.1.2 requires that a web service has a description provided in an XML-based format. Most current RESTful web services are described by a combination of text documents and XML schemata [Hadley, 2006]. This combination may not be fully machine processable. The Web Application Description Language (WADL) [Hadley, 2006] provides a machine-processable answer to the description of RESTful web services.

Richardson and Ruby [2007] define WADL more clearly than its technical specification by Hadley [2006] as:

> ... an XML vocabulary for expressing the behaviour of HTTP resources.

Such an XML resource may be a RESTful web service. The following snippet demonstrates what part of a WADL document would look like for the RESTful calculator example. The full WADL file is provided in Appendix B. The WADL specification is used as a guideline in generating the WADL file :

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<application xmlns="http://research.sun.com/wadl/2006/10">
```

```
<resources base="http://146.231.121.204:8080
/RestfulCalc/resources//resources">
   <resource path="add">
      <method name="GET">
        <request>

          <param name="i" type="xsd:string"
          style="query" required="true"/>
          <param name="i" type="xsd:string"
          style="query" required="true"/>
        </request>
        <response>
        <representation mediaType="application/xml"
        element="result" />"
   </response>
      </method>
      </resource>
   ...
</resources>
</application>
```

The <resources> element is utilised for grouping a set of related resources together [Hadley, 2006]. The WADL snippet shows only the resource corresponding to the add functionality.

Within each <resource> element a <method> element is named by the corresponding HTTP method that will be used by the requester to invoke the resource. In the example, the <method> element is named GET and this refers to the HTTP GET method [Fielding et al., 1999].

The <method> element contains the two sub-elements: <request> and <response>. The <request> element shown in the example specifies the parameters to be added to the URI and the <response> element specifies how the result will be returned.

WADL is still in its infancy and not widely deployed [Richardson and Ruby, 2007]. This is further evidenced by the author's experiences in developing the two calculator web service examples. The generation of a WSDL document for the SOAP calculator web service is trivial because a tool such as Java2WSDL automatically generates a WSDL document. The generation

of a WADL document is completed by hand because the Glassfish Project [Sun Microsystems, 2007d] includes only a WADL-to-Java converter and not a Java-to-WADL converter. However, WADL still has the potential to play a role similar to that played by WSDL in SOAP web services.

### 2.5.3 Summary

The simplicity of RESTful web services is evident when messaging and description protocols are considered. WADL and WSDL describe web services offering the same functionality, yet the RESTful approach provides a much simpler and streamlined description. One of the major advantages of the SOAP web services approach is the automatic generation of messages via the machine processable WSDL [Richardson and Ruby, 2007]. WADL provides an opportunity for the realisation of the same advantage with RESTful web services.

A comparison of SOAP and REST is provided next for the purposes of selecting an appropriate approach for the security analysis in this thesis.

## 2.6 Comparison of SOAP and REST

There is a long standing debate that is engaged in answering which of the SOAP and RESTful web services is the better approach? For example zur Muehlen et al. [2005] attempt to answer which of the two provides the best approach for web services choreography and Richardson and Ruby [2007] evaluate the SOAP approach according to the same rules of analysis they apply to RESTful web services.

An overall conclusion on which approach is better is out of scope, although the comparisons in the preceding examples agree that both approaches have some degree of merit to them. Richardson and Ruby [2007] suggest that SOAP web services have more of a value proposition in distributed systems that are characterised by multiple role-players interacting across different domains.

The standpoint that each approach has its merits is shared by the author. A decision is made on which approach is the more appropriate for an analysis of end-to-end mobile web services security. This decision does not imply that one approach is more secure than another, but that one approach is more suitable for the investigation carried out in this thesis.

The two criteria used in this selection are: the interoperability of the potential security employed by each approach; and the applicability of the end-to-end argument to these mechanisms. An argument for the selection of SOAP is presented based on the assertion that SOAP allows for

better interoperability than RESTful web services when security is applied. In addition, the end-to-end security argument makes more sense when considered in the context of SOAP security.

### 2.6.1 Secure Interoperability

The argument that SOAP web services allow for more interoperability than RESTful web services seems indefensible when SOAP's interoperability criticisms are considered. For example Richardson and Ruby [2007] argue that RESTful web services allow for better interoperability than SOAP web services. However, when security mechanisms are applied, RESTful web services fall short in that there is no standard mechanism to convey security requirements between the requester and provider. This poses an interoperability problem as the two endpoints may not have the means to agree on a security configuration required for interaction. Further analysis of this situation is as follows:

RESTful web services messages primarily rely on HTTP security. This often has its form in the Transport Layer Security protocol (TLS) [Dierks and Rescorla, 2006]. Since RESTful response messages carry XML documents, the XML Encryption and XML Signature standards may also be applied to these XML documents. TLS and/or XML Encryption may be utilised to achieve message confidentiality and message integrity may be achieved by XML Signature. The same applies to SOAP web services as TLS, XML Encryption and XML Signature may be used to secure SOAP messages.

Security in RESTful web services is limited by the absence of a standard mechanism to share security requirements. Tyagi [2006] states among other reasons, that SOAP web services must be used instead of RESTful web services when:

> The architecture must address complex nonfunctional requirements. Many web services specifications address such requirements and establish a common vocabulary for them. Examples include Transactions, Security, Addressing, Trust, Coordination, and so on....With the RESTful approach, developers must build this plumbing into the application layer themselves.

WADL is the most standard mechanism for REST, encountered by the author, that allows a provider and requester to share requirements. However, WADL does not provide for the exchange of security requirements. The communication of security requirements through WSDL is enabled by the WS-Policy specification [Vedamuthu et al., 2007a].

It may be argued that TLS provides adequate security for RESTful web services and that the requirements for a TLS session do not need to be explicitly expressed before instantiating the

session. This is true when the inclusion of the string "https://" at the beginning of the URI is enough information to indicate the requirement for TLS. However, this argument limits RESTful web services to a dependence on TLS in a manner that SOAP web services are not. For example a RESTful web service provider has no standard manner to express that it utilises the XML Encryption standard or how it utilises this standard. Instead the developer of the provider must attempt to express this information with an out-of-bands mechanism. However, a SOAP web service may express all of its security requirements through WSDL [Nadalin et al., 2006a].

It is the author's opinion that it does not matter that TLS may render the XML Encryption standard redundant for RESTful web services. In the event of a provider employing a security mechanism other than TLS, the challenge still remains in expressing the security requirements in a standard manner.

### 2.6.2 Application of the End-to-End Argument

The end-to-end argument introduced in section 1.2 may be applied in a point-to-point scenario, that is, a web services scenario composed of a requester and provider only. This point-to-point scenario is apparent in a RESTful web services interaction as no intermediaries are supported by the RESTful approach. An HTTP session is established directly between two endpoints. However, an analysis of end-to-end security with such an approach is meaningless as there is no differentiation between point-to-point security and end-to-end security. The same mechanisms that achieve point-to-point security will achieve end-to-end security.

SOAP web services support the intermediary role through the extensibility of the SOAP header. This header carries security configuration information and the extensibility of the SOAP protocol allows various parts of a SOAP message to be secured from one endpoint to another, with the existence of intermediaries.

The existence of intermediaries also means that point-to-point security is inadequate as such security only applies from one endpoint to its closest intermediary. Therefore, SOAP web services allow a distinction to be drawn between point-to-point and end-to-end security. The analysis presented in this thesis is better aided when considered in the context of SOAP web services as end-to-end security mechanisms can be more clearly identified. A deeper discussion of end-to-end security mechanisms is provided in the following chapter.

### 2.6.3 Summary

SOAP web services are the preferred approach for the investigation carried out in this thesis. The utilisation of RESTful web services requires the definition of a standard security requirements communication mechanism. Further difficulty is encountered by the lack of a distinction between end-to-end security and point-to-point security within RESTful web services. These obstacles are overcome with SOAP web services.

Given that RESTful web services are eliminated from the discussion of this thesis, the general web services definition provided in 2.2.1.2 is no longer necessary. A more tightly constrained web services definition based on SOAP is adopted for this thesis.

The W3C Web Services Architecture document [Booth et al., 2004] provides a web services definition as follows:

> A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialisation in conjunction with other Web-related standards [Booth et al., 2004].

This definition meets the requirements of this thesis as it mentions the WSDL and SOAP messages which have been motivated in this section to best facilitate the discussion on web services security presented.

## 2.7 Summary

RESTful and SOAP web services are two pervasive web services implementations that cannot be ignored due to their popularity. Justifications must be given when selecting one approach as opposed to the other. This is particularly important in this thesis as the selection of SOAP may be non-intuitive.

RESTful web services are simpler and generate less network traffic than SOAP web services. This makes them an ideal implementation for mobile web services which operate in a resource constrained environment. Despite SOAP web services being less desirable than RESTful web services on the mobile device, this thesis is not concerned about the implementation of mobile web services per se. Rather it is concerned with an investigation into interoperable end-to-end security for mobile web services. This investigation is better performed with SOAP web services.

When an end-to-end web services security analysis is needed, SOAP web services become a more appropriate choice for the analysis than RESTful web services. The maturity of SOAP web services partly manifests in a standard mechanism for conveying security requirements. The support for intermediaries by the SOAP web services approach allows for a meaningful discussion of end-to-end security.

SOAP is therefore selected for the value it will add to the security analysis of this thesis. However, this does not mean that SOAP web services are more secure than RESTful web services because TLS may adequately meet the message security needs of RESTful web services. The following chapter analyses end-to-end security for web services.

# Chapter 3

# Web Services Security

## 3.1 Introduction

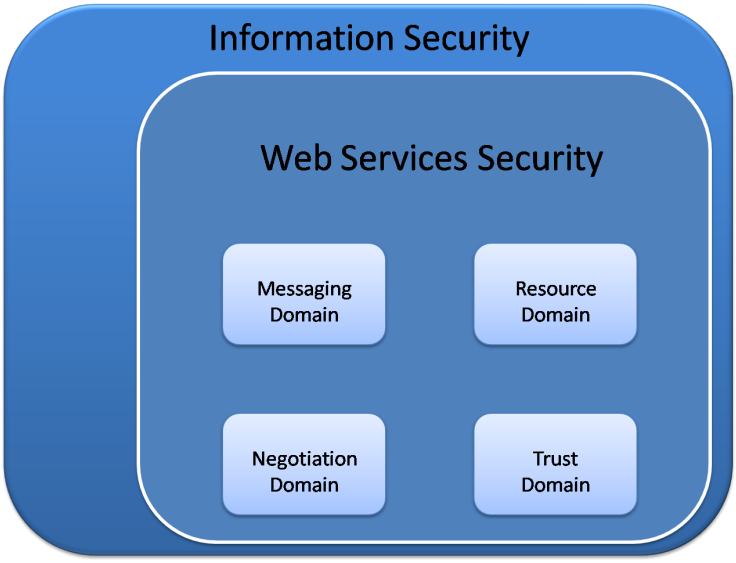


Figure 3.1: Web services security domains.

This chapter discusses SOAP web services security. The *NIST Guide to Secure Web Services* document provides the most recent comprehensive discussion of SOAP web services security found, in literature, by the author [Singhal et al., 2007]. The analysis from this recommendation document is utilised, in the first part of this chapter, to set the foundation for the identification of end-to-end web services security mechanisms in latter parts of the chapter.

Web services security may be considered a subset of information security as shown in Figure 3.1 [Singhal et al., 2007]. Four types of web services security may be identified and these types

are split into the domains shown in Figure 3.1. The messaging domain is identified as the type of web services security with which this thesis is concerned. The challenges that must be met in providing this type of security and the threats that hinder its realisation are reviewed. These challenges and threats allow security approaches to be analysed and their suitability for providing end-to-end message security to be determined. Message level security is found to be the most suitable approach and its implementation, directed by the WS-Security specification [Nadalin et al., 2006a], is discussed.

Some general information-security concepts are applicable to web services security because web services security is a subset of information security [Singhal et al., 2007]. These concepts are introduced within this chapter.

## 3.2 Web Services Security within Information Security

The field of information security is very wide as evidenced by the security concepts defined in the NIST Glossary of Information Security Terms document [Kissel, 2006]. NIST identify the following general security concepts that relate to web services: authentication; authorisation; integrity; non-repudiation; confidentiality and privacy [Singhal et al., 2007]. These concepts are defined by NIST as follows [Singhal et al., 2007]:

**Confidentiality:** Preserving authorised restrictions on information access and disclosure, including means for protecting personal privacy and proprietary information.

**Integrity:** The property that data has not been altered in an unauthorised manner while in storage, during processing, or in transit.

**Authentication:** Verifying the identity of a user, process, or device, often as a prerequisite to allowing access to resources in an information system.

**Authorization:** The permission to use a computer resource, granted, directly or indirectly, by an application or system owner.

**Non-repudiation:** Assurance that the sender of information is provided with proof of delivery and the recipient is provided with proof of the sender's identity, so neither can later deny having processed the information.

**Privacy:** Restricting access to subscriber or relying party information in accordance with Federal law and organization policy.

NIST assume these concepts are relevant to web services because they are relevant to web applications [Singhal et al., 2007]. This assumption is based on an assertion that web services and web applications share the same underlying architecture, for example HTTP. The previous chapter's analysis attests that such an assertion is not always true because SOAP is transport protocol neutral. Web services need not utilise the same underlying architecture as web applications.

Despite this limited reasoning by NIST, the author is of the opinion that the concepts they identify are nevertheless relevant. These concepts are corroborated by other authors who identify a similar set of concepts when discussing web services security. Kearney et al. [2004b] identify the same concepts, other than privacy, and the discussion on security by Hirsch et al. [2006] also refers to these concepts. The security concepts identified by NIST are accepted here because of the consensus between authors in regards to their relevance.

Figure 3.1 on page 32 shows that web services security may be split into four domains. The general security concepts identified here fall into these domains.

## 3.3 Web Services Security Domains

NIST group different aspects of web services security into the domains of messaging, resource, negotiation and trust [Singhal et al., 2007]. The messaging domain is discussed in detail here because it deals directly with web services messages and the end-to-end argument presented in section 1.2.

The resource, negotiation and trust domains are not considered because they deal with issues that are irrelevant to the process of securing web services messages [Singhal et al., 2007]. For example, the negotiation domain deals with the establishment of agreements between the requester and provider [Singhal et al., 2007]. ebXML is a technology commonly advocated to support such an establishment. It facilitates the establishment of agreements on a different process or channel within which web services transactions occur [Grangard, 2001]. This demonstrates that agreements may dictate what security to apply to a message but their establishment is independent of the process of securing the message. This thesis is concerned with the security agreed upon as opposed to the mechanisms that lead to this agreement. The messaging domain is considered as it is the only domain that deals directly with securing web services messages.

### 3.3.1 Messaging

The messaging aspect of web services security is concerned with securing messages as they travel along a network within a web services environment [Singhal et al., 2007]. Kearney et al. [2004b]

refer to this aspect as communications security because it involves securing the communications channel along which messages flow.

The security concepts related to this area of web services security are confidentiality, integrity and authentication [Singhal et al., 2007]. Threats to a SOAP message's confidentiality and integrity exist because SOAP was not designed with inherent security and SOAP messages may travel over the open Internet. Authentication is necessary because messages from authorised entities need to be distinguished from those of malicious entities [Kearney, 2005]. The challenges and threats specific to the messaging domain are elaborated upon in the following section.

## 3.4 Threats and Challenges

Web services present a security paradigm shift because they challenge traditional security assumptions and practises [Khoo and Zhou, 2004]. One such practise is the deployment of firewalls. Protecting web services with traditional firewalls is inadequate because SOAP messages are often transported by HTTP. Most traditional firewalls do not validate the data payload that passes through the HTTP ports, 80 and 443.

Some arguments have been made for the integration of firewalls into web services security. Cremonini et al. [2003] make a case for the deployment of "Semantic-firewalls" which analyse the content and meaning of SOAP messages. However Cremonini et al. [2003] do not advocate firewalls as a sufficient security mechanism but as one complementary to other web services security mechanisms. The practise of protecting web services with firewalls also requires a shift in security thinking. The challenge of providing security according to the web services paradigm and the threats that may prevent its realisation are discussed in this section.

### 3.4.1 Web Services Security Challenges

The WS-I identify a set of challenges in providing message security and a set of threats to these challenges [Schwarz et al., 2007]. This identification forms a requirements document for their work on specifying interoperable web services security in the Basic Security Profile [McIntosh et al., 2007]. This thesis adopts these same requirements since interoperable web services security is one of its research goals as presented in section 1.5 on page 5.

The WS-I identify the following security challenges [Schwarz et al., 2007]:

- data confidentiality;
- data integrity;

- peer authentication;
- data authentication;
- message uniqueness.

Peer authentication refers to establishing the identity of a web services entity [Schwarz et al., 2007]. This type of authentication is insufficient because the extensibility of SOAP allows intermediaries to insert parts into a SOAP message. It is necessary to identify and authenticate each SOAP message part according to its creator. This is referred to as data authentication [Schwarz et al., 2007].

The second challenge identified by the WS-I is data integrity which is defined as follows by RFC 2828 [Shirey, 2000]:

> The property that data has not been changed, destroyed, or lost in an unauthorised or accidental manner [Shirey, 2000].

The WS-I differentiate this term from its usage in information management by stating that integrity is concerned with detecting message "alteration even when under active attack" [Schwarz et al., 2007]. The challenge of web services message integrity is to ensure that a SOAP message is not changed by accident or by a malicious entity as it travels along the communication path.

Data confidentiality is the third challenge identified by the WS-I and it requires that a SOAP message part is viewed by intended recipients only [Schwarz et al., 2007]. The extensibility of SOAP complicates this challenge because it allows SOAP message parts to have different recipients. Protection needs to be provided against genuine web service transaction entities viewing SOAP message parts not intended for them.

The final challenge identified by the WS-I is that of message uniqueness and they define it as:

> ... the ability to insure that a specific message is not resubmitted for processing [Schwarz et al., 2007].

An attacker may capture a legitimate SOAP message and resend it to a provider. This may have unintended consequences such as billing a client twice for one transaction. Mechanisms that detect the resubmission of messages must be provided to meet this challenge.

### 3.4.2 The Criteria for End-to-End Messaging Security

The web services security challenges of authentication, confidentiality and integrity correspond to the security concepts relevant to the messaging domain and described by NIST [Singhal et al., 2007]. The WS-I introduce one extra challenge not considered by NIST and that is message uniqueness. This addition is complementary to the taxonomy by NIST because it does not contradict any of the concepts they highlight. The challenges of confidentiality , integrity, authentication and message uniqueness are considered as the goals to be achieved by web services messaging security.

These challenges provide criteria for a test for end-to-end messaging security. An end-to-end security approach is considered as one capable of meeting the messaging security challenges end-to-end. To this end the strategies and mechanisms mentioned in the rest of this thesis are tested for this capability. Messaging security threats work against the realisation of these challenges.

### 3.4.3 Messaging Security Threats

The WS-I identify a set of threats that may prevent the realisation of the challenges of confidentiality integrity, authentication and message uniqueness [Schwarz et al., 2007]. These threats are defined as follows [Schwarz et al., 2007] :

**Message Alteration:** The message information is altered by inserting, removing or otherwise modifying information created by the originator of the information and mistaken by the receiver as being the originator's intention.

**Confidentiality:** Information within the message is viewable by unintended and unauthorized participants.

**Falsified Messages:** Fake messages are constructed and sent to a receiver who believes them to have come from a party other than the sender.

**Man in the Middle:** A party poses as the other participant to the real sender and receiver in order to fool both participants.

**Principal Spoofing:** A message is sent which appears to be from another principal.

**Forged claims:** A message is sent in which the security claims are forged in an effort to gain access to otherwise unauthorized information.

**Replay of Message Parts:** A message is sent which includes portions of another message in an effort to gain access to otherwise unauthorized information or to cause the receiver to take some action.

**Replay:** A whole message is resent by an attacker.

**Denial of Service:** (The) attacker does a small amount of work and forces (the) system under attack to do a large amount of work.

| | Data Confidentiality | Data Integrity | Peer Authentication | Data Authentication | Message Uniqueness |
|---|---|---|---|---|---|
| Message Alteration | | * | | | |
| Confidentiality | * | | | | |
| Falsified Messages | | | * | * | |
| Man in the Middle | | | * | * | |
| Principal Spoofing | | | * | * | |
| Forged Claims | | | * | * | |
| Replay of Message Parts | | | * | * | * |
| Replay | | | | | * |
| Denial of Service | | | | | * |

Table 3.1: Mapping of web services message goals and threats by Schwarz et al. [2007]

These threats may be mapped onto the SOAP message security goals identified in section 3.4.1. This mapping is shown in table 3.1 [Schwarz et al., 2007]. The identification of threats to confidentiality and integrity is straightforward as the threats are antithetical to the challenges. Threats to authentication arise from the problem of establishing identity during a web services transaction. The threats to message uniqueness are based on the failure to discriminate between messages.

Table 3.1 may give the incorrect perception that threats may be dealt with in isolation under each challenge. However, some overlap between challenges may be encountered when mitigating certain threats. An example is a variant of the man in the middle attack which relies on “active wiretapping” defined by RFC 2828 [Shirey, 2000] as:

> An attack that intercepts and accesses data and other information contained in a flow in a communication system.

With this attack, an attacker intercepts a message for the sake of eavesdropping. The attacker does not need to assume an identity but only requires a copy of the message stream as the message is transmitted. Meeting the challenge of authentication is inappropriate to mitigate this threat if the challenge of confidentiality is not met.

## 3.5 Web Services Messaging Security Stack

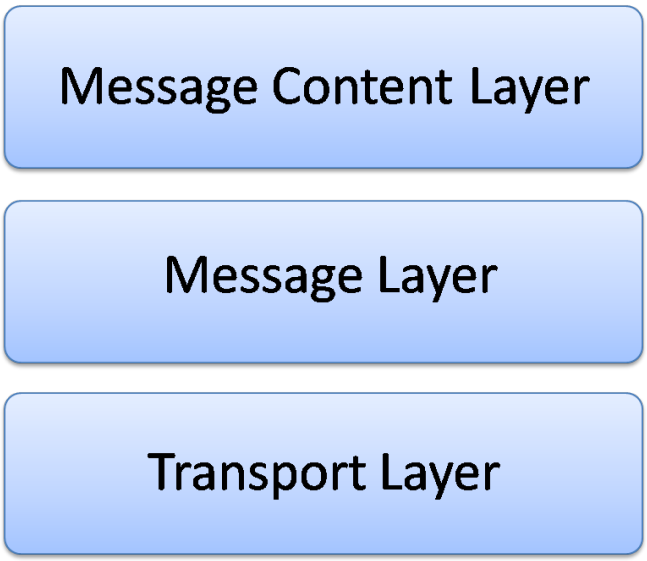


Figure 3.2: Web services security layers [Kearney et al., 2004a].

Layered architectures are a popular mechanism for describing networked systems, for example the layered OSI reference model [Jeckle and Wilde, 2004, Zimmermann, 1988]. The layering and separation of concerns adhered to by layered architectures allows the security functions within a networked system to be more clearly identified.

NIST categorise web services security functions according to the OSI reference model, for example deployment of a dedicated line at the link layer, TLS at the transport layer and XML Encryption at the application layer [Singhal et al., 2007]. Jeckle and Wilde [2004] derive a web services protocol stack with layers that mirror the functionality of the OSI model's layers, for example the Internet protocols that transport SOAP messages are placed at the bottom layer of their stack. This layer corresponds to the physical layer in the OSI model. Jeckle and Wilde [2004]

place SOAP message security at the "transport layer" of their stack because security functionality is found in the transport layer of the OSI model. Geuer-Pollmann and Claessens [2005] position security within a "WS-*" specification stack. This stack layers SOAP web services specifications but it does not adhere to the separation of concerns principle because some layers overlap. However, a distinction between layers containing Internet protocol security and layers containing SOAP security is made.

Despite the different layered web services models, a trend towards the separation of Internet protocol security and SOAP message security is apparent. Figure 3.2 better illustrates this separation through the 3 layers of web services security identified by Kearney et al. [2004a] : transport layer; message layer; and message content layer. Each of these layers are independent of each other as dictated by the separation of concerns principle.

### 3.5.1 Transport layer security


Figure 3.3: Transport layer security when multiple hops exist.

Securing the underlying Internet protocols that carry SOAP messages is done at the transport layer [Kearney, 2005]. This layer may be mapped to the network and transport layers of the OSI stack and examples of security at this layer are TLS and virtual private networks. This type of security encapsulates the higher layers shown in figure 3.2, for example TLS will encrypt an entire message payload regardless of the security deployed at a higher layer.

Although Kearney [2005] states that transport layer security is commonly adopted, its suitability for web services is criticised. NIST and Kearney [2005] affirm that transport level security can only guarantee security between an endpoint and its closest intermediary [Singhal et al., 2007]. If the intermediaries are untrusted, there is no way to guarantee a message's security from one endpoint to another. Figure 3.3 illustrates a scenario where the requester has no way of ensuring that the second TLS connection is ever instantiated.

A related criticism of transport security is the lack of transport independence [Singhal et al.,

2007]. The close coupling of security with transport protocols is problematic because security will be terminated and instantiated with each transport link. This is illustrated in figure 3.3 where two TLS connections are instantiated for one transaction. The termination of the first connection and the instantiation of the second requires security information to be repackaged and transferred from one connection to another [Singhal et al., 2007]. This may result in security vulnerabilities from poor security implementations of this complex transfer. The endpoints have no control over the quality of this protection along the entire message path because they rely on the quality of the security implemented by intermediaries.

Transport layer security also protects the entire message payload and this makes it impossible to protect individual message parts [Khoo and Zhou, 2004]. The protection of individual parts of a message is important when it is undesirable for certain intermediaries to access some message parts.

#### 3.5.1.1 The Transport Layer and End-to-end Security

Transport layer mechanisms such as TLS may provide protection against all the threats mentioned in section 3.4.3. TLS allows for mutual authentication to mitigate the authentication threats and it provides encryption to mitigate the confidentiality threats [Schwarz et al., 2007]. The deployment of message authentication codes (MAC) [Shirey, 2000] by TLS mitigates integrity threats [Dierks and Rescorla, 2006]. MAC will guarantee the message uniqueness of a TLS session only. The replay of a TLS session transporting a SOAP message can be detected but the replay of the SOAP message with different TLS sessions cannot be detected.

However, the previously mentioned transport layer criticisms indicate that the mitigation of the security threats cannot be achieved end-to-end when intermediaries exist. Intermediaries must be relied upon to mitigate threats along the message path. If an intermediary is unauthorised to access an entire message then confidentiality threats are inherent to transport layer security. The threats to data authentication are also inherent in transport layer security as this type of security authenticates an entire message and not individual parts. Transport layer security fails to provide end-to-end web services message security because it does not work at the required granularity. The need for finer-grained end-to-end web services security has led to the development of message layer security.

### 3.5.2 Message Layer Security

Message layer security secures SOAP messages themselves [Kearney, 2005]. XML security mechanisms provide security at this layer because SOAP is based on XML. XML Encryption provides mechanisms for encrypting XML documents and these documents may be signed using XML Signature. Since SOAP uses XML in a specific manner, WS-Security details how to use these XML security standards to secure SOAP messages [Nadalin et al., 2006a].

The criticisms levelled at transport layer security are dealt with by message layer security. Message layer security is not dependent on the communication links between web services entities. To this end criticisms based on transport protocol dependence are defeated . XML Encryption and XML Signature may be used to secure individual XML message parts. This fine granularity overcomes the transport layer criticism that individual message parts cannot be secured according to their intended recipients.

#### 3.5.2.1 The Message Layer and End-to-end Security

The threats in section 3.4.3 may be mitigated end-to-end by message layer security. XML encryption mitigates confidentiality threats and XML Signature mitigates threats to authentication and integrity [Schwarz et al., 2007]. WS-Security defines mechanisms for authentication and mitigates threats to message uniqueness using unique message identifiers and timestamps [Nadalin et al., 2006a]. Although message layer security provides the end-to-end security for web services, the meaning of the security employed remains unestablished. This is the role of the message content layer.

### 3.5.3 Message Content Layer

The message content layer is unique to the analysis by Kearney [2005]. This layer is concerned with the meaning of security mechanisms deployed at lower levels, for example determining whether a digital signature means that the owner of the signing key has signed the message or that an entity with access to the signing key created the message [Kearney, 2005]. Such a consideration is relevant because the plausibility of signing key theft defeats non-repudiation.

This layer deals with security issues whose solutions are currently non technological [Schwarz et al., 2007]. This layer is not considered further as it is an undeveloped open issue that warrants future work of its own.

### 3.5.4 Summary

Both transport and message layer security may mitigate the threats identified in section 3.4.3. Transport layer security may meet web services security challenges in a point-to-point environment only. Message layer security can meet these challenges in an end-to-end manner.

It is possible to combine transport and message level security since they are independent. The *WS-I Security Challenges, Threats and Countermeasures* document [Schwarz et al., 2007] details how to implement such a combination but this is not considered here because end-to-end security cannot be provided by transport layer security. Combining transport layer security with message layer security does not add any value to the provision of end-to-end web services security.

## 3.6 WS-Security

It is the author's experience that WS-Security is the pervasive specification for providing message layer security. SOAP headers are the mechanism for extending SOAP [Gudgin et al., 2007]. Since the SOAP specification leaves security to be implemented as an extension, WS-Security specifies the `<wsse:Security>` header to hold security information [Nadalin et al., 2006a]. A `<wsse:Security>` header carries security information for one recipient only and a `<wsse:Security>` header will be created for each entity that security information is intended.

Relevant parts of a `<wsse:Security>` header are used to illustrate the mechanisms employed by WS-Security. This header is taken from the WS-Security secured version of the calculator web service introduced in section 2.3 on page 18. A full version the header is available in Appendix C.

### 3.6.1 Confidentiality through Encryption

WS-Security specifies how XML Encryption is used to provide SOAP message confidentiality [Nadalin et al., 2006a]. The encryption strategy employed by WS-Security combines symmetric and asymmetric encryption [Ferguson and Schneier, 2003]. Figure 3.4 illustrates this strategy for a scenario where a requester sends a message to a provider. A requester encrypts a symmetric key using the provider's public key. This symmetric key is used to encrypt parts of a message and the encrypted version of the symmetric key is attached to the message. The provider, on receipt of the message, will decrypt the encrypted symmetric key with its private

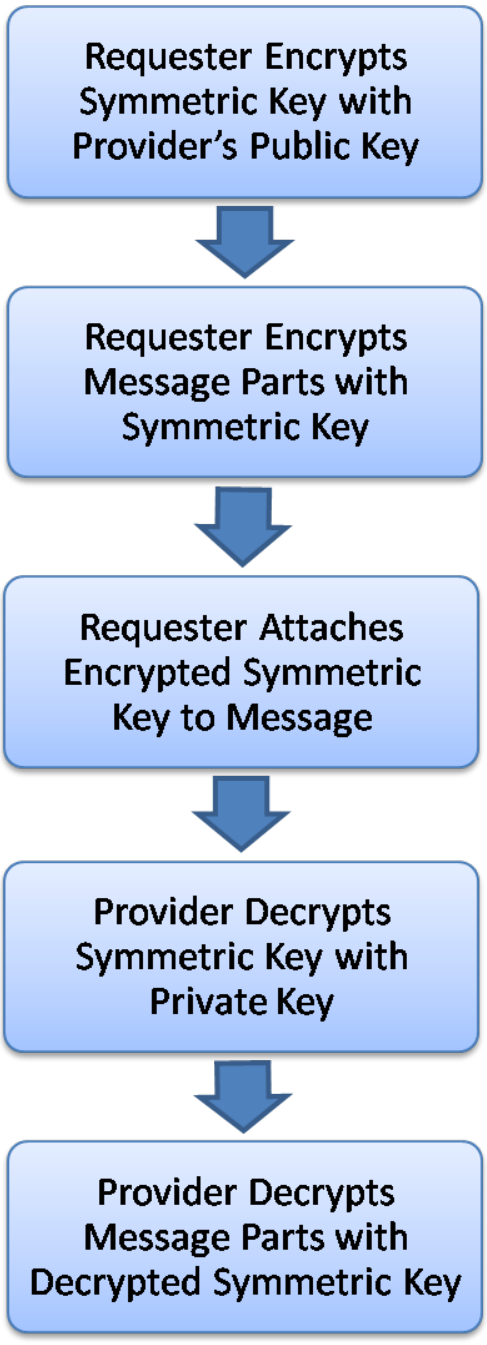


Figure 3.4: WS-Security encryption process [Nadalin et al., 2006a].

key and decrypt the remaining encrypted SOAP message parts with the decrypted symmetric key. The response will be encrypted by the provider and decrypted by the requester with the symmetric key which by then is mutually known.

Four critical elements required for this encryption strategy are the `<ds:KeyInfo>`, `<xenc:EncryptedKey>`, `<xenc:ReferenceList>` and `<xenc:EncryptedData>` elements.

The `<ds:KeyInfo>` element is not specific to the encryption process because it is defined by XML Signature [Eastlake et al., 2005]. This element is useful to XML Encryption because it references keys used in an encryption process. The following snippet shows an example of a `<ds:KeyInfo>` element:

```
<ds:KeyInfo xmlns:xsi...>
    <wsse:SecurityTokenReference>
        <wsse:KeyIdentifier
        ValueType="http://docs.oasis-open.org/wss/
        2004/01/oasis\-200401\-wss\-x509\-token\-profile\-1.0\
        #X509SubjectKeyIdentifier"
        EncodingType...>dVE29ysyFW/iD1la3ddePzM6IWo=
        </wsse:KeyIdentifier>
</wsse:SecurityTokenReference>
</ds:KeyInfo>
```

The `<ds:KeyInfo>` may hold multiple `<wsse:SecurityTokenReference>` elements as defined in WS-Security [Nadalin et al., 2006a]. These elements hold references to security tokens. A security token may be referenced by either a URI , or a key identifier or a reference to where the token is embedded in the SOAP message. The `<wsse:KeyIdentifier>` element represents a subject key identifier and this identifier is used to identify a public key from a specific certificate [Housley et al., 1999] .

The encrypted symmetric key is held inside the `<xenc:EncryptedKey>` element:

```
<xenc:EncryptedKey ...URI=\#5002>
    <xenc:EncryptionMethod
    Algorithm="http://www.w3.org/2001/04/xmlenc\#rsa\
    -oaep\-mgf1p"/>
    ...
```

```
<xenc:CipherData>
   <xenc:CipherValue>ixQT5...nahtIM=
   </xenc:CipherValue>
</xenc:CipherData>
```

The <xenc:EncryptionMethod> method specifies the algorithm utilised in encrypting the symmetric key [Imamura et al., 2005]. The algorithm used in the snippet is RSA with Optimal Asymmetric Encryption Padding (OAEP) [Kaliski and Staddon, 1998]. The <xenc:CipherData> element is mandatory as it holds the encrypted value of the symmetric key.

The parts of the message encrypted with symmetric key are referenced using the <xenc:ReferenceList> element:

```
<xenc:ReferenceList...>
   <xenc:DataReference URI="\#5008">
   </xenc:DataReference>
   ...
</xenc:ReferenceList>
```

The <xenc:ReferenceList> element in the snippet points to the message part referenced by URI #5008. This message part is an <xenc:EncryptedData> element:

```
<S:Body...>
   <xenc:EncryptedData ... Id="5008">
      <xenc:EncryptionMethod
      Algorithm="http://www.w3.org/2001/04
      /xmlenc\#aes128-cbc"/>
   <ds:KeyInfo...>
      <wsse:SecurityTokenReference>
         <wsse:Reference
         ValueType="http://docs.oasis-open.org
         /wss/oasis-wss-soap-message-security-1.1
         \#EncryptedKey" URI="\#5002"/>
      </wsse:SecurityTokenReference>
   </ds:KeyInfo>
   <xenc:CipherData>
```

```
        <xenc:CipherValue>FTB...W7E=</xenc:CipherValue>
    </xenc:CipherData>
    </xenc:EncryptedData>
...
</S:Body>
```

The <xenc:EncryptedData> element also contains an <xenc:EncryptionMethod> element [Imamura et al., 2005] which specifies that the Advanced Encryption Standard (AES) algorithm ["NIST", 2001] is utilised to encrypt the SOAP body shown in section 2.4.1. The <ds:KeyInfo> element inside the <xenc:EncryptedData> points to the encrypted symmetric key through a direct reference represented by the <wsse:Reference> element. A mandatory <xenc:CipherData> holds the cipher text resulting from the encryption of the SOAP body.

#### 3.6.1.1 Summary

WS-Security utilises a combination of elements specified by the XML Encryption and those specified by itself, to provide data confidentiality. This combination is necessary because XML Encryption was not designed specifically for SOAP but for XML. WS-Security fills in the areas where the XML Encryption may be inappropriate for SOAP. XML Signature is the second W3C recommendation that WS-Security utilises.

### 3.6.2 Integrity through Signatures

WS-Security specifies how to use XML Signature to provide digital signatures that achieve end-to-end message integrity for SOAP messages [Nadalin et al., 2006a]. Any part of the SOAP header or body may be signed. Multiple signatures may also exist for a single message part. This is useful when intermediaries exist because the receiving endpoint can establish a "chain of trust" on the signed message part by validating each of its signatures [Nadalin et al., 2006a]. If an alteration is detected, it can be traced to the interaction between the last intermediary whose signature was valid and the first intermediary whose signature was invalid.

The Digital Signature Standard ["NIST", 2000] describes a digital signature generation process beginning with the creation of a message digest of the data to be signed. The signature creation process ends with the application of a digital signature algorithm's sign operation to the

message digest. This strategy is a performance optimisation as signing message digests is faster than the signing of actual data [Ferguson and Schneier, 2003]. However, signing XML with XML Signature requires an extra preliminary step in the process, XML canonicalisation [Boyer, 2001].

#### 3.6.2.1 XML canonicalisation

The extensible nature of XML allows two XML documents to have a different structure but the same logical meaning [Boyer, 2001]. This feature is problematic when signing XML because hashing algorithms will generate a unique message digest for a each unique physical representation of data. It is possible to have two logically identical documents whose message digests and consequent signatures do not correspond. Canonicalisation algorithms transform logically equivalent documents into physically equivalent documents, such that they will result in the same digest value when hashed.

The inclusive [Boyer, 2001] and exclusive [Boyer et al., 2002] canonicalisation algorithms are supported by WS-Security [Nadalin et al., 2006a]. However, the exclusive canonicalisation algorithm is the suggested algorithm to be used as it allows a message signed part to be more easily inserted into multiple documents. This easier transplant is enabled by the exclusive canonicalisation feature of copying all needed XML namespaces into the canonicalised XML. Such a transplant is needed when a token is signed and used to authenticate multiple SOAP messages. An exclusive canonicalised message part is robust to namespace definition changes that may be encountered when inserting the part into multiple documents with varying namespace declarations. Signature verification will fail if the namespaces in the inserted message part and the rest of the document are inconsistent. While the implementation in this thesis does not need to insert an XML part into multiple SOAP messages, exclusive canonicalisation is used as it is "strongly recommended" by WS-Security [Nadalin et al., 2006a]. Canonicalisation from this point onwards refers exclusive canonicalisation only.

#### 3.6.2.2 XML Signature structure

A digital signature is represented by the `<ds:Signature>` element [Eastlake et al., 2005]. This element comprises of the following critical elements : `<ds:SignedInfo>`; `<ds:SignatureValue>`; and `<ds:KeyInfo>`. The `<ds:KeyInfo>` element holds signing key information using the same structure described in section 3.6.1.

The `<ds:SignedInfo>` element contains references to the message parts the signature signs [Eastlake et al., 2005]. Each reference holds the message digest of the part it refers. The

<ds:SignedInfo> also contains details of the hashing and canonicalisation algorithm used to create the message digest. These references are encapsulated in <ds:Reference> elements within the <SignedInfo element>:

```
<ds:SignedInfo>
    <ds:CanonicalizationMethodAlgorithm =
    "http://www.w3.org/2001/10/xml-exc-c14n#">

       <exc14n:InclusiveNamespaces PrefixList="wsse S"/>
    </ds:CanonicalizationMethod>
    <ds:SignatureMethod Algorithm="http://www.w3.org
    /2000/09/xmldsig#hmac-sha1"/>
    <ds:Reference URI="#5007">
       <ds:Transforms>
          <ds:Transform Algorithm=
          "http://www.w3.org/2001/10/xml-exc-c14n#">
          <exc14n:InclusiveNamespaces PrefixList="S"/>
          </ds:Transform>
       </ds:Transforms>
       <ds:DigestMethod Algorithm="http://www.w3.org
       /2000/09/xmldsig#sha1"/>
       <ds:DigestValue>bpX...Jzc=</ds:DigestValue>
    </ds:Reference>
    ...
</SignedInfo>
```

The <ds:Reference> element in the snippet references the SOAP body. This message part may undergo transformations before its digest is created. The details of these transformations are held in the optional <ds:Transforms> element and the transformation detailed in the snippet is exclusive canonicalisation. The <ds:DigestMethod> element specifies the hashing algorithm used and the <ds:DigestValue> element holds the resulting message digest obtained from applying the hashing algorithm to the transformed version of the message part. The snippet shows that the Secure Hash Algorithm-1 gloss(SHA-1) [Eastlake and Jones, 2001] is used to generate the message digest over the canonicalised SOAP body.

Multiple <ds:Reference> elements may exist in a <ds:SignedInfo> element since multiple message parts may be signed with the same signing key [Eastlake et al., 2005]. Once the <ds:SignedInfo> element has been created it is canonicalised and its canonicalised form is signed to generate the value of the signature. This process differs from that used in generating conventional digital signatures because message digests are not directly signed. The entire <ds:SignedInfo> element is signed instead because for each signature multiple message digests may exist but one signature value is required.

The algorithm used to canonicalise the <ds:SignedInfo> element is held in the <ds:CanonicalizationMethod> element and the digital signature algorithm is specified in the <ds:SignatureMethod> element. The preceding snippet shows that the signature is generated by a combination of exclusive canonicalisation and the Keyed-Hashing for Message Authentication (HMAC) with SHA-1 algorithm [Krawczyk et al., 1997]. The HMAC-SHA1 algorithm utilises the symmetric key mentioned in 3.6.1 to generate a MAC with the SHA-1 algorithm. This is consistent with XML Signature which allows a variety of signature generation algorithms to be used such as the Digital Signature Algorithm (DSA) ["NIST", 2000] and MAC algorithms. The value of the signature is placed in the <ds:SignatureValue> element:

```
<ds:SignatureValue>FE3...W+4=</ds:SignatureValue>
```

The signatures shown in this section are detached signatures and XML Signature defines such signatures as those that have the <ds:Signature> element and the signed elements as siblings or in separate documents [Eastlake et al., 2005]. XML Signature specifies two other kinds of signature: enveloped and enveloping . An enveloped signature requires that the <ds:Signature> element is a child of the signed element. An enveloping signature has the signed element as a child of the <ds:Signature> element. WS-Security states that detached signatures are the only permitted signature because SOAP headers are prone to change [Nadalin et al., 2006a]. Unfortunately the specification does not further expand on this rationale. The author finds that the reasons for such a stance are not immediately obvious, although other authors such as Rosenberg et al. [2004] do not further explain it. The reasons why the enveloping and enveloped signatures are inappropriate for WS-Security are explored here.

#### 3.6.2.3 WS-Security and XML Signature types

WS-Security specifies that all "security-related information" must be held in a <wsse:Security> header [Nadalin et al., 2006a]. It also allows for an intermediary to insert

elements into an existing <wsse:Security> header, hence its description of SOAP headers as "mutable" [Nadalin et al., 2006a]. The only signature that is compatible with these two conditions is the detached signature. The <ds:Signature> element represents a detached signature if it exists within the <wsse:Security> header and references message parts that are not its parent element or its child element. This implies that the <wsse:Security> header cannot be signed as a single unit. Therefore, if the <ds:Signature> element exists within the <wsse:Security> header and has a reference to the entire header, it represents an enveloped signature.

The WS-I Basic Security Profile [McIntosh et al., 2007] and the blogs authored by Wilson [2004] and Tay [2004] are the only resources found by the author that further discuss the implications of utilising an enveloped signature with WS-Security. Tay [2004] addresses the implications more directly in his blog post in comparison to Wilson [2004]. However, the opinion of Wilson [2004] is referred to here as he was the team lead for Microsoft's Web Services Enhancement project [Microsoft Corporation, 2007b] at the time at which he wrote on the issue. The blog post made by Tay [2004] is not considered because it contains some critical errors in terminology.

Wilson [2004] and the WS-I Basic Security Profile [McIntosh et al., 2007] state that the enveloped signature prevents intermediaries from adding new elements into the security header. If an intermediary were to insert an element into the header, the message digest of the header would change and not correspond to the message digest carried in the <ds:Reference> element. This would cause the signature to fail even if the intermediary was authorised to change the header. The enveloping signature type is subsequently incompatible with WS-Security because it does not take into consideration that the <wsse:Security> header may be modified by an intermediary. The detached signature allows for such modifications because it operates at a finer granularity. Instead of signing the entire header, parts of the header are signed. This allows new parts to be inserted by intermediaries.

The enveloping signature is also incompatible with WS-Security because it allows security information to be carried outside the <wsse:Security> header [McIntosh et al., 2007]. An example of what an enveloping signature would look like for a <wsse:Security> header is shown in the following snippet:

```
<ds:Signature>

 <ds:SignedInfo>...</ds:SignedInfo>
 ...
 <Object>
```

```
  <wsse:Security>
 ...
 </wsse:Security>
 </Object>
</ds:Signature>
```

The problem highlighted in this snippet is that security information appears outside the `<wsse:Security>` header, for example security information is found in the `<ds:SignedInfo>` element. This is inconsistent with the WS-Security approach of the `<wsse:Security>` header being the parent XML node from which all security information is found.

The author notes that although the WSI-Basic Security Profile [McIntosh et al., 2007] discusses the issue of which type of signature to use, this discussion is based on the previous WS-Security specification, version 1.0 [Nadalin et al., 2004]. The discussion provides the same reasoning as that reached here but it reaches a different conclusion from WS-Security version 1.1 [Nadalin et al., 2006a] on which signature types should be used. The Basic Security Profile [McIntosh et al., 2007] prohibits use of enveloping signature but "discourages" the enveloped signature. WS-Security version 1.1 prohibits the enveloping signature's use. The author accepts the analysis provided by WSI-Basic Security Profile but its conclusions on which signatures may be used are considered outdated for this thesis.

#### 3.6.2.4 Summary

WS-Security uses digital signatures as described by XML Signature to achieve confidentiality[Nadalin et al., 2006a]. These signatures are specific to XML but enveloped and enveloping signatures are inappropriate for SOAP security. To this end, WS-Security selects the detached signature as the only XML signature type that is adequate for providing end-to-end SOAP message integrity.

### 3.6.3 Authentication through Tokens

The approach to authentication taken by WS-Security is the provision of "claims" which are defined as:

> ... a declaration made by an entity (e.g name, identity, key ,group, privilege, capability etc) [Nadalin et al., 2006a].

These claims are provided by tokens attached in the <wsse:Security> header. Tokens may be binary such as Kerberos tickets [Nadalin et al., 2006b] and X.509 certificates [Nadalin et al., 2006e]. Binary tokens are attached using the <wsse:BinarySecurityToken> element. Tokens may also be XML-based such as the Security Assertion Markup Language (SAML) tokens [Nadalin et al., 2006c] and the <UsernameToken> element [Nadalin et al., 2006d]. XML tokens are included as direct sub-elements of the <wsse:Security> header.

Claims may be categorised either as endorsed or unendorsed [Singhal et al., 2007]. An endorsed claim is one with which identity can be inherently established. An example of a token representing an endorsed claim is an X.509 certificate token [Nadalin et al., 2006a]. The certificate represented by the token can be trusted to belong to a specific identity if a trusted certificate authority has endorsed it. Digital signature verification may be used to determine whether signed message parts were signed with the key referenced by the certificate [Singhal et al., 2007]. This provides data authentication by guaranteeing that the signed message parts were constructed by the certificate holder.

An unendorsed claim is one with which an identity cannot be inherently established [Singhal et al., 2007]. An example of a token representing such a claim is the <UsernameToken> element.

```
<wsse:UsernameToken...>
<wsse:Username>wsituser</wsse:Username>
<wsse:Password...>...</wsse:Password>
...
</wsse:UsernameToken>
```

The <UsernameToken> element holds a username and password pair [Nadalin et al., 2006d]. Peer authentication may be established by this pair but a trusted third party cannot vouch for the information presented by the token [Singhal et al., 2007]. The decision to accept or reject these credentials is determined by the trust policy employed by the authenticating entity because an identity cannot be inherently determined using the <UsernameToken> element.

Data authentication may also be achieved with the <UsernameToken> element since a signing key may be derived from the password it holds [Nadalin et al., 2006d]. Digital signature verification will confirm whether signed message parts were signed by token. However, since the identity established from peer authentication is inconclusive, this data authentication may be meaningless. It proves only that the identity referenced by the token created the message parts but that identity may have not been completely established.

SAML tokens are another commonly advocated mechanism for representing claims [Nadalin et al., 2006c]. SAML allows entities to authenticate each other through assertions [Ragouzis et al., 2006]. An entity authenticates another based on whether its trust policy allows it to accept the assertions received. This means SAML provides unendorsed claims too [Singhal et al., 2007]. SAML assertions will provide data authentication if they reference an encryption key that signs parts of a message [Ragouzis et al., 2006]. In this case the authentication process is twofold: Firstly, the signatures are verified according to the referenced key. Once this verification is done, the SAML assertions are considered. This process suffers from the same weakness as data authentication with the `<UsernameToken>` element because it proves that the asserting entity created the message but the assertions do not inherently prove the identity of this entity.

#### 3.6.3.1 Summary

The extensibility provided by WS-Security allows for an "infinite" number of authentication mechanisms [Seely, 2002]. Authentication mechanisms implemented as endorsed claims simplify the trust considerations of an endpoint because the identities represented by such claims have been verified by a third party. However, mechanisms implemented as unendorsed claims require an endpoint to do extra work in determining whether the identities represented by the claims should be trusted. For this reason endorsed claims seem more desirable than unendorsed claims.

The aspect of trust associated with these authentication mechanisms is not considered in order to keep within the scope established in section 3.3. The trustworthiness of all claims is considered equal for the purposes of this thesis.

### 3.6.4 Message Uniqueness

The WSI name two types of information required to assure SOAP message uniqueness: a "unique message identifier" and a "timestamp" [Schwarz et al., 2007]. A unique message identifier distinguishes a message from other messages. A timestamp specifies the message creation and expiry time.

A unique message identifier allows a receiving entity to reject the message if the same identifier has already been received but after a period of time, the store of previous identifiers may be purged. This allows an attacker to wait for a period of time before resubmitting a message. A timestamp allows a receiving entity to reject a message if it is received after its expiry

time or if its creation time is earlier than that of the latest store purge. WS-Security specifies a <wsu:Timestamp> element for setting the lifetime of the security information held in the <wsse:Security> header [Nadalin et al., 2006a]:

```
<wsu:Timestamp ...>
<wsu:Created>2007-06-18T11:43:31Z</wsu:Created>
    <wsu:Expires>2007-06-18T11:48:31Z</wsu:Expires>
</wsu:Timestamp>
```

The <wsu:Created> element specifies when the security information was created and the <wsu:Expires> element specifies when the security information lifetime ends. A SOAP message is rejected by a receiving entity if it is received after the time specified by the <wsu:Expires> element. This element does not provide a unique message identifier and this allows a SOAP message to be resubmitted by an attacker before the expiry time. The WSI identify the Username token as the only security token from the set of tokens they review that provides both a unique message identifier and timestamp [Schwarz et al., 2007]:

```
<wsse:UsernameToken...>
    ...
    <wsse:Nonce>...</wsse:Nonce>
    <wsu:Created>...</wsu:Created>
</wsse:UsernameToken>
```

The <wsse:Nonce> element holds a nonce [Shirey, 2000] that may be used to uniquely identify a message [Nadalin et al., 2006d]. The <wsu:Created> element states when the nonce was created. A receiving entity will reject a message if the nonce exists within its nonce store or if it was created before the last nonce store purge.

#### 3.6.4.1 Summary

The username token inherently represents both a unique message identifier and a timestamp [Schwarz et al., 2007]. WS-Security does not limit the type of tokens that may be used to protect a SOAP message and other tokens that provide a unique message identifier and a timestamp may exist. It is noted that the username token is the only token, amongst those mentioned explicitly by WS-Security, that provides both types of information [Schwarz et al., 2007].

### 3.6.5 WS-Security Summary

NIST describe some shortcomings of WS-Security which fall within the messaging domain and others within other domains, for example certificate validation within the trust domain [Singhal et al., 2007]. Concerns that lie in other domains may seem unwarranted as WS-Security focuses on message security [Nadalin et al., 2006a]. These concerns demonstrate that WS-Security has implications for issues outside the scope of this thesis.

The concerns that fall within the messaging domain are the susceptibility of WS-Security to replay attacks and the lack of protection for tokens [Singhal et al., 2007]. These concerns are valid when WS-Security is implemented as an isolated solution to an individual challenge. For example, a token used for authentication can be substituted by an attacker if the WS-Security integrity protection mechanisms are not implemented. Similarly, WS-Security implementations are susceptible to replay attacks if the challenge of message uniqueness is not considered. A holistic approach to the implementation of WS-Security is needed to meet each of the challenges mentioned in section 3.4.1.

The construction of end-to-end secured messages falls under the messaging aspect of web services introduced in section 2.2.3. This construction may be informed by a description stage [Rosenberg et al., 2004].

## 3.7 WS-Security Description

Messages may be secured without the description stage noted in section 2.2.3 [Rosenberg et al., 2004]. However, the description of security requirements in a machine readable language is important because it allows a further realisation of the web services promise of machine-to-machine interaction. Such a description also provides a standard mechanism to communicate security requirements, the lack of which is identified in section 2.6 as a shortcoming of RESTful web services.

A brief review of the description of WS-Security requirements is provided in this section. The review is not as comprehensive as the review of secure messaging given in this chapter. End-to-end message security cannot be implemented without a comprehensive consideration of secure messaging but it may be achieved without a machine readable description. The description of security requirements is mentioned here to provide a complete picture of end-to-end web services security.

### 3.7.1 WS-Policy

The WS-Policy Framework recommendation [Vedamuthu et al., 2007b] specifies how a set of requirement assertions may be constructed. These assertions communicate security and other requirements in a web services environment. WS-Policy assertions and WSDL seemingly perform the same function of description but WS-Policy differs from WSDL in that it allows any web services entity to communicate its requirements [Rosenberg et al., 2004]. WSDL allows for the communication of the provider's requirements only.

WSDL does not provide an inherent mechanism to describe security requirements. This is consistent with the SOAP design strategy of leaving security to be implemented as an extension [Gudgin et al., 2007]. To this end, WS-Policy assertions may be used as an extension to WSDL and this is described by the WS-Policy Attachments document [Vedamuthu et al., 2007a]. However, they may also exist as a standalone description of any entity's requirements.

The strategy of using WS-Policy assertions within WSDL is reviewed. This strategy allows for the provider's requirements to be specified only. This is appropriate for this thesis because it is concerned with whether a mobile requester can interoperate with a secured traditional provider. In this regard, only the provider's requirements are relevant to this study.

Requirements are represented in WS-Policy by assertions grouped into policy expressions [Vedamuthu et al., 2007b]. Policy expressions are held within a `<wsp:Policy>` element:

```
<wsp:Policy wsu:Id="SecureCalcPortBinding_add_Input_Policy">

    <wsp:ExactlyOne>
       <wsp:All>
       <ns7:EncryptedParts>
          <ns7:Body/>
       </ns7:EncryptedParts>
       <ns8:SignedParts>
          ...
       </ns8:SignedParts>
    ...
    </wsp:All>
    </wsp:ExactlyOne>
```

```
</wsp:Policy>
```

The policy expression in the snippet specifies the security configurations for an add message of the secure calculator web service referred to in section 3.6. Each policy expression has a set of operators that specify how assertions are applied [Vedamuthu et al., 2007b]. Operators may be nested but inner operators may not contradict their parent operators. The snippet utilises both the `wsp:ExactlyOne` and `wsp:All operators`.

The `wsp:ExactlyOne` and `wsp:All` operators are represented by XML elements of the same name [Vedamuthu et al., 2007b]. WS-Policy operators are formatted without XML style braces in the text of this thesis. This distinguishes references to operators from references to their XML element representation and is consistent with the WS-Policy document formatting style.

The `<wsp:ExactlyOne>` element must carry at at least one sub-element representing a group of assertions or further policy expressions. The requirements stated by any of its sub-elements are mandatory. The `<wsp:All>` element specifies that all the requirements represented by its sub-elements must be met but it may contain no sub-elements. Such an absence indicates that no requirements are specified. However, if the `<wsp:ExactlyOne>` element has no child elements this indicates that the policy it represents is not allowed.

The `wsp:All` operator in the snippet specifies that the requester must both encrypt the parts specified in the `<ns7:EncryptedParts>` element and sign the parts specified in the `<ns8:SignedParts>` element. The `wsp:ExactlyOne` operator specifies that these requirements are mandatory, that is, a requester cannot ignore the requirements the `wsp:All` operator relates to. The WS-Policy Framework is generic and may be used to describe requirements other than those of WS-Security.

### 3.7.2 WS-Policy and WS-Security

The WS-Policy Framework recommendation document uses assertions that describe WS-Security requirements in its examples [Vedamuthu et al., 2007b]. However, the recommendation does not fully describe the assertions that may be used to describe WS-Security requirements.

The WS-SecurityPolicy specification details WS-Policy assertions that describe security requirements as per WS-Security [Nadalin et al., 2007] . This specification provides four groups of assertions: protection; token; security binding and supporting token. Protection assertions describe which parts of the SOAP message need to be secured and how they are secured. An example of such an assertion is the `<ns7:EncryptedParts>` element which describes the parts of the message that are to be encrypted.

Token assertions describe tokens that are used to secure messages, for example the username token [Nadalin et al., 2007]:

```
<ns3:UsernameToken
ns3:IncludeToken="http://schemas.xmlsoap.org/ws
/2005/07/securitypolicy/IncludeToken/AlwaysToRecipient">
```

The IncludeToken attribute describes whether the token should be included and when it should included. The snippet shows that the username token must be sent from the requester to the producer only.

Security binding assertions describe how security mechanisms are configured. Supporting token assertions specify tokens that are used by the mechanisms described in security binding assertions [Nadalin et al., 2007]:

```
<ns3:SignedSupportingTokens>
...
    <ns3:UsernameToken...></ns3:UsernameToken>
...
</ns3:SignedSupportingTokens>
```

This snippet indicates that a username token is included in a message signature. The token that creates this signature is described by a security binding assertion [Nadalin et al., 2007]:

```
<ns4:SymmetricBinding>
...
    <ns4:AlgorithmSuite>
    ...
       <ns4:Basic128/>
    ...
    </ns4:AlgorithmSuite>
    <ns4:IncludeTimestamp/>
    <ns4:Layout>
    ...
       <ns4:Lax/>
```

```
        ...
        </ns4:Layout>
        ...
        <ns4:OnlySignEntireHeadersAndBody/>
        ...
    </ns4:SymmetricBinding>
```

WS-Policy specific policy expressions have been removed from this snippet to demonstrate the assertions specified by WS-SecurityPolicy [Nadalin et al., 2007]. The assertion shown in the snippet specifies that a 128 bit symmetric key is required for encryption using the AES algorithm. This key must also be used to sign the message headers and body. A timestamp must be included in the message and the ordering of the headers must conform to Lax ordering as specified by WS-Security [Nadalin et al., 2006a].

A combination of the assertions specified by the WS-Policy Framework recommendation [Vedamuthu et al., 2007b] and WS-SecurityPolicy [Nadalin et al., 2007] allows for the complete description of WS-Security requirements. The following snippet shows an example of this combination:

```
<wsp:ExactlyOne>
    <wsp:All>
        <ns4:Basic128/>
    </wsp:All>
</wsp:ExactlyOne>
```

This snippet shows that the AES algorithm must be used with a 128 bit key. The `wsp:ExactlyOne` and `wsp:All` operators specify that the use of these encryption algorithm details is mandatory. One of the features of WS-Policy expressions is their suitability for inclusion in WSDL documents.

### 3.7.3 WS-Policy and WSDL

The primary difference between the WSDL document of the simple calculator web service introduced in section 2.3 and that of its secure version introduced in section 3.6, is the attachment of

WS-Policy expressions in the secure version's WSDL document. The WS-Policy Attachments recommendation details how such an attachment maybe carried out [Vedamuthu et al., 2007a].

A policy expression may be attached as a child element of the <definitions> element in a WSDL file [Vedamuthu et al., 2007a]. The WSDL parts that an expression applies must reference this expression:

```
<binding name="SecureCalcPortBinding"..>
    ...
     <operation name="add">
    ...
       <input>
       <wsp:PolicyReference
       URI="#SecureCalcPortBinding\_add\_Input\_Policy"/>
       ...
       </input>
    ...
    </operation>
    ...
</binding>
```

The reference to a policy expression is held within a <wsp:PolicyReference> element. In this snippet the <wsp:PolicyReference> element is inserted inside the concrete description of the add message. The requester will secure the described add message according to the expression referenced.

### 3.7.4 Summary

A combination of the WS-Policy Framework [Vedamuthu et al., 2007b] , WS-Policy Attachment [Vedamuthu et al., 2007a] and WS-SecurityPolicy [Nadalin et al., 2007] specifications allows for a machine readable description of web services security requirements . The failure to process or provide such a description does not disqualify an entity from engaging in an end-to-end secured web services transaction because human developers can build such security in by hand [Rosenberg et al., 2004]. However, the machine generation allowed by such descriptions mitigates some of the complexity associated with building in these security requirements by hand.

## 3.8 Summary

End-to-end web services security is provided by implementations of WS-Security. The WS-I Basic Security Profile describes how WS-Security may be implemented in an interoperable manner [McIntosh et al., 2007]. The discrepancies that may exist when the WS-I Basic Security Profile is applied to WS-Security version 1.1 are demonstrated by the WS-I Basic Security Profile's more relaxed prescription of which signature types may be used. However, the WS-I Basic Security Profile provides a standard baseline set of requirements for the interoperable deployment of end-to-end web services security. This set of requirements is implemented on traditional web services platforms as evidenced by the Microsoft WS-I Basic Security Profile 1.0 Reference Implementation [Microsoft Corporation, 2007d].

This thesis is concerned with the interoperability between mobile requesters and traditional providers when end-to-end security is employed by the providers. For such interoperability mobile requesters must be able to implement WS-Security. They must also conform to the WS-I Basic Security Profile to the extent that traditional web services providers do. Therefore, this thesis's requirement for interoperability is not based on a mobile requesters conformance to the WS-I Basic Security Profile. It is based on the extent to which a mobile requester may participate in WS-Security secured transaction with a traditional provider. The following chapter discusses the considerations for implementing WS-Security for a mobile requester.

# Chapter 4

# Mobile Web Services

## 4.1 Introduction

This chapter deals with mobile web services and their end-to-end security. The smartphone implementation of WS-Security, introduced in chapter 3, is discussed. The challenges of the mobile computing paradigm are identified in this chapter and a configuration for the implementation of WS-Security on smartphones is developed according to these challenges. The configuration presents a base set of end-to-end security mechanisms that a smartphone platform may be reasonably expected to implement and thus participate in end-to-end secured web services transactions.

Mobile web services are a by-product of the convergence between the fields of mobile computing and web services [Hirsch et al., 2006]. The mobile environment is diverse as evidenced by the fact that a mobile device may be considered as any device ranging in size from a pager to a laptop [Li and Knudsen, 2005] [Teder, 2006]. Mobile environment concepts utilised within this thesis are identified to limit the scope of discussion.

## 4.2 Mobile Concepts

Section 1.3.2 on page 3 motivates for the limitation of scope to smartphones. There is no universal definition for a smartphone as evidenced by loose definitions provided by Borcea et al. [2004] and Chang and Chen [2005]. However, the convergence of telephony and PDA functionality is a common smartphone characteristic mentioned in literature discussing smartphones, for example it is mentioned in the work of Borcea et al. [2004], Chang and Chen [2005], Fox and Box [2004] and Zheng and Ni [2006]. This thesis adopts a smartphone definition by [Yuan, 2005] because it is centred on this commonly accepted smartphone trait:

> A Smartphone combines the functions of a cellular phone and a handheld computer in a single device. It differs from a normal phone in that it has an operating system and local storage, so users can add and store information, send and receive email, and install programs to the phone as they could with a PDA. A smartphone gives users the best of both worlds–it has much the same capabilities as a handheld computer and the communications ability of a mobile phone.

In technical terms, the smartphones discussed here exhibit the same characteristics as the family of devices covered by the Java Micro Edition (Java ME) Connected Limited Device Configuration (CDLC) [Pandey, 2006a] . The characteristics include a 16 or 32 bit processor, a battery as the main source of power and the ability to operate with limited network access. The provision of end-to-end web services security by platforms that run on this type of device is of interest to this thesis.

The phone function of a smartphone strongly ties it to the cellular network to which it connects. As a result, cellular network standardisation bodies such as the European Telecommunications Standards Institute (ETSI) and the 3rd Generation Partnership Project (3GPP) define some mobile device aspects [ETSI, 2007] [3GPP, 2007a]. One such aspect is the distinction between the Subscriber Identity Module (SIM) and the device that houses the SIM [Third Generation Partnership Project, 2001b]. The SIM is a smart card that holds subscriber information on a Global System for Mobile (GSM) network [Third Generation Partnership Project, 2001a]. Its equivalent on the Universal Mobile Telecommunication System (UMTS) network is the Universal Subscriber Identity Module (USIM) application that runs on a smart card called the UMTS Integrated Circuit Card (UICC) [Kasera and Narang, 2005]. This thesis refers to the SIM and USIM/UICC combination as the SIM because the differences between the two are not significant to the research presented. The distinction between the SIM and its housing device, such as a smartphone, is important because the SIM is a security device that may be used to meet the security challenges identified in the previous chapter [Third Generation Partnership Project, 2001b]. The SIM presents another type of mobile platform that may assist smartphone platforms in the provision of end-to-end mobile web services security.

The SIM is specified for use on GSM and UMTS networks by ETSI and 3GPP specifications respectively. However, smartphones do not work only on GSM and UMTS; for example the second generation Personal Digital Communications (PDC) network is popular in Japan and the third generation cdma2000 network is popular in North America [Kasera and Narang, 2005]. The phone function of smartphones is not dependant on a cellular network [Teder, 2006]. Smartphones may leverage wireless LAN (WLAN) technologies to complete their cellular phone func-

tion [Hsieh et al., 2007]. It is for this reason that the discussion that follows is cellular network agnostic unless stated otherwise. When network specific considerations such as the SIM are discussed, they are noted as such.

The consumer devices related concept of fragmentation is also an attribute of the mobile phone domain [Teder, 2006]. The term fragmentation refers to the state of a domain when the variation within that domain hinders the ability to deploy enough numbers of a product to make a profit. Web services are valuable to the mobile environment because they offset the problem of fragmentation.

## 4.3 Why Mobile Web Services ?

The need for end-to-end mobile web services security is established here before it is discussed further. The importance of mobile web services is briefly introduced to demonstrate that they are a valuable technology requiring real world security attention. The reasons that make cellular network security inappropriate for web services are then discussed.

### 4.3.1 The Importance of Mobile Web Services

Mobility is a primary benefit provided by smartphones [Fox and Box, 2004]. Smartphone applications allow faster access to information from less location-constrained access points. A major benefit of mobility is improved decision making as a result of readily available information.

The mobile phone environment is characterised by different devices, platforms and Application Programming Interfaces (API) [Teder, 2006]. This diversity poses two mobility deployment barriers: Firstly, it increases the number of application builds needed to profitably deploy an application in the smartphone market; Secondly, an Internet application must be customised for every mobile configuration with which it interfaces. The large number of existing configurations makes this complex and time consuming [Open Mobile Alliance, 2006]. These two problems inevitably hinder the advancement of the mobile phone environment because it depends on “low cost data services” for growth [Open Mobile Alliance, 2006]. Web services solve the fragmentation problem faced when interfacing Internet applications with mobile applications.

Interoperability is a major value proposition of web services. They provide an interface between systems that may be implemented on different platforms [Wright, 2005]. This interoperability may be extended to the interfacing of Internet and mobile applications [Open Mobile Alliance, 2006]. Web services expose an interoperable interface through which smartphone applications running on various platforms may access an Internet application. The provision of this

interface is the single customisation to the Internet application that is required to provide such access. This is significant in enterprise environments because it reduces the costs of integrating smartphones applications with currently deployed systems.

Although mobile web services overcome one of the problems to providing mobility, barriers to web services adoption must be overcome for them to be a viable solution. The lack of web services security is such a barrier [Kearney, 2005].

### 4.3.2 The Need for End-to-end Security

Cellular networks may provide confidentiality, integrity and authentication for messages carried by them. An example is the security provided by the General Packet Radio Service (GPRS) on GSM networks [Sanders et al., 2003]. However, this protection only lasts the extent of the mobile network and mobile web service messages may travel beyond this network [Moyo et al., 2006]. Messages are therefore vulnerable to malicious activity once they leave the mobile network. The web services message security strategies discussed in section 3.5 protect a web services message beyond the context of the mobile network.

Cellular network security may be inappropriate even when the entire transaction occurs within the context of the mobile network. Some cellular network security is proprietary and has been kept from public review, for example GSM security [Gindraux and Deloitte & Touche, 2002]. This requires web services participants to trust the mobile network providers in the provision of security. Reported vulnerabilities within proprietary cellular network security mechanisms show that such trust is unwarranted [Biryukov et al., 2001] [ins]. Web services message security allows web services participants to take ownership of message protection and select the security mechanisms they trust [Moyo et al., 2006].

The issue of blindly trusting cellular network security does not universally apply because some cellular network security is available for public review, for example the openly specified Kasumi algorithm used by UMTS networks [3GPP, 2007b]. However, such security operates at a lower level than transport layer security and suffers from the same inability to provide end-to-end security when intermediaries exist. It is dependent on the communication links between web services entities and requires intermediaries to be trusted for the provision of end-to-end security.

The mobile computing paradigm differs from traditional computing paradigms and it is inappropriate to assume that WS-Security may be implemented on smartphone platforms in the same manner as on traditional platforms [Nadalin, 2003]. Mobile environment considerations are factored in when implementing WS-Security on smartphone platforms.

## 4.4 Mobile Environment Considerations

Mobile web services operate in a more resource constrained environment than traditional web services [Hirsch et al., 2006]. These constraints include limited processing power, battery life and network bandwidth. WS-Security is shown to add a performance overhead to web services implementations [Tang et al., 2006]. This section identifies the resource considerations that must be accounted for when WS-Security is implemented for mobile web services. These considerations are categorised as device constraints and network constraints.

### 4.4.1 Device Constraints

The current processing power and memory of smartphones renders the implementation of cryptographic operations on smartphones a solved problem [Zheng and Ni, 2006]. Argyroudis et al. [2004] analyse the implementation of TLS , S/MIME [Ramsdell, 1999] and IPSec [Kent and Atkinson, 1998] on PDAs, Itani and Kayssi [2004] successfully demonstrate an AES algorithm implementation on a smartphone. However, the adverse effect that CPU processing and limited memory have on battery life is an area of concern when implementing WS-Security.

The constraints of battery life and processing power are obviously linked because the smartphone CPU is powered by the battery. Battery life will decrease as the processing time spent on expensive cryptographic operations increases [Argyroudis et al., 2004]. Reducing the time spent on processing WS-Security related cryptographic operations will prolong battery life. Two approaches are identified for doing this:

The first approach is to offload cryptographic operations from the CPU to specialist devices that optimally carry out the operations. These specialist devices may be inherent to the hardware, for example cryptographic engines running as part of a multi-processor architecture [Ashkenazi and Akselrod, 2007]. The specialist devices may also be separate devices such as the SIM. Such devices also consume battery power but their ability to carry out a cryptographic process more efficiently may result in battery power saving.

The second approach is to employ processor efficient cryptographic operations. These operations must also minimise memory usage because memory bottlenecks may aggravate the battery power consumption of a CPU [Martin, 1999]. This second approach is more ideal than the first of offloading cryptographic operations because it is device independent. However, sending data over the network consumes battery power more than CPU usage [Kangasharju, 2007].

### 4.4.2 Network Constraints

Mobile web services operate in a networked environment characterised by limited bandwidth and high latency [Hirsch et al., 2006]. The requirement that devices continue processing in the absence of a network connection is also an environmental norm.

Web services are more verbose in their messaging than traditional web applications [Ng et al., 2005]. The addition of the `<wsse:Security>` header further compounds the size of a web services message . This increase is attributed to the extra XML of the attached header and the Base64 encoding [Josefsson, 2003] of binary security data such as cipher-text, signatures and hashes. A Base64 encoding representation is 33% larger than the data it represents. The growth of data resulting from WS-Security is problematic because mobile subscribers are often billed by the amount of data they transfer [Tian et al., 2004] and a mobile device's battery power is consumed more by data transfer operations than CPU processing [Kangasharju, 2007]. Minimising the size of the data sent over a mobile network is desirable.

Solutions exist to mitigate the additional size of SOAP messages, for example data compression and binary XML encodings [Tian et al., 2004]. Combining data compression with XML Encryption is a poor solution because data compression yields better results when applied to data before it is encrypted [Kangasharju et al., 2006]. Compression algorithms utilise patterns to reduce the size of data [Kangasharju, 2007]. Encryption algorithms work against this compression approach because they convert plain-text into cipher-text containing a minimal amount of patterns. This renders compression algorithms less efficient in reducing the size of cipher-text than that of plain-text. To this end, it is better to compress plain-text and then encrypt the compressed plain-text.

Compressing and then encrypting XML data results in interoperability problems. XML Encryption does not provide a mechanism to instruct a decrypting entity that it must decompress data after it is decrypted. Such instructions must be provided by non standard extensions to XML Encryption. Therefore, binary XML encodings are preferred for reducing the size of secured XML elements [Kangasharju et al., 2006].

The Message Transmission Optimization Mechanism (MTOM) [Gudgin et al., 2005] is, at the time of writing, the only approved standard solution for the reduction of SOAP message sizes. It is currently standardised as a W3C Recommendation. MTOM optimises the representation of binary data within a SOAP message thereby reducing the size of the message. This is useful for WS-Security because it reduces the additional message size resulting from the Base64 encoding of binary data [Ng et al., 2005].

Binary encoding such as those provided by .NET Remoting perform better than MTOM

because they optimise the entire message as opposed to just binary data [Ng et al., 2005]. Such solutions introduce the dilemma of trading off interoperability for performance because they are not standard. The Efficient XML Interchange (EXI) format is a W3C attempt to standardise a binary encoding for XML and it is currently in the draft phase [Schneider and Kamiya, 2007]. However, Microsoft's opposition to the concept of a standardised binary encoding casts doubt over its future universal adoption [Pal et al., 2003].

The mobile network constraints of high latency and disconnected operation may be handled programatically through the use of asynchronous method calls [Kangasharju et al., 2007]. An asynchronous method does not block after it has finished executing for example, when a requester sends a SOAP message to a provider [Hirsch et al., 2006]. This allows processing to take place while a response is pending. On the response's arrival a callback method is invoked allowing the response to be handled.

WS-Security must be implemented within the context of the mobile environment constraints presented in this section. The architectures with which WS-Security may be provided in a mobile environment are presented in the following section.

## 4.5 Mobile Web Services Architecture

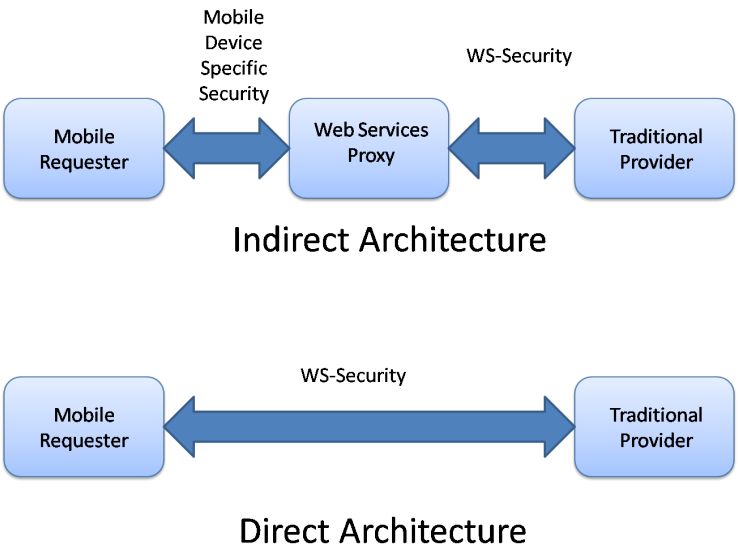


Figure 4.1: Mobile web services architectures [Open Mobile Alliance, 2006].

OMA identify the indirect and direct mobile web services architectures as models for providing mobile web services [Open Mobile Alliance, 2006]. Figure 4.1 illustrates the difference

between these architectures.

The indirect architecture is appropriate for requesters without the full functionality required to interact with a provider [Open Mobile Alliance, 2006]. This architecture is closely associated with the Wireless Application Protocol (WAP) which provides Internet access to mobile devices that do not possess the capability to connect using traditional Internet protocols [Gupta and Gupta, 2001]. This architecture may be applied in the web services domain so that a requester without the capability to engage in web services transactions may communicate with a provider via a proxy [Open Mobile Alliance, 2006] . The proxy typically translates between a mobile device specific protocol and the SOAP protocol.

This indirect architecture is of interest to this thesis because the proxy may also be used to secure a SOAP message when a requester does not possess this capability [Kangasharju et al., 2006]. Figure 4.1 illustrates the use of the indirect architecture when a mobile requester can process SOAP messages but is unable to apply WS-Security to them. The requester secures the messages that travel between it and the proxy with a security mechanism within its set of capabilities, for example TLS. The proxy in turn secures these messages according to the WS-Security requirements of the provider. The indirect architecture suffers from some drawbacks including having to trust the proxy [Gupta and Gupta, 2001]. The two modes of security employed within this architecture are always broken at the proxy because one mode is terminated for the other to be instantiated. This allows the proxy unlimited access to all data as it converts the security from one mode to another. The direct architecture is an option for deploying WS-Security for mobile web services when a proxy cannot be trusted.

The direct architecture requires a mobile requester that fully supports the requirements to communicate with the provider [Open Mobile Alliance, 2006]. All processing, including the securing of SOAP messages, must be carried out by the provider. Figure 4.1 illustrates that there is no proxy needed for this architecture. To this end, the direct architecture cannot support as many devices as the indirect architecture because it depends on the capability of the platform hosting the requester. However, the direct architecture better adheres to the end-to-end argument because the security of the SOAP message is never broken during its transit between the requester and provider. This thesis bases its mobile web services security configuration on the direct architecture. Previous work on providing WS-Security for resource constrained devices with the direct architecture is discussed before this configuration is laid out.

## 4.6 Related Work

Three sets of related work are discussed in this section. These are the SOAP Message Security: Minimalist Profile by Nadalin [2003], the implementation of WS-Security on resource constrained devices by Helander and Xiong [2005] and an efficient XML security framework by Kangasharju [2007].

### 4.6.1 SOAP Message Security: Minimalist Profile

The SOAP Message Security: Minimalist Profile specifies how WS-Security may be implemented on resource constrained devices , such as smartphones [Nadalin, 2003]. The Minimalist Profile identifies XML Signature processing as a potential performance bottleneck and it specifies a stricter subset of WS-Security to increase the efficiency of this processing. It seeks to realise the greatest efficiency gain by eliminating the need for a message receiving entity to perform canonicalisation when verifying a signature. The Minimalist Profile makes this possible by specifying that signed message parts must be sent in canonical form. Therefore, an entity still needs to canonicalise the signed message parts it sends but not those it receives.

This thesis does not consider the requirements of the Minimalist Profile because it has been in draft since 2003 and seems to have been abandoned [Kangasharju, 2007]. It is the author's opinion that the Minimalist Profile's stance on canonicalisation will not always achieve a large efficiency benefit for mobile web services in practise. The current norm is to deploy mobile web services as requesters and a requester may send more signed data than it receives. This is evident in the secure calculator example introduced in section 3.6. The requester, in this example, signs message parts including the encrypted message body, authentication tokens and encrypted symmetric key. However, the provider signs only the encrypted message body of the response. The requester benefits from not canonicalising the single response message part but the requester still canonicalises the multiple message parts it sends. Therefore, the larger portion of work still needs to be carried out by the requester. This criticism may seem unfair because it does not apply generally but only to the case where a request message contains more signed data than the response message. However, no mention of this likely case is provided by the Minimalist Profile. This omission may be interpreted as evidence of the Profile's undeveloped nature. The implementation of WS-Security on resource constrained devices by Helander and Xiong [2005] also does not conform to this draft specification.

### 4.6.2 WS-Security on "Low-Cost Devices"

Helander and Xiong [2005] demonstrate the feasibility of implementing WS-Security on a resource constrained device. They combine symmetric and asymmetric cryptography with the aim of minimising asymmetric cryptographic operations. This reduces the performance penalty that asymmetric cryptography introduces over symmetric cryptography. Asymmetric encryption is utilised for the exchange of authentication data and symmetric encryption keys. The rest of the SOAP message parts are symmetrically encrypted with the exchanged symmetric key. Digital signatures are generated with the symmetric HMAC-SHA1 algorithm.

The work by Helander and Xiong [2005] differs from that presented in this thesis because it demonstrates WS-Security on a specialist device and not a smartphone. Such a demonstration is important to this thesis because it shows that WS-Security is technically feasible on a device with similar resource constraints as a smartphone. However, the demonstration does not provide insight into the extent that smartphone platforms leverage this technical capacity. The study by Kangasharju [2007] more tightly ties WS-Security on to a smartphone platform.

### 4.6.3 XML Security on Mobile Devices

Kangasharju [2007] describes an approach that efficiently implements XML Encryption and XML Signature on mobile devices. The approach is twofold: firstly, an API stream processes the XML Encryption, XML Signature and serialisation steps so that an XML document may be secured and serialised in one pass; secondly, the resulting serialised and secure XML is compressed before it is sent. They demonstrate the efficiency of this approach by creating a secure SOAP message that conforms to WS-Security.

This work is important to this thesis because it demonstrates the technical feasibility of applying XML Encryption and XML Signature on a smartphone. Since WS-Security is based on XML Encryption and XML Signature, the work by Kangasharju [2007] can also be taken to show the technical feasibility of WS-Security on smartphones.

The work by Kangasharju [2007] differs from that carried out in this thesis because it uses WS-Security as its proof of concept rather than its main subject matter. This is evidenced by its non-consideration of the challenges of authentication and message uniqueness. It also does not consider whether the security resulting from its described approach allows a mobile requester to interoperate with a traditional provider. However, the work presented by Kangasharju [2007] is an option for enabling WS-Security on smartphones. It is evaluated in the next chapter with other options that enable end-to-end mobile web services security. This evaluation is based on

the implementation of a configuration that provides WS-Security for mobile devices.

## 4.7 Mobile WS-Security Configuration

The three sets of related work highlighted in the previous section agree that a WS-Security implementation on resource constrained devices needs to factor in resource constraints. The configuration provided in this section identifies WS-Security mechanisms that meet the challenges of confidentiality, integrity, authentication and message uniqueness. The mechanisms are selected on the basis of how they support interoperability with traditional web services and how demanding they are on resources when mobile environment constraints are considered.

The configuration presented in this section does not suggest a one size fits all WS-Security configuration for all mobile web services. An appropriate configuration is dependent on the unique requirements of a particular deployment. Deployment specific security needs may render the configuration presented here inappropriate because it may provide good performance in the light of mobile environment constraints, but a poor level of security if it does not meet these needs.

The consideration of every deployment specific security need is intractable. To this end, the configuration presented is general because it takes into consideration the mobile environment constraints common to all mobile web services only. This generalisation provides reasonable criteria with which to examine the provision of interoperable end-to-end mobile web services security. This criteria forms a base set of WS-Security mechanisms that a smartphone platform must support for the provision of interoperable mobile web services security. Each WS-Security mechanism within this set is analysed under the security challenge it meets. However, these mechanisms are mostly relevant to device constraints. The aspect of the configuration that deals with network constraints is considered separately from these mechanisms.

### 4.7.1 Network Constraints

Section 4.4.2 discusses that one of the drawbacks of WS-Security is the extra data that needs to be sent across the network and mitigation of this data growth is desirable. However, for the purposes of this study, the message reduction must be done in a manner interoperable with traditional web services. Ng et al. [2005] compare data compression, binary XML encoding and MTOM as options for reducing message sizes. Their results show that in cases where XML Encryption and XML Signature are applied, binary XML encoding performs the best followed by MTOM

and finally data compression. MTOM is chosen to mitigate the impact of network constraints because it is standardised and may reduce the sizes of secured messages end-to-end.

Kangasharju [2007] states that the only XML binary encoding available for mobile devices is Xebu. However, Xebu, while open source, is not subject to any standardisation process, such as the process that is underway for EXI. It is also the authors experience that traditional web services do not support this type of encoding. Xebu is not considered because it may result in non-interoperability with traditional web services.

MTOM is the next best performing approach [Ng et al., 2005] and unlike Xebu it is standardised as a W3C recommendation. However, it has the drawback of optimising only the sections of a message that are Base64 encoded. It is relevant only for reducing message sizes that contain binary such as that resulting from meeting the challenges of confidentiality and integrity.

Section 4.4.2 rules out XML data compression as an option because its usage requires non standard extensions to XML Encryption which may result in non-interoperable security. The use of compression at the transport level, as demonstrated by Kangasharju [2007], is also inappropriate because it suffers from the same inadequacies as transport layer security. Such compression is reliant on intermediaries for its establishment and is out of the control of the two endpoints. The above-mentioned reasons and the finding by Ng et al. [2005] that data compression performs worse than MTOM when XML data is encrypted and signed, rule out data compression as an option for the configuration presented.

Despite the "drawback" of selective optimisation carried out by MTOM, it is selected for the configuration presented here. It is a more desirable option than XML binary encoding because it is standardised and it performs better than data compression. The following sub-sections consider mechanisms that meet web services messaging security challenges within the device constraints faced by mobile web services.

### 4.7.2 Confidentiality

Although encryption provides confidentiality section 4.4.1 discusses the adverse performance effect that such cryptographic operations have on mobile web services. Symmetric encryption is less resource demanding than asymmetric encryption and therefore more appealing for the smartphone. However, unlike asymmetric encryption, symmetric encryption requires the encryption keys to be mutually known [Ferguson and Schneier, 2003]. The problem of key exchange is factored into the strategy for this reason.

It is not surprising, when device constraints are considered, that GSM and UMTS networks provide for confidentiality with symmetric encryption [van der Merwe, 2003] [Kasera

and Narang, 2005]. The problem of key exchange on these networks is solved by pre-loading symmetric keys onto the SIM. WS-Security explicitly describes an alternative solution to the problem of key exchange and this entails encrypting a symmetric key with asymmetric encryption[Nadalin et al., 2006a]. This encrypted symmetric key is then shared between the encrypting and decrypting entities.

The approach of combining symmetric and asymmetric encryption is selected to meet the challenge of confidentiality. This is similar to the related work by Helander and Xiong [2005] which demonstrates an efficient combination of the two techniques on resource constrained devices. This efficiency is achieved by utilising asymmetric encryption for the encryption of symmetric keys only. This combination is still less efficient than the GSM/UMTS approach which requires no asymmetric encryption because the symmetric key is shared when the SIM is manufactured . However, the GSM/UMTS approach is not considered because it does not apply to smartphones on networks that do not utilise the SIM.

The GSM/UMTS approach limits the number of providers that mobile requesters interact with because the requesters may interact only with providers whose keys are loaded when the SIM is manufactured. The combination of symmetric and asymmetric encryption allows for requester interaction with an unlimited number of providers because a provider's public key may be extracted from it's publicly available digital certificate. The requester may download as many provider certificates as it wishes and generate a symmetric key for each.

Meeting the challenge of integrity also requires a consideration of symmetric and asymmetric cryptography

### 4.7.3 Integrity

The generation of digital signatures with symmetric signature generation algorithms is selected for providing integrity because symmetric cryptographic operations are less resource intensive than asymmetric ones [Ferguson and Schneier, 2003]. These algorithms also require a secret key to be shared but this problem is already solved with the key exchange for the encryption mentioned in the previous sub-section. The symmetric key utilised for encryption is utilised for signing data. However, the savings accrued from choosing symmetric algorithms over asymmetric algorithms may be minimal because XML canonicalisation causes the greatest overhead for XML Signature processing [Shirasuna et al., 2004].

The configuration described here identifies a minimum set of mechanisms that are used to identify whether a smartphone platform can provide interoperable end-to-end mobile web services security. The provision of an optimised canonicaliser does not improve this interoperabil-

ity and is therefore not a one of mechanism covered by the configuration. The provision of a symmetric signature generation algorithm is the only criteria identified for the strategy. This algorithm also provides for data authentication.

### 4.7.4 Authentication and Message Uniqueness

Peer and data authentication are identified in section 3.4.1 on page 35 as two types of authentication that may require different approaches. The challenge of message uniqueness is mentioned here with peer authentication because the configuration uses the same token to meet both challenges.

The username token is selected because it is the only token reviewed by the WS-I that plays the dual role of authentication and message uniqueness [Schwarz et al., 2007]. This approach is efficient because it allows two challenges to be met with one process. Another authentication token such the X.509 certificate requires another mechanism to be deployed for message uniqueness.

Data authentication is already provided for by the XML signature process that provides integrity. There is no need for a data authentication mechanism apart from that which meets the challenge of integrity. The verification of a digital signature in a message allows the provider to assume that the entity with which it shares the secret key, is the entity that created the signed data.

The authentication discussed thus far is only one way, that is a requester authenticating with a provider. However, the authentication of the provider by the requester may be assumed because a requester may need a copy of the provider's digital certificate to extract the provider's public key. This public key is used in the asymmetric encryption of the symmetric key mentioned in section 4.7.2. The identity of the provider may be determined through certificate validation and this may be managed through a smartphone platform's certificate management functionality. The Symbian, Java ME and Windows Mobile smartphone platforms provide functionality to manage certificates and the degree to which they are trusted [Symbian Software, 2007, Ortiz, 2005, Microsoft Corporation, 2007a].

## 4.8 Summary

This chapter provides a platform and cellular network independent configuration for the provision of WS-Security for mobile web services. Table 4.1 shows the configuration of mechanism within the configuration.The two considerations behind this configuration are interoperability

| | Device Constraints | Network Constraints |
|---|---|---|
| Confidentiality | Symmetric and Asymmetric Encryption Algorithms | MTOM |
| Integrity | Symmetric Signature Algorithm | MTOM |
| Authentication | Username Token | |
| Message Uniqueness | UsernameToken | |

Table 4.1: Mobile web services security configuration

and resource efficiency. Interoperability is a theme central to this thesis and the resource efficiency of mechanism is important because of the unique mobile environment constraints.

Interoperability and resource efficiency may in some cases be anti-thetical, for example binary XML encodings perform better than MTOM, but the standardisation of MTOM facilitates better interoperability. In other cases, a mechanism may offer efficiency gains without adding any value to the proposition of interoperability with traditional web services, for example an optimised canonicaliser.

This thesis is concerned with the interoperability issues faced by mobile web services when they interact with end-to-end secured traditional providers. The configuration presented in this chapter considers interoperability as a primary goal and efficiency as a secondary goal. This explains why MTOM is selected for the configuration over XML binary encoding options such as Xebu and why optimised canonicaliser are not mentioned by it. The following chapter discusses the implementation of the configuration presented here.

# Chapter 5

# Experiment

## 5.1 Introduction

The research goals of this thesis are laid out in section 1.5. The first goal of identifying an appropriate means to achieve interoperable, end-to-end, mobile web services security has been met in previous chapters. WS-Security was identified in chapter 3 as an end-to-end web services standard suitable for mobile web services. The mobile WS-Security configuration described in chapter 4 outlined a minimum set of mechanisms that a smartphone platform must support to be considered capable of providing interoperable, end-to-end, mobile web services security.

This chapter is concerned with the second research goal of determining the feasibility of implementing interoperable, end-to-end, mobile web services security. The feasibility of implementing WS-Security on smartphone hardware is demonstrated in the related work presented in section 4.6 and is considered a solved problem. This chapter is concerned with the software feasibility of implementing interoperable, end-to-end, mobile web services security. The term software feasibility here refers to the adequacy of available software libraries.

The investigation into this feasibility is conducted through an experiment with objectives which are laid out in the second section of this chapter. The experiment attempts to implement the mobile WS-Security configuration provided in the previous chapter. The introduction of experiment objectives is followed by a description of the method and technology used to carry out the experiment. The remainder of the chapter presents the findings of the experiment through a description of the attempted implementation of the mobile WS-Security configuration.

## 5.2 Experiment Objectives

The experiment set up in this chapter meets the second thesis research goal of determining the feasibility of interoperable, end-to-end, mobile web services security. This goal has two sub-goals that determine the nature of the experiment. These two sub-goals are described in this section as objectives of the experiment.

### 5.2.1 To Examine the State of Current Libraries

Current, WS-Security-enabling, smartphone software libraries are analysed through an implementation of the mobile WS-Security configuration detailed in section 4.7. The failures experienced in the implementation of the configuration are noted when they are due to inherent shortcomings of the libraries.

The development of a new API that implements the configuration is not an objective of this experiment per se. The development of a new API is carried out only when it aids the analysis of other existing APIs. The smartphone platforms used in the experiment are selected on the basis of how widely applicable the results from analysing their APIs, will be.

### 5.2.2 To Present Cross-Platform Results

Results that apply to all smartphones are the most desirable because they allow conclusions to be drawn about the mobile web services field in general. However, the generation of such results is impractical because of the fragmentation of the smartphone environment described in section 4.3.1, and the lack of a universally adopted smartphone platform.

The fragmentation of the smartphone environment makes the generation of results that apply to every smartphone intractable because a requester needs to be ported to every type of smartphone platform. The intricacies of each platform make this a complex task. The differences between the Symbian C++ language and the C++ language used to develop Window Mobile applications illustrates the complexity of such a porting effort [Wigley, 2005]. The two C++ languages are so different that they even define different C++ language primitives.

A porting effort would not be needed if a universally adopted smartphone platform existed. The Java ME platform is described by Klingsheim et al. [2007] as the most popular smartphone development platform but it is not found on all smartphones. An example is the Apple iPhone Software Development Kit (SDK) which is scheduled for release in February 2008 and will support native applications only [Jobs, 2007]. Java ME applications are not native applications

because they run on a Java Virtual Machine (JVM) [Li and Knudsen, 2005].

The difficulty of generating results that apply to all smartphones requires the selection of an alternative approach that also allows conclusions to be drawn about the mobile web service field in general. It is possible to provide results that are cross-platform. The term cross-platform is used here to refer to results that apply across multiple operating systems as opposed to across smartphone platforms. Such results allow general conclusions to be drawn about the mobile web services field because they do not refer to a single device or operating system manufacturer. These results are generated using the method described in the following section.

## 5.3 Method

The analysis of smartphone APIs is carried out by implementing the test configuration with existing APIs. The first experiment objective of analysing the state of current APIs, limits the experiment to developing new APIs only when they facilitate the further analysis of existing APIs. The development of new APIs is further limited to those parts of the configuration listed under the configuration's device constraints. This limitation is necessary in order to focus on those aspects of the configuration that influence interoperability.

The absence of the configuration parts specified under network constraints does not have an impact on a mobile requester's ability to communicate with an end-to-end secured provider. The absence of MTOM support decreases the practicality of a WS-Security-secured mobile web services transaction because of the increased message sizes resulting from WS-Security, as was discussed in section 4.4.2. However, a requester may still interact with an end-to-end provider without MTOM.

In contrast, the absence of the configuration components specified under the device constraints prevents a requester from interacting with an end-to-end provider. The parts specified under the devices constraints are considered to be critical. To this end, an implementation of these critical components is conducted when existing components are found to be inadequate or do not exist. MTOM support is not implemented because it does not aid in determining whether a requester may interact with an end-to-end secured provider. The rest of the method described in this section details the approach taken in the investigation of critical aspects of the configuration.

### 5.3.1 Top-Down Analysis of APIs

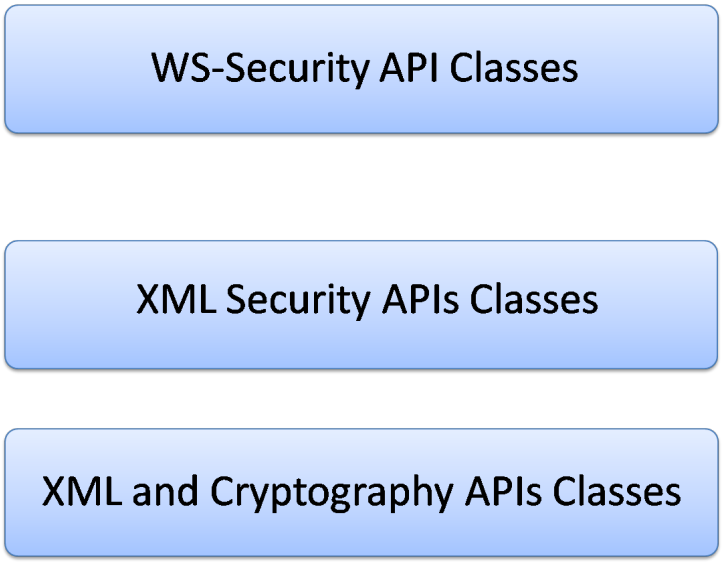


Figure 5.1: Hierarchy of API categories under investigation.

Figure 5.1 identifies the types of smartphone API that work towards the implemention of the mobile WS-Security configuration. They are presented in a hierarchy based on the discussion of WS-Security in section 3.6. This discussion shows that WS-Security is built on XML security specifications such as XML Encryption. These XML security specifications are in turn built with XML processing and cryptography APIs.

The hierarchical model, shown in figure 5.1, facilitates a top-down analysis of smartphone APIs that play a role in the provision of interoperable, end-to-end, mobile web services security. WS-Security APIs are investigated first. New WS-Security APIs are built with XML security APIs when existing WS-Security APIs are inadequate or absent. New XML security APIs are built with base XML and cryptography APIs where existing XML security APIs are also inadequate or missing. This top-down analysis ends when XML and cryptography APIs are absent or inadequate to build XML security APIs. These base APIs are not built because such an effort does not aid the analysis of any lower level APIs. This top-down analysis of smartphone APIs is carried out using the approach described in the following section.

### 5.3.2 Practical Approach to the Analysis

The top-down analysis introduced in the previous section is driven by whether the APIs in a particular category adequately fulfil their role in implementing the mobile WS-Security configu-

ration. APIs existing at a lower level of the hierarchy are analysed only if the APIs at the current layer under investigation are incapable of fulfilling their role. The definition of these roles enables the analysis to take place. The roles are identified through a mapping of the API categories to the requirements of the mobile WS-Security configuration.

#### 5.3.2.1 WS-Security API

WS-Security APIs must coordinate the XML security APIs to apply the mobile WS-Security configuration mechanisms to a SOAP message. The XML security APIs may expose independent functionality, for example one library may sign XML and another may encrypt XML. It is the responsibility of the WS-Security API to combine such functionality to secure a SOAP message.

#### 5.3.2.2 XML Security API

The XML security APIs must directly implement the mobile WS-Security configuration mechanisms that meet the challenges of confidentiality, integrity , authentication and message uniqueness. Confidentiality must be met through the asymmetric encryption of symmetric keys and the symmetric encryption of XML according to XML Encryption. The challenge of integrity must be met by symmetric signature generation according to XML Signature.

The XML security APIs must insert WS-Security tokens into a SOAP message. The mobile WS-Security configuration dictates that the username token must be used to meet the challenges of authentication and message uniqueness. XML security APIs must support the creation and insertion of this token into a SOAP message. The failure of existing XML security APIs to meet the challenges of confidentiality, integrity, authentication and message uniqueness as dictated by the mobile WS-Security configuration, requires new XML security APIs to be constructed from XML and cryptography APIs.

#### 5.3.2.3 XML and Cryptography API

The XML APIs must enable the XML security APIs to insert WS-Security tokens into SOAP messages and transform XML into a format upon which cryptography operations may act. The cryptography APIs must provide the symmetric and asymmetric cryptography algorithms required by the XML security APIs. The symmetric AES and asymmetric RSA encryption algorithms are selected for the encryption carried out during the experiment because they are considered de facto industry standards [Ferguson and Schneier, 2003]. The adoption of these algorithms within industry means that they are likely to be supported by a secure provider.

The HMAC-SHA1 symmetric algorithm is used to generate signatures in the experiment. The HMAC-SHA1 algorithm is selected because HMAC must be carried out on platforms that support the TLS protocol [Dierks and Rescorla, 2006]. A secure provider is reasonably expected to implement the HMAC-SHA1 algorithm because TLS is the most popular mechanism deployed to secure web services. This popularity means that a secure provider will at least support TLS and HMAC [Kearney, 2005].

The end-to-end security that is enabled by these cryptography algorithms is necessitated by the possible existence of intermediaries. The reason why intermediaries are not implemented in the experiment, forms the final topic of discussion concerning the experiment method.

### 5.3.3 The Exclusion of Intermediaries from the Experiment

The implementation of the mobile WS-Security configuration is carried out on mobile requesters and a traditional provider only. Intermediaries are not deployed because this does not add value to the research presented. The implications that intermediaries have on end-to-end security are discussed in section 3.5.1.1 and WS-Security is selected because it deals with these issues. The deployment of the intermediaries within the experiment would serve only to prove that WS-Security is appropriate for securing a web services transaction with intermediaries along the message path. This has been established in section 3.6 and is out of the scope of the discussion because the experiment is concerned with whether a mobile requester may interact with an end-to-end secured provider.

## 5.4 Implementation of the Mobile Requester

This section introduces the mobile platforms utilised in the experiment. Each of these platforms introduce implementation-specific issues that are also considered in this section. The hardware devices used in the experiment are also introduced here.

### 5.4.1 Selection of Platforms

The Java ME [Li and Knudsen, 2005] and .NET Compact Framework (CF) [Fox and Box, 2004] are selected as the smartphone platforms for the implementation carried out in the experiment. These platforms are selected because they meet the objective of providing widely applicable results. Java ME is available across multiple smartphone operating systems including Symbian, Windows Mobile, Palm OS, Linux and the proprietary RIM OS that runs on Blackberry devices

[Chang and Chen, 2005]. Statistics from 2005 show that there are over 86 Java ME licensees and one billion mobile devices comprising of over 600 different models that support Java ME [Teder, 2006]. The results obtained with Java ME are consistent with the second implementation objective of providing cross-platform results.

Microsoft's .NET CF may be considered as a direct competitor to Java ME because of its similarities to Java ME [Teder, 2006]. .NET CF applications run on a smartphone's operating system via a "virtual machine" in a similar fashion to Java ME applications [Fox and Box, 2004] [Li and Knudsen, 2005]. The .NET "virtual machine" is referred to as the Common Language Runtime (CLR) and any operating system that has a CLR running on it may run .NET CF applications.

Despite this potential, Microsoft provides a .NET CF CLR for Microsoft smartphone operating systems only [Siegemund et al., 2006]. These operating systems accounted for 11% of the smartphone operating system market in 2006 [Canalys, 2007]. There is ongoing work to provide CLRs for other smartphone operating systems such as Net60 for Symbian [Red Five Labs, 2007]. Net60 runs .NET CF 1.0 applications on the Symbian operating system. The Symbian operating system accounted for 67 % of the smartphone operating system market in 2006 [Canalys, 2007]. Such efforts to port CLRs suggest that the potential for .NET CF applications to run across multiple operating systems may be realised in the future. It is partly for this potential that the .NET CF is selected. Although the results generated in this thesis mostly apply to the Windows Mobile smartphones, they may apply widely in near future, for example when the final version of Net60 is launched.

The other reason the .NET CF is selected is that it allows the results obtained from the Java ME platform to be compared with results from a similar platform. This comparison provides context to the results generated. When both platforms share a shortcoming, it may be considered as a problem for the mobile web services field in general. When a shortcoming is evident in only one platform then it may be considered as a platform specific problem as opposed to one that affects the mobile web services field in general.

The Symbian operating system hosts programmable platforms that provide mobile web services, for example the S60 platform C++ APIs [Hirsch et al., 2006]. It is not selected for the experiment and this decision requires explanation because it is the dominant smartphone operating system [Canalys, 2007]. The generation of results on a platform like S60, although not cross-platform, would apply to the majority of smartphones on the market because of Symbian's market dominance.

The generation of such results is avoided because they apply to one type of operating system only and in the case of S60, mostly to the device manufacturer Nokia. This contradicts the

implementation objective of providing cross-platform results. Implementation-specific details of the Java ME and the .NET CF with regard to their use in the experiment are provided in the following sub-sections.

### 5.4.2 Java ME Platform Considerations

Applications for the Java ME platform are developed with the Java language and the platform is split into configurations, profiles and optional APIs [Li and Knudsen, 2005]. Configurations specify a Java Virtual Machine (JVM) for a set of devices and a base set of APIs common to these devices. Section 4.2 mentioned that smartphones fit into the set of devices covered by the CLDC.

Java ME profiles are layered on top of configurations and they provide APIs for developing Java ME applications [Li and Knudsen, 2005]. The Mobile Information Device Profile (MIDP) is a profile that runs on top of the CLDC and the applications that are developed with MIDP are called MIDlets. The MIDP APIs are considered to be the minimum set of capabilities that "achieve broad portability and successful deployments" [JSR 118 Expert Group, 2006].

Optional APIs provide functionality that is not included in profile APIs [Li and Knudsen, 2005]. The J2ME Web Services Specification API (JWSA) and the Security and Trust Services API for J2ME (SATSA) are optional APIs of interest to this thesis [Ellis and Young, 2004] [JSR 177 Expert Group, 2004].

The Java ME requester is implemented as a MIDlet developed with MIDP 2.0 [JSR 118 Expert Group, 2006]. MIDP 2.0 is selected because it specifies the latest set of MIDP APIs. MIDP 2.1 is a maintenance release of MIDP 2.0 and it does not specify any new APIs. It provides clarification on the MIDP 2.0 APIs to aid the API fragmentation prevention effort of the Mobile Service Architecture (MSA) [Sharp, 2007] [JSR 248 Expert Group, 2006]. The MSA is a continuation of previous efforts to prevent the fragmentation of the Java ME platform caused by optional APIs [Teder, 2006]. The differences between MIDP 2.0 and MIDP 2.1 are not considered relevant to this thesis.

### 5.4.3 .NET Compact Framework Platform Considerations

.NET CF applications may be developed with the C# or Visual Basic languages [Janecek and Hlavacs, 2005]. C# is chosen as the language for developing the .NET CF requester because the author is more familiar with it than he is with Visual Basic.

The .NET CF provides a uniform set of APIs through its class libraries [Fox and Box, 2004].

This is unlike the Java ME platform which has API fragmentation problems due to the variation in configuration, profile and supported optional APIs. This difference is demonstrated by the mobile web services which are provided by default on the .NET CF but as optional APIs on the Java ME platform [Teder, 2006]. However, some of the class libraries are not guaranteed to work across all Microsoft operating systems on which the .NET CF runs [Wilson, 2003]. An example is the `InputPanel` .NET CF class which is used to implement an on-screen keyboard for devices running the Microsoft Pocket PC operating system. This class throws an error when it is instantiated on a CLR running on the Microsoft Windows CE operating system.

The .NET CF requester is built on .NET CF version 2.0 and version 1.0 [Wilson, 2005]. Version 2.0 is, at the time of writing, the latest stable version of the .NET CF offered by Microsoft. The .NET CF version 1.0 requester is run on the Net60 platform because Red Five Labs have implemented this version with Net60 [Red Five Labs, 2007].

### 5.4.4 Mobile Hardware

The Nokia N80 smartphone is used as the device on which the Java ME part of the experiment is carried out. The N80 has a dual ARM central processing unit (CPU) with a clock speed of 220 MHz and 64 MB of SDRAM memory [Nokia Corporation, 2007]. This smartphone is selected because it supports MIDP 2.0, JWSA and SATSA. The Net60 platform is also installed on the smartphone to demonstrate the applicability of the .NET CF results on a non-Microsoft operating system.

The device chosen for the .NET CF experiment is an iMate Jamin. The Jamin has a OMAP 850 processor with a clock speed of 200Mhz, 64 MB of RAM and 128 MB of ROM. It is chosen because it runs Windows Mobile 5.0 which supports both version 1.0 and 2.0 of the .NET CF [Wilson, 2005]. Windows Mobile 5.0 was the latest Windows Mobile version at the time of conducting the experiment, although version 6 is now available for some devices.

The N80 and Jamin have WLAN capabilities and all communication with the provider during the experiment is done via a WLAN. The use of a WLAN as opposed to an operator network, such as GSM, does not influence the results of the experiment because the SOAP protocol is agnostic of the lower level protocols that transport it, as mentioned in section 2.4.1. The WLAN was selected for the experiment because the author could excersize some control over its availability and did not need to pay for its use. The implementation of the provider is discussed in the following section.

## 5.5 Implementation of the Traditional Provider

A secure calculator web service provider is implemented with the Java language and is deployed on the GlassFish Application Server [Sun Microsystems, 2007i]. The open source GlassFish server provides the code on which the production quality Sun Java System Application Server is based [Sun Microsystems, 2007g]. The Sun Java System Application Server is not used in the experiment because it did not contain the Metro web services stack at the time of conducting the experiment [Sun Microsystems, 2007j]. Metro is available on GlassFish version 2 and it consists of the Java Architecture for XML Binding (JAXB) API [Ramesh Nagappan et al., 2003], the Java API for XML-Based Web Services (JAX-WS) [Kohlert and Gupta, 2007] and WSIT [Sun Microsystems, 2007h].

WSIT is the component of Metro that assists the experiment because it provides the functionality to secure a web service and apply MTOM in a manner that is interoperable between Java and .NET implementations [Sun Microsystems, 2007h]. This technology is favourable for the experiment because one of the mobile requesters is implemented on the .NET CF platform. Interoperability problems may arise from the differences between the Java and .NET platforms, for example one platform may map a data type into XML in a manner not understood by the other platform [Ye, 2004]. WSIT addresses these platform interoperability concerns and eliminates the need to consider them in the experiment results. Interoperability problems unearthed by the .NET CF part of the experiment may be isolated to shortcomings of the mobile requester platform as opposed to fundamental interoperability problems between the .NET and Java platforms.

Securing the calculator web service with the configuration presented in section 4.7 is simplified by an example provided in the WSIT tutorial that demonstrates how the mechanisms in the configuration may be applied [Sun Microsystems, 2007h]. The rest of this chapter details the testing carried out in the experiment. The .NET CF part of the tests are presented first because they yield the most success.

## 5.6 Microsoft .NET Compact Framework Tests

This section details the testing done on the Microsoft .NET CF version 2.0. A .NET CF calculator application that prompts a user for two integers and displays the result of their addition is built. This application sends the two integers to a provider via a .NET CF SOAP proxy object [Fox and Box, 2004]. This proxy object inserts the integers into a SOAP message and sends a message to the provider. The proxy object waits for a response message and the result carried in the

response message is passed back to the .NET CF application. The application and the proxy object with which it interfaces constitute the .NET CF requester. The proxy object is of interest to the experiment because it is responsible for creating SOAP messages and is therefore best placed to secure them.

The standard API that provides WS-Security on the .NET Framework is the Microsoft Web Services Enhancements (WSE) library. The use of this library on the .NET CF is considered first.

### 5.6.1 Microsoft Web Services Enhancements (WSE)

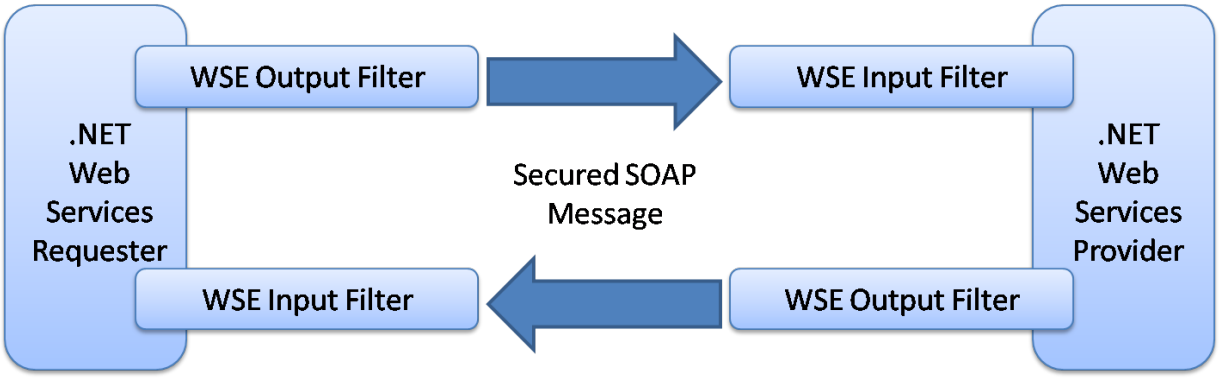


Figure 5.2: Web Services Enhancements architecture, after Microsoft Corporation [2007c].

The WSE provides extra web services functionality including WS-Security for .NET web services [Microsoft Corporation, 2007b]. The WSE adds value to SOAP messages by applying a set of filters on the messages as they leave and arrive at a web services entity, as shown in figure 5.2 [Microsoft Corporation, 2007c]. The filter approach requires the requester proxy class to be modified but leaves the requester .NET application unchanged.

Fox and Box [2004] state that the WSE is unavailable for .NET CF 1.0 and the Microsoft Developer Network (MSDN) forums indicate that it is also unavailable for the .NET CF 2.0 [MSDN Forums, 2006]. In the light of this absence, third party WS-Security APIs are sought. The MSDN Forums point to the Smart Device Framework (SDF) by OpenNetCF Consulting [2007].

The SDF provides the functionality of the WSE 2.0 on the .NET CF. These libraries are contributed by Casey Chesnut [Chesnut, 2004]. Fox and Box [2004] also reference his libraries when discussing WSE implementations on the .NET CF. The latest edition of the WSE is version 3.0 and the author finds no evidence that the SDF implements this version. However, the author could find no other publicly available libraries that attempt to implement WSE or WS-Security

on the .NET CF. An attempt to implement the mobile WS-Security configuration with the SDF is carried out for this reason.

### 5.6.2 The Smart Device Framework

The SDF provides functionality that is missing from the Microsoft .NET CF class libraries but is demanded by developers [OpenNetCF Consulting, 2007]. An example of such functionality is that of the WSE.

The SDF documentation is poor. The online SDF API reference names the SDF classes but does not provide any further description of them, for example the names and details of the public class members. It may be assumed that the WSE documentation by Microsoft applies to the SDF because the SDF is an implementation of the WSE. However, the WSE names its classes differently from those of the SDF and this complicates any attempt to use the WSE API documentation to understand the SDF.

Examples demonstrating how to use the SDF to implement WS-Security are also limited. The only such demonstration found by the author is by Chesnut [2004]. This demonstration shows that WS-Security is implemented with the SDF in a different manner from that in which it would be implemented with the WSE. The WSE provides a new class from which SOAP proxy classes must inherit [Microsoft Corporation, 2007c]. .NET requesters usually derive their proxy classes from the `System.Web.Services.SoapHttpClientProtocol` class but the proxy of a requester using the WSE must inherit from the `Microsoft.Web.Services2.WebServicesClientProtocol` class. Requester WSE filters are applied using the methods of this proxy base class. The WSE uses SOAP extensions to apply filters for the provider. A SOAP extension is a .NET Framework class that provides access to incoming and outgoing SOAP messages before and after they are serialised [Shepherd, 2004]. The SDF differs from the WSE by using SOAP extensions for securing requester SOAP messages instead of a new proxy base class.

The lack of documentation and examples of the SDF's WS-Security functionality requires some effort to understanding how the SDF works to provide WS-Security. This understanding allows a more critical analysis to be conducted in the investigation of the SDF's capability to provide interoperable, end-to-end, mobile web services security.

The SDF version 2.1 is the latest edition of the libraries and its commercial version ships with its source code. This source code and the examples from Chesnut [2004] were used to gain some insight into the SDF.

#### 5.6.2.1 SDF WS-Security Classes

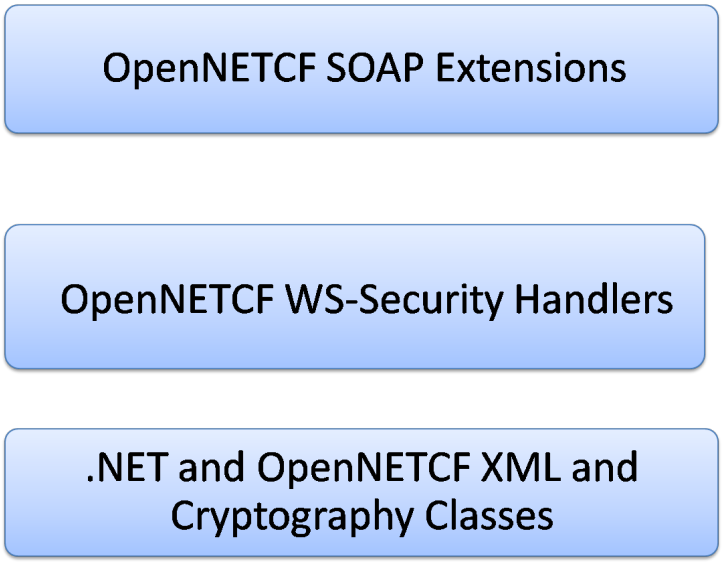


Figure 5.3: WS-Security components of the SDF.

Figure 5.3 maps the SDF WS-Security related classes to the API hierarchy presented in section 5.3.2. The SDF exposes WS-Security API functionality with SOAP extensions. These SOAP extensions call XML security handler classes to encrypt, sign and add WS-Security tokens to a SOAP message. The SDF uses the .NET CF and its own XML processing classes to modify the SOAP messages. The SDF provides cryptography classes that the .NET CF does not provide, for example the HMAC-SHA1 algorithm.

The mapping presented in figure 5.3 demonstrates that the SOAP extension classes coordinate the actions of lower level handler classes that secure a SOAP message directly. The main XML security handler classes provided by the SDF are: the `HeadersHandler` class, `XmlSigHandler` class and `XmlEncHandler` class.

The `HeadersHandler` class adds SOAP headers that do not represent a WS-Security token or carry information for the XML Encryption and XML Signature processes. Examples of these headers include WS-Addressing elements such as the `<To>` element [Gudgin et al., 2006]. The WS-Addressing specification details how transport-related information is placed inside a SOAP envelope such that a SOAP message is not reliant on transport protocols to deliver this information.

The `XmlEncHandler` class adds WS-Security tokens to the message and encrypts SOAP message parts. The `XMLEncHandler` class is instructed on what tokens to add, what message

parts to encrypt and how to encrypt the parts by an `XMLEncObject` class. The `XMLEncObject` class contains members representing the tokens that must be added to a message and the parameters for the XML Encryption process that is carried out by the `XMLEncHandler` class. Examples of such parameters include the message parts that must be encrypted, encryption algorithms and encryption keys.

The `XmlSigHandler` class signs SOAP message parts and it is similar to the `XMLEncHandler` class because it may also add WS-Security tokens to a SOAP message. The `XmlSigHandler` class is configured by a `XMLSigObject` that is similar to the `XMLEncObject`. The `XMLSigObject` specifies tokens the must be added to a message and contains members that allow for the setting of parameters for the XML Signature process carried out by the `XMLSigHandler` class. Examples of these parameters include the message parts to be signed, signature generation algorithms and signing keys. Such interactions between these XML security classes and the SOAP extensions, when implementing the mobile WS-Security configuration, are detailed in the following discussion.

#### 5.6.2.2 The Mobile WS-Security Configuration with the SDF

The sequence diagram shown in figure 5.4 on the next page illustrates how the SDF is used to apply the mobile WS-Security configuration. This sequence diagram, like others that follow, does not strictly adhere to the Unified Modelling Language (UML) [Booch, 2000]. The interactions between the classes shown in the sequence diagram are not represented as method calls but as textual descriptions of the interactions. This deviation from UML is taken because it is the author's intent to describe these interactions rather than model the design of the SDF. The sequence diagram simplifies a real world web services transaction because it considers the securing of the request message only and not the handling of a secure response message. However, this simplification does explain how the SDF is used to implement the mobile WS-Security configuration.

The configuration mechanisms are set through the relevant members of the `XMLEncObject` and `XMLSigObject` classes by the .NET CF application. An example of how the symmetric AES algorithm parameter for the XML Encryption process is set up, through the `XMLEncObject`, is shown in the following snippet:

```
OpenNETCF.Web.Services2.XmlEncObject xeo
= new OpenNETCF.Web.Services2.XmlEncObject();
xeo.SymmAlg = new AES128();
xeo.SymmAlg.GenerateKey();
```

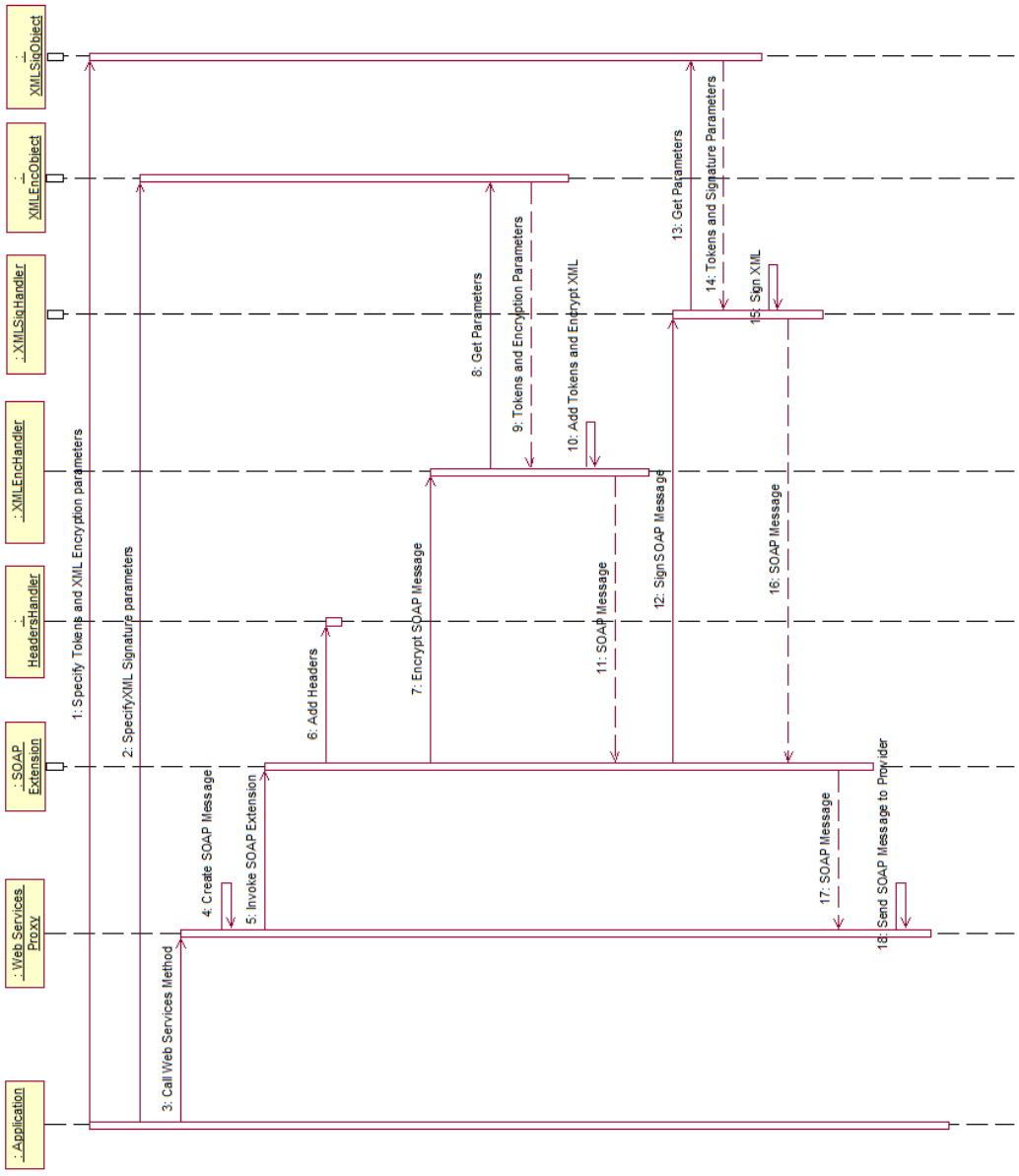


Figure 5.4: Sequence diagram of SDF class interactions.

The .NET CF application calls the proxy object's web service call method. This method is annotated with C# attributes that apply the the `WsExtension` and `HeadersExtension` SOAP extension classes provided by the SDF.

```
[OpenNETCF.Web.Services2.HeadersHandler()]
[OpenNETCF.Web.Services2.WsExtension()]
public new int add...
{
    object[] results = this.Invoke("add",...);
    return ((int)(results[0]));
}
```

The `HeadersExtension` SOAP extension calls the `HeadersHandler` class to add SOAP headers. The `WsExtension` SOAP extension invokes the `XMLEncHandler` class and then the `XMLSigHandler` class. Invoking the classes in this order encrypts and then signs the encrypted message. Figure 5.4 on the preceding page illustrates that the `XMLEncHandler` class and `XMLSigHandler` classes get the instructions of how to process the message from the `XMLEncObject` and `XMLSigObject` respectively. An example code snippet of this interaction cannot be reproduced here because the software license employed by OpenNetCF Consulting [2007] does not allow this. The `XMLSigHandler` returns the message to the `WsExtension` SOAP extension which returns the secured message to the proxy object. The proxy object sends the message to the provider. The implementation of the mobile WS-Security configuration with the SDF fails. The following shortcomings are identified as reasons for this failure.

### 5.6.3 Shortcomings of the SDF

The mobile WS-Security configuration cannot be implemented with the SDF classes. The shortcomings of the SDF in this regard are found in the configuration sections that deal with network constraints, confidentiality and integrity.

#### 5.6.3.1 Network Constraints

The SDF does not support MTOM. This is expected because MTOM is supported as of WSE 3.0 and the SDF is an implementation of WSE 2.0 [Microsoft Corporation, 2005]. This is not a serious shortcoming when interoperability is concerned because it is possible for a mobile

.NET CF requester that does not support MTOM to interoperate with a secured provider. The following shortcomings are more critical because they affect the ability of a requester to interact with a secured provider.

#### 5.6.3.2 Confidentiality

The SDF supports the sharing of a symmetric key by using the EncryptedKey mechanism specified by WS-Security but it does not provide the functionality required to encrypt `<wsse:Security>` header elements. This affects WS-Security tokens such as the username token which are carried in the clear as a result of this shortcoming.

#### 5.6.3.3 Integrity

Although the SDF does not support the encryption of `<wsse:Security>` header elements, it supports the signing of some of these elements. The noticeable exception found during the experiment is the lack of support for signing the username token. The SDF does not provide means to specify that this token must be signed when it is attached.

The second integrity shortcoming found is the inability of the SDF to create signatures with the same key that is used for encryption. The SDF supports symmetric signature generation with derived keys only and it possesses the means to derive a key from the username token as specified by *WS-Security Username Token Profile* [Nadalin et al., 2006d]. This provides an interoperable alternative to the creation of signatures with a symmetric key. This approach is frowned upon by the *WS-Security Username Token Profile* because a user may select a weak password, allowing the a malicious entity to guess the password and trivially derive the key [Nadalin et al., 2006d]. It is for this reason that the SDF implementation of symmetric key signature generation is rejected. The SDF inability to use the symmetric encryption key to sign XML is considered a shortcoming.

#### 5.6.3.4 Summary of SDF Shortcomings

The failed implementation of the mobile WS-Security configuration with the SDF forms the analysis carried out at the WS-Security API layer shown in figure 5.1. The top-down analysis approach requires that a WS-Security API is built with the XML security APIs when existing WS-Security APIs cannot implement the configuration. However, the shortcomings that prevent the WS-Security APIs from implementing the configuration are inherent to the underlying XML handler classes that exist at the XML security API layer. An example is the inability to use the same key for encryption and signature generation which is a failing of the `XMLSigHandler`

class. The SDF XML security API classes are modified to build a WS-Security API class that implements the configuration. This class is in the form of a custom SOAP extension named `GFExtension`. This class leverages the modified SDF XML security API classes.

### 5.6.4 Modification of the SDF XML Security Classes

The classes that exist at the XML security API layer are modified to enable the construction of a WS-Security API that supports the implementation of the mobile WS-Security configuration. The modifications form the analysis carried out at the XML security API layer shown in figure 5.1. These modifications are built into new C# classes that inherit from the SDF `XMLEncObject`, `XMLSigObject`, `XMLEncHandler` and `XMLSigHandler` classes. These new classes represent improvements to the SDF and not a re-implementation per se. The changes introduced in these classes overcome the shortcomings of the SDF detailed in this section. The SDF license does not allow modifications to the source code of its latest version to be redistributed although some of the code is taken from open source projects. The SDF XML canonicaliser is taken from the open source Mono Project [Mono Project Community, 2007]. Source code snippets of the modifications are not provided because of these licensing constraints.

#### 5.6.4.1 Confidentiality

Two new classes are created to overcome the confidentiality shortcomings of the SDF: the `GFXMLEncObject` class that inherits from the `XMLEncObject` class and the `GFXMLEncHandler` class that inherits from the `XMLEncHandler` class. The string "GF" is prepended to the parent class names to indicate that the new classes are intended to enable interoperability with a web service running on a GlassFish server. This naming convention applies to the rest of the modified SDF classes.

The SDF's failure to apply XML Encryption to `<wsse:Security>` header elements is traced to the `XMLEncObject` class. The `XMLEncObject` class contains a member variable that names one message part that must be encrypted. This member is always set to name the SOAP body. The `GFXMLEncObject` class provides a new member that allows for the `<wsse:UsernameToken>` element to be specified as a target for the XML Encryption process.

The `XMLEncHandler` class is hard coded to interact with the `XMLEncObject` class and there is no way to instruct it to use the `GFXMLEncObject` class. Even if this were possible, the `XMLEncHandler` cannot create an `<xenc:EncryptedData>` element other than the one

that represents the encrypted SOAP body. The GFXMLEncHandler class is modified such that other <xenc:EncryptedData> elements may be created. These changes allow the encryption of the username token and the insertion of the <xenc:EncryptedData> element that represents the encrypted username token into the <wsse:Security> header. Two classes are also modified to rectify the integrity shortcomings of the SDF.

#### 5.6.4.2 Integrity

The GFXMLSigObject and GFXMLSigHandler classes inherit from the SDF XMLSigObject and XMLSigHandler classes respectively. The SDF cannot sign a SOAP message with the same symmetric key used for encryption because the XMLSigHandler class does not interface with the XMLEncObject class. Such an interface provides the XMLSigHandler class access to the symmetric key referenced by the XMLEncObject class. The GFXMLSigObject class provides a member that references the GFXMLEncObject class. This allows the GFXMLSigHandler class to access the symmetric key referenced by the GFXMLSigObject class. The GFXMLSigHandler class is further modified to use this symmetric key with the HMAC-SHA1 algorithm for the signature generation and signature verification processes. The generation of the <ds:KeyInfo> element is modified such that it is able to reference the <xenc:EncryptedKey> element that holds the encrypted symmetric key.

#### 5.6.4.3 Result of modifications

The modifications detailed in this section allow a .NET CF requester to participate in a transaction with a provider secured with the mobile WS-Security configuration less MTOM. These modifications should in theory apply to all .NET CF platforms that support the SDF. This assertion is tested on the Net60 platform.

### 5.6.5 Red Five Labs Net60 Testing

The SDF 1.4 is used for testing on the Net60 platform because it is compatible with the .NET CF 1.0 [Struys, 2006]. SDF editions after version 2.0 are compliant with the .NET CF 2.0 only. The requester implemented in the previous section is back-ported to the .NET CF 1.0 and then modified to use SDF 1.4 classes. The SDF 1.4 XML security API classes are changed in the same way that the SDF 2.1 XML security API classes were modified.

Most of the changes carried out in this porting exercise are related to the cryptography API classes. The .NET CF 2.0 provides cryptography support but this support is missing on the

.NET CF 1.0 [Wilson, 2005]. These cryptography libraries are provided by the SDF 1.4 and the requester is modified to utilise these libraries. However, the SDF 1.4 does not support the AES symmetric encryption algorithm and this is problematic because the experiment method described in section 5.3.2 dictates the use of the AES algorithm. This lack of support by the SDF 1.4 requires a compromise to allow the testing on Net60 to proceed and the Triple DES symmetric algorithm is selected because it is the predecessor to the AES algorithm [Schneier, 1996] [Ferguson and Schneier, 2003]. The other changes required during the porting effort are purely aesthetic, for example some SDF libraries appear in a different namespace in the SDF 1.4 to the one they are placed under in the SDF 2.1.

#### 5.6.5.1 Result of Net60 Testing

The port to Net60 fails for reasons that cannot be conclusively established in this thesis. The .NET CF application is loaded successfully and prompts the user for input but it returns an error code of "-5" when the web services method call is made with the SOAP proxy object. A network traffic capture shows that no outward traffic is generated by the requester running on Net60. On device debugging cannot be done with Net60 and this inhibits an attempt to isolate the problem within the requester code.

The SDF 1.4 requester runs successfully on the Microsoft .NET CF 1.0 CLR of the Jamin. An insecure version of the requester is run on Net60 and it successfully interacts with an insecure provider. These two successful runs suggest that the application of security mechanisms to the SOAP message is the cause of the failure to implement the mobile WS-Security configuration on Net60. Cryptography operations are suspected to be the particular source of the failure because some .NET cryptography operations are called from the operating system on which the CLR runs [Freeman and Jones, 2003].

#### 5.6.5.2 Possible Explanation for the Net60 Failure

The Platform Invoke (P/Invoke) feature of the .NET CF allows .NET CF applications running on a CLR to access the libraries of the underlying operating system [Janecek and Hlavacs, 2005]. The operating system libraries are considered to be unmanaged code because they run directly on the hardware of the smartphone. .NET CF code is considered to be managed code because it runs on a CLR that controls its access to the hardware. Support for P/Invoke is possible because Microsoft manufactures CLRs for their operating systems. This allows Microsoft to control how the unmanaged code of the operating system is called by the managed .NET CF code [Janecek and Hlavacs, 2005].

The .NET CF cryptographic operations that are called from the underlying operating system are provided by .NET classes that have the text "`CryptoServiceProvider`" appended to their names [Freeman and Jones, 2003]. The `RSACryptoServiceProvider` class is an example of such classes and it is used in the `GFXMLSigHandler` class to encrypt the symmetric key with the RSA algorithm. The .NET CF 2.0 and SDF 1.4 versions of this class invoke the RSA cryptography functionality found in the operating system on which the CLR runs. It is suspected that such cryptography operations cause Net60 to fail because they call the functionality of the Windows Mobile operating system, but Net60 runs on Symbian. To this end, Net60 may not be translating these calls appropriately to Symbian.

The explanation provided here for the failure of the Net60 test is speculative and cannot be conclusively backed up with any evidence. The test with Net60 may fail because of an implementation bug in Net60. The latest version of Net60 is, at the time of writing, in the release candidate stage and is expected to contain some pre-release bugs.

### 5.6.6 Summary of .NET CF Implementation Experiences

The modification of the SDF XML security API classes allow a requester to communicate with a provider when the mobile WS-Security configuration is applied. This demonstrates that interoperable end-to-end mobile web services security may be implemented on the .NET CF. However, this functionality is obfuscated by poor developer support for the SDF.

The lack of comprehensive documentation leaves the SDF source code as the main reference guide for the usage of the SDF. Comments are available in the source code but they are brief. There are also few examples available to developers that demonstrate the use of the SDF in securing a SOAP message. The only examples found are provided by the C# code on the web site of Chesnut [2004]. This code illustrates how a .NET CF application and proxy class may interact with the SDF classes to implement WS-Security .

The examples by Chesnut [2004] are of minimal assistance when the SDF libraries need to be modified to implement the mobile WS-Security configuration because they do not betray the inner workings of the SDF libraries. The SDF modifications may seem trivial but the lack of documentation, detailed source code commenting and examples adds to the complexity of understanding which parts of the SDF require modification. Gaining this understanding takes up most of the time spent on the .NET CF experiment. However, the examples by Chesnut [2004] are slightly helpful because they allow a developer to identify the SDF classes that interface with a requester and then drill down using the SDF source code of these classes. The second part of the experiment requires testing to be done on the Java ME platform.

## 5.7 Java ME Testing

Java ME web services are implemented with a proxy approach similar to that of the .NET CF [Ortiz, 2004]. The Java ME calculator web services requester consists of a MIDlet that prompts a user for input and a SOAP proxy that is responsible for the construction of SOAP messages. The proxy is referred to as a stub in Java Web services [Ortiz, 2004].

The .NET Framework and Java platforms adhere to the same web services design but differences are noticeable in the nature of the processes driving WS-Security on each of the platforms. The .NET Framework is characterised by a WS-Security implementation process determined by Microsoft. The SDF simply follows Microsoft's lead instead of pioneering its own effort to provide WS-Security; for example the SDF is a reimplementation of WSE and not a "new" WS-Security API.

The effort to provide WS-Security for Java is more varied. Independent efforts directed towards the provision of WS-Security exist. For example, the Apache Foundation is working on WS-Security through its Synapse project and the Metro Project is building WSIT on top of the XML and Web Services Security Project [Apache Software Foundation, 2007] [Sun Microsystems, 2007k]. The standardisation of XML Encryption and XML Signature Java APIs is different from that of the WSE because it is a community process carried out through JSR 105 and JSR 106 respectively [Nadalin and Mullan, 2005] [Nadalin, 2005]. Despite the availability of multiple WS-Security and XML Security Java APIs, the author finds none targeted for the Java ME platform.

### 5.7.1 WS-Security and MTOM on Java ME

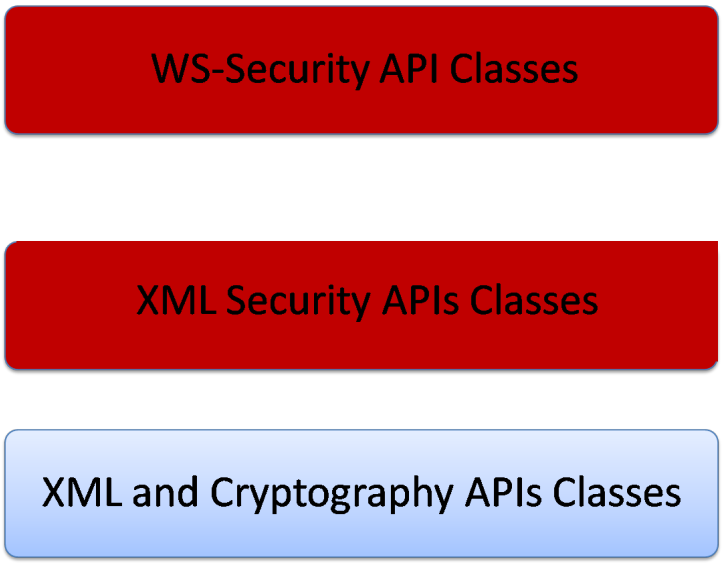


Figure 5.5: Missing Java ME APIs of the API hierarchy presented in figure 5.1.

The red blocks in Figure 5.5 show the Java ME API categories that are missing within the hierarchy presented in section 5.3.1. XML and cryptography APIs are available, for example kXML is an XML API and part of SATSA deals with cryptography. However, the XML and cryptography APIs are not leveraged to provide publicly available XML security APIs. Kangasharju [2007] suggests that his XAS-ext library provides WS-Security functionality on Java ME but this library is not yet publicly available.

No Java ME API is found to implement MTOM either. MTOM support is available on the Java Standard and Enterprise Edition through WSIT but WSIT is not available for Java ME [Sun Microsystems, 2007h].

An implementation of the WS-Security and XML security API classes is carried out but this implementation differs from that carried out on the .NET CF because there are no existing WS-Security and XML security API classes to modify. A fresh implementation of these APIs is required and the design that this implementation follows is heavily influenced by the design of the SDF.

### 5.7.2 Design

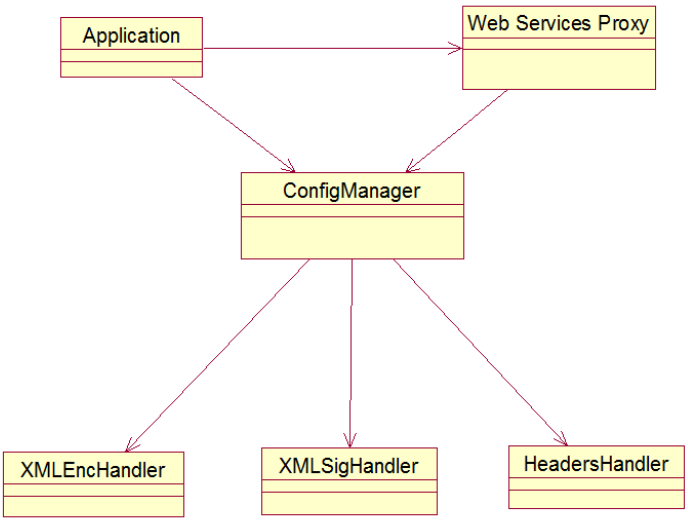


Figure 5.6: Class diagram of Java ME implementation components.

It is assumed that the SDF design can successfully provide WS-Security on the Java ME platform because it does so on the .NET CF platform. Some deviations from the SDF design are taken to avoid the shortcomings of the SDF detailed in section 5.6.3. Figure 5.6 presents a class diagram of the resulting Java ME design.

A single configuration class named `ConfigManager` is provided in place of the `XMLSigObject` and `XMLEncObject` of the SDF. This class is responsible for configuring the underlying XML security API classes named after their SDF counterparts as `HeadersHandler`, `XMLSigHandler` and `XMLEncHandler`. The sequence diagram in figure 5.7 on the next page details the interactions between these classes.

The `ConfigManager` class presents a single interface to the XML security APIs for the Java ME MIDlet and proxy class. The sequence diagram in figure 5.7 illustrates that this interface allows an application to set the security configuration to be applied to a SOAP message. A security configuration consists of a set of parameters that are needed to secure a SOAP message, for example a symmetric encryption key needed to encrypt SOAP message parts. The `ConfigManager` also allows preset security configurations to be selected. SOAP extension classes are .NET Framework specific and to this end a different approach to securing SOAP messages is needed on Java ME. Figure 5.7 shows that the application must extract the SOAP message from the proxy class before it is sent to the provider. The SOAP message must be sent

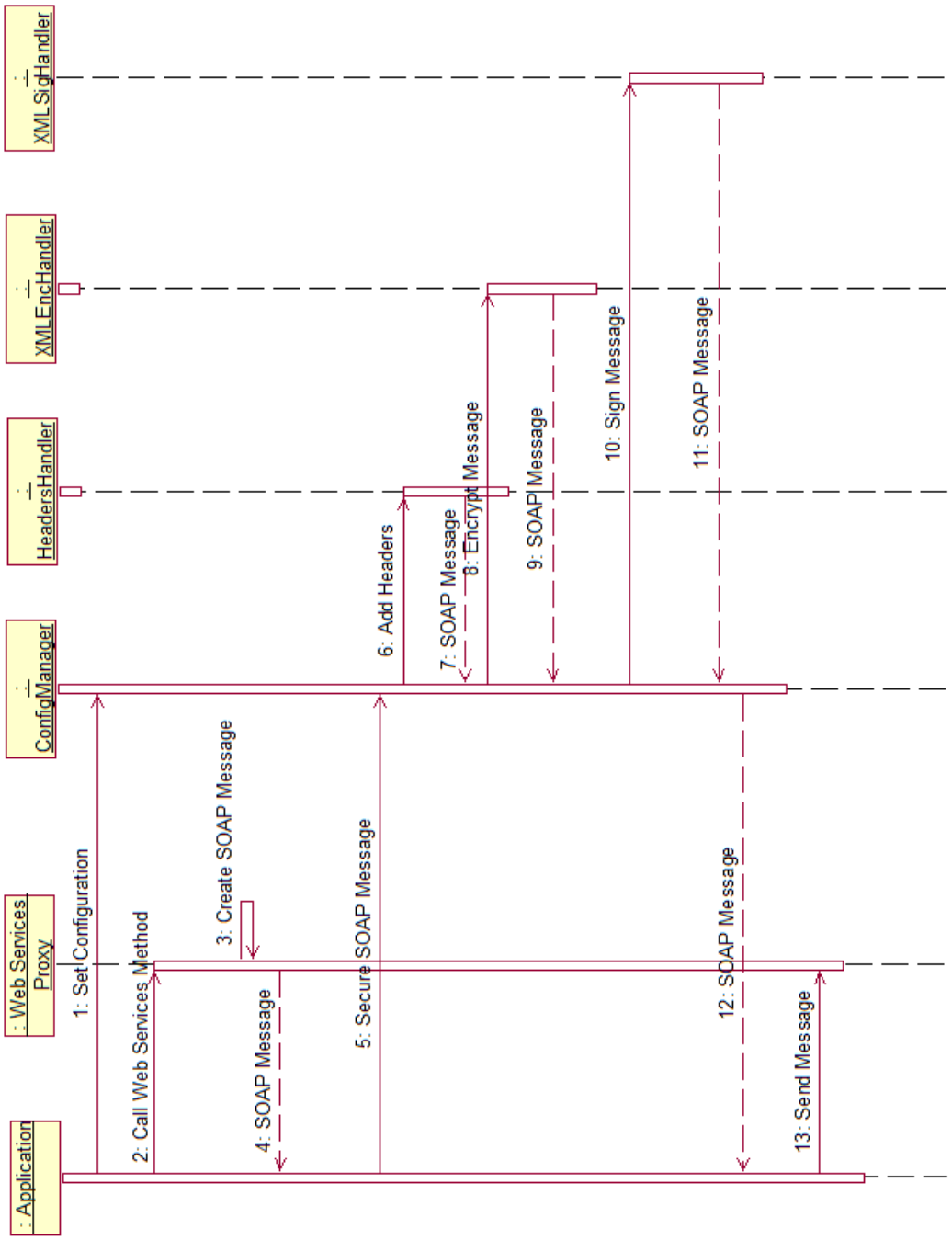


Figure 5.7: Sequence diagram of Java ME class interactions.

to the `ConfigManager` class to be secured and then returned to the proxy class. The proxy class sends the secure message to the provider at this point.

#### 5.7.2.1 Advantages of the design

The shortcomings of the SDF highlighted in section of 5.6.3 result from a design flaw that isolates WS-Security configuration information. The same symmetric key for example, cannot be used by the encryption and signature generation processes because the key is referenced by the `XMLEncObject` class which is inaccessible to the `XMLSigHandler` class. The `ConfigManager` class avoids this problem by providing a single location from which all WS-Security information may be accessed.

The `ConfigManager` class also presents a single entry point for securing a SOAP message. This provides a less complex interface between the XML security API classes and the MIDlet because only one object needs to be instantiated and set up. This also provides a more modular and extensible design as additional custom XML security API classes may access the security information exposed by the `ConfigManager` class. The implementation of this design with Java ME XML and cryptography APIs is detailed in the following section.

### 5.7.3 Implementation of the XML Security API

The implementation of the XML security API classes requires the selection of XML and cryptography APIs. The sequence diagram in figure 5.7 shows that the selection of the SOAP proxy for Java ME is also critical because the proxy must provide a facility to send a SOAP message to the MIDlet before it leaves for the provider. To this end, SOAP proxies are also analysed in the Java ME experiment. The implementation of the XML security API classes begins with an attempt to secure a SOAP message with the standard Java ME XML and cryptography APIs.

#### 5.7.3.1 JSR 172 and JSR 177

The JWSA and SATSA are standardised through the Java Community Process (JCP) as JSR 172 and JSR 177 respectively [Sun Microsystems, 2007a] [Ellis and Young, 2004] [JSR 177 Expert Group, 2004]. The JWSA is considered as an XML API and a SOAP proxy because it provides XML parsing functionality and creates SOAP messages on behalf of a MIDlet.

These APIs are optional and therefore likely to contradict the implementation objective of producing widely applicable results. The MSA includes support for JSR 172 and JSR 177 in its collection of specified APIs but this does not guarantee that both APIs will be widely deployed.

The MSA is a new effort and is yet to be widely adopted, as evidenced by Sun Microsystem's compilation of existing MSA-compliant devices, which lists only eight such devices as of December 2007 [Sun Microsystems, 2007e]. However, these APIs are considered in this thesis because they are standard and this makes them likely candidates to be shipped with smartphones that provide Java ME mobile web services. The implementation of the mobile WS-Security configuration with WSA and SATSA fails. The shortcomings of JWSA that contribute to this failure are considered first.

#### 5.7.3.1.1 JSR 172 Shortcomings

Siddiqui [2006] details how to secure SOAP messages with a combination of JWSA and SATSA. The security implementation presented is flawed by his own admission because JWSA does not support XML attributes. Attribute support is required by the XML Signature standard and the signatures generated by the implementation of Siddiqui [2006] lead to interoperability problems because a provider needs to be modified to understand the signatures.

The attempted implementation of the mobile WS-Security configuration exposes that JWSA does not provide any SOAP header support and this is a failing of the JWSA's SOAP proxy role. This shortcoming immediately disqualifies the JWSA as an API for providing WS-Security because the `<wsse:Security>` header cannot be created without header support. Shortcomings that contribute to the failure in implementing the mobile WS-Security configuration are also found in SATSA.

#### 5.7.3.1.2 JSR 177 Shortcomings

Two of the four SATSA packages provide a MIDlet access to the SIM and this makes it possible to use the SIM's cryptographic operations. This approach is avoided because it applies to smartphones that support the SIM only. Huang [2006] shows how security may be applied using SATSA and no SIM functionality but his article exposes three shortcomings that hinder the implementation of the mobile WS-Security configuration:

Huang [2006] firstly states that asymmetric encryption is not supported by SATSA without the SIM because no safe mechanism other than on the SIM exists to store private keys in Java ME. The author finds this reasoning limited because a safe mechanism to store symmetric keys other than on the SIM does not exist on Java ME, yet SATSA supports symmetric encryption. It is also contended that symmetric keys require a safe storage mechanism more than asymmetric public keys because asymmetric public keys are meant to be publicly available but symmetric keys must be kept secret [Ferguson and Schneier, 2003]. However, the limited reasoning by

Huang [2006] does not change the fact that asymmetric encryption support is unavailable on Java ME.

The second shortcoming of SATSA is the lack of a means to generate a symmetric key. Huang [2006] shows how to derive keys by encrypting the IMEI with an existing symmetric key and using the resulting cipher-text as a derived key. However Huang [2006] does not show how the existing key is generated. An attempted implementation of the mobile WS-Security configuration lead to the conclusion that a symmetric key cannot be generated with SATSA.

A third shortcoming of SATSA, exposed in the article by Huang [2006], is the lack of support for MAC generation. This prevents the creation of a digital signature with a symmetric key, for example with the HMAC-SHA1 algorithm. The implementation of XML Signature with SATSA is impossible without using the SIM because SATSA delegates MAC generation and asymmetric cryptography operations to the SIM.

Despite these shortcomings SATSA is not entirely rejected because it is the standard Java cryptography API and it supports symmetric encryption without appealing to SIM functionality. SATSA is therefore used for symmetric encryption and its shortcomings are supplemented by the Bouncy Castle Crypto APIs for Java [Legion of the Bouncy Castle, 2007].

#### 5.7.3.2 Bouncy Castle API

The Bouncy Castle Crypto APIs for Java (BC) are an open-source collection of cryptography APIs that include an API targeted for the Java ME platform [Legion of the Bouncy Castle, 2007]. The three shortcoming of SATSA are rectified by the functionality of the BC because the Java ME version supports asymmetric encryption, the creation of encryption keys and the generation of signatures using a MAC.

The symmetric key used in the symmetric encryption carried out by SATSA is generated by the `ConfigManager` class using the BC. The `XMLEncHandler` class uses the BC to encrypt this generated key. The `XMLSigHandler` class uses the generated key with the BC implementation of the HMAC-SHA1 algorithm to sign XML. To this end, the cryptography requirements to implement the mobile WS-Security configuration are met on the Java ME platform and what remains to conclude this part of the experiment is the identification of an appropriate XML API and SOAP proxy.

#### 5.7.3.3 Third party XML APIs

The shortcomings of the JWSA require that a third party API be investigated. The kXML API is considered as a third party XML API that may be used for the implementation of the design

presented in section 5.7.2 [kXML Community, 2005]. The other third party API considered is NanoXML. NanoXML is rejected because an attempted implementation of the XML security API classes reveals that its Java ME version does not have namespace support [NanoXML Community, 2007]. Such support is critical to the implementation of WS-Security, for example during XML Canonicalisation [Boyer et al., 2002].

The kXML API is selected because Yuan [2002] demonstrates how it may be combined with the BC to secure XML on the Java ME platform. A new SOAP proxy is selected because the JWSA lacks header support as mentioned in section 5.7.3.1.1. The kSOAP API is considered as an obvious choice for a SOAP proxy because it is built on kXML. It is assumed that interfacing kSOAP and kXML classes is trivial [kSOAP Community, 2006][Narayana et al., 2007].

The implementation of the `XMLEncHandler`, `XMLSigHandler` and `HeadersHandler` classes with the kXML API is successful. The kXML `Element` class represents an XML element and it is the only kXML class that is required for the implementation of the `XMLEncHandler`, `XMLSigHandler` and `HeadersHandler` classes. The `Element` class provides the XML attribute support absent in the WSA and it allows for the XML elements it represents to be exported as byte arrays. The BC and SATSA cryptography algorithms accept byte-array representations of plain-text. Therefore, kXML's capability to export XML elements as byte arrays allows encrypted and signed representations of XML elements to be produced.

Despite the successful implementation of `XMLEncHandler`, `XMLSigHandler` and `HeadersHandler` classes with kXML, the interface between kXML and kSOAP is not as trivial as initially assumed. Figure 5.7 shows that the conversion between kXML and kSOAP is done by the `ConfigManager` class. The kSOAP API represents a SOAP message with the `SoapEnvelope` class and the SOAP headers are represented within this class as an array of kXML `Element` objects. This simplifies the attachment of the headers created with kXML to a `SoapEnvelope` object. However, the SOAP body is not represented as a kXML `Element`. Attempts to retrieve and set a `SOAPEnvelope` object's SOAP body member with a kXML `Element` failed. The kSOAP API is rejected as a SOAP API and a third SOAP proxy is considered.

#### 5.7.3.4 Wingfoot SOAP

The Wingfoot SOAP API (WSOAP) is the third SOAP proxy considered for the implementation of the design shown in figure 5.6 [**?**]. WSOAP is considered because it is the only third party SOAP API listed by Sun Microsystems [2007c].

WSOAP represents a SOAP message with the `Envelope` class and this class allows the

SOAP headers and body to be set with Java strings. The kXML elements created by the XML security API classes are converted to Java strings and these strings are used by the ConfigManager class to set up the SOAP header and body members of the `Envelope` class. However, this functionality allows the mobile WS-Security configuration to be implemented only when an insecure SOAP message is constructed manually or by another proxy.

WSOAP provides a web service method call through the `Call` class. The `Call` class plays one of two roles: it is responsible for sending a SOAP message that is manually created with the `Envelope` class or it creates a new SOAP message based on the parameters set through its members.

The first Call class role allows for the `ConfigManager` class to secure a SOAP message but the way it allows this defeats the purpose of a proxy class. The MIDlet must create a SOAP message and send it to the `ConfigManager` before it is sent to the `Call` class. WSOAP is selected for its role as a SOAP proxy which is to create a SOAP message on behalf of a MIDlet.

The second role of the `Call` class constructs the SOAP message using parameters sent to it by the application but the `Call` class does not allow for this message to be modified before it is sent to the provider. Figure 5.7 shows that this is problematic because the constructed SOAP message must be accessed by the ConfigManager for it to be secured before it is sent to the provider. It is for these reasons that WSOAP is also considered inadequate for the implementation of the mobile WS-Security configuration.

#### 5.7.3.5 End of Java ME testing

The testing on Java ME ends with the failure to implement the mobile WS-Security configuration with WSOAP. The literature that is available on Java ME mobile web services is dominated by references to the JWSA, kSOAP and WSOAP proxies. Shu Fang Rui [2006] for example, discusses how to enable mobile web services with JWSA, and Siddiqui [2006] shows how to secure such web services, although the security demonstrated is non-interoperable as previously discussed in section 5.7.3.1.1. Narayana et al. [2007] build a mobile web services provider with kSOAP and Gehlen and Bergs [2004] use WSOAP in their experiments on mobile web services performance. No other proxies could be found in literature and it is concluded on this basis, that the WSA, kSOAP and WSOAP proxies are the most common SOAP proxies available for Java ME. To this end, an analysis with these proxies allows general conclusions to be drawn about the state of interoperable, end-to-end, mobile web services security provision on the Java ME platform.

### 5.7.4 Summary of Java ME Development Experiences

The Java ME third party APIs examined in this thesis, with the exception of a SOAP proxy, fill the gaps needed to implement the mobile WS-Security configuration. The problems of interfacing the kXML WS-Security API with a SOAP proxy may suggest that it is impossible to implement WS-Security with third party Java ME APIs. This was shown not to be the case after the experiment was concluded. The work by Narayana et al. [2007] was discovered by the author at the time of writing and this work demonstrates a different approach to that taken here. The kSOAP libraries were modified by Narayana et al. [2007] to implement a mobile web services provider that supports WS-Security. This work is different from that presented here because it implements a provider instead of a requester. However, it shows that interaction with a provider might have been successfully achieved in the Java ME experiment if kSOAP had been modified. However, the work by Narayana et al. [2007] does not disqualify the implementation carried out with kXML. The kXML implementation demonstrates that is possible to implement the mechanisms of the mobile WS-Security configuration with a third party Java ME XML API. The modification of the SOAP proxy is considered out of scope.

The only noticeable difference between the functionality of the .NET CF and Java ME platforms is found in the formatting of dates with Java ME. MIDP does not provide a class that allows the formatting of dates to be changed from the format employed by the Java ME `java.util.Date` class. This format is incompatible with the date format specified by WS-Security for the username token and the `<wsu:Timestamp>` element [Nadalin et al., 2006a]. The workaround for this requires that the date be exported as a Java string and that the string be manipulated into a format compatible with WS-Security.

The third party Java ME documentation is more detailed than that provided by OpenNetCF Consulting [2007]. The details of the members exposed by the third party Java ME APIs are given in the API documentation and examples of how to use these APIs are available. Examples that directed this implementation effort were mostly taken from the IBM Developer Works site [Machines, 2007], for example the demonstration of XML security by Yuan [2002] and the use of SATSA without a SIM by Huang [2006]. These documents proved more helpful in assisting the implementation than the source code demonstration of the SDF provided by Chesnut [2004].

Some aspects of the available third party API documentation still need improvement. For example, the process of converting a kXML `Element` object to a byte array is unclear from the API documentation. Articles or code examples of how this process is carried out are also not found. The same problem of a lack of documentation is found when an attempt to read in a certificate with the BC is made. Such problems lead to the employment of a trial-and-error

approach when implementing some of the functionality required by the mobile WS-Security test configuration.

The author considers the development in Java ME less challenging than that carried out on the .NET CF with the SDF. However, this subjective view was influenced more by the timing of the Java ME experiment than by the better quality of developer support experienced or the strength of the Java ME platform. The Java ME experiment was started after the .NET CF experiment began and the initial lessons learnt from the .NET CF experiment were transferred to the Java ME experiment. Some of the SDF code also influenced the Java code. These reasons, coupled with the fact that the .NET CF experiment required the modification of existing code whilst the Java ME experiment required a fresh implementation, made it difficult for the author to form a qualitative conclusion as to which mobile platform allows for the easier implementation of the mobile WS-Security configuration.

## 5.8 Summary

| | Standard .NETCF 2.0 | Modified SDF | Standard Java ME | Third Party Java ME |
|---|---|---|---|---|
| Confidentiality | √ | √ | X | √ |
| Integrity | X | √ | X | √ |
| Authentication | √ | √ | X | √ |
| Message Uniqueness | √ | √ | X | √ |
| MTOM | X | X | X | X |

Table 5.1: Summary of platform capabilities in according to the configuration requirements

Table 5.1 reflects the capabilities required for the .NET and Java ME platforms to implement the mobile WS-Security configuration. It is evident from the implementation reported in this chapter, that the SDF mostly aggregates the existing functionality of the standard .NET CF libraries to present an API that may be used to secure a SOAP message with WS-Security. The only new functionality that the SDF introduces when it is used to implement the mobile WS-Security configuration is the HMAC-SHA1 algorithm needed to generate signatures that meet the challenge of integrity. The .NET CF standard libraries do not provide this functionality, hence the SDF implementation of the HMAC-SHA1 algorithm is used. Table 5.1 therefore shows that the challenges of confidentiality, authentication and message uniqueness may be met by a fresh

implementation of XML security classes using the standard .NET CF XML and cryptography APIs. There is no need for such an implementation because it already exists in the SDF, although the SDF implementation needs modification to meet the requirements of the configuration.

Table 5.1 shows that the situation with Java ME is different because the implementation of the mobile WS-Security configuration cannot be done with the standard Java ME XML and cryptography APIs. The absence of SOAP header support in the JWSA automatically disqualifies this standard Java ME API from use in the implementation of any WS-Security mechanism. The burden of implementing the mobile WS-Security configuration is carried entirely by Java ME third party APIs. The kXML and BC libraries provide new implementations of XML processing and cryptography operations instead of utilising existing standard platform functionality as the SDF does on the .NET CF.

Libraries that provide MTOM support are not found on the .NET CF or Java ME platforms. This decreases the feasibility of implementing interoperable, end-to-end, mobile web services security because of the increased size of secured SOAP messages. This is taken to be an indicator that it is feasible to implement this type of security on smartphone platforms but the mobile web services field is not completely ready for its deployment. Further analysis of the mobile web services field with regard to the findings gleaned from the implementation reported in this chapter is provided in the following chapter.

# Chapter 6

# Discussion and Recommendations

## 6.1 Introduction

The implementation detailed in the previous chapter may seem controversial because the results drawn from it are based on one configuration. It may be possible that the SDF, for example, supports interoperable configurations other than the mobile WS-Security configuration specified in section 4.7. However, section 4.7 also suggests that the configuration does not represent the definitive criteria for all interoperable, end-to-end, mobile web services security.

The ability of a platform to support a particular configuration is not of interest to this thesis but the issues that affect mobile web services in the provision of interoperable, end-to-end, security are of interest. The purpose of the implementation was to reveal these issues and this chapter discusses four such issues exposed by the work described in the previous chapter: The lack of developer support; the need for WS-Security API standardisation; the importance of operating system independence; and the requisite for message optimisation.

The first part of this chapter discusses each of these issues as they arise within the .NET CF and Java ME platforms. The implementation experiences from the previous chapter inform this discussion. This platform-specific discussion is used to identify areas that need improvement to further the provision of interoperable, end-to-end, security by mobile web services in general. Recommendations aimed to improve these areas are detailed.

## 6.2 Developer Challenges

The challenges that a developer faces when implementing interoperable, end-to-end, mobile web services security are detailed in this section. These challenges are gleaned from the implemen-

tation of the mobile WS-Security configuration discussed in the previous chapter. The first challenge discussed is that of poor documentation.

### 6.2.1 Poor Documentation

Sections 5.6.6 and 5.7.4 highlight that the poor quality of the documentation relied upon during the experiment adds to the complexity of the implementation carried out. Reliance on this poor documentation results from the need to use third-party libraries to implement the mobile WS-Security configuration. The documentation of the third-party libraries is, in general, less descriptive than that of the standard Java ME and .NET CF libraries.

Java ME developers are forced to use the poorly documented, third-party libraries because the standard Java ME libraries do not yet support the implementation of WS-Security. .NET CF developers may avoid the poor documentation of the SDF by developing their own WS-Security API with the standard .NET CF libraries. The implementation detailed in the previous chapter shows that the standard .NET CF classes provide the functionality to implement the mobile WS-Security configuration, with the exception of the HMAC-SHA1 algorithm. Implementing the HMAC-SHA1 algorithm and other cryptography algorithms introduces extra complexity when securing mobile web services.

### 6.2.2 Complexity of Securing Mobile Web Services

Implementing Java Enterprise Edition web services that are secured with interoperable end-to-end security by WSIT is relatively trivial because of the developer tools available. The NetBeans Integrated Development Environment (IDE) ships with a plug-in that allows a developer to specify a security configuration with the graphical user interface (GUI) of the IDE [Sun Microsystems, 2007f] [Sun Microsystems, 2007h]. The Microsoft Visual Studio 2005 IDE is also integrated with the WSE such that a security configuration may also be specified through the IDE GUI. These tools hide the complexity of configuring a requester or provider with WS-Security.

These tools or their equivalents are not found for the Java ME or .NET CF platforms. This exposes mobile developers to the intricacies of configuring the mobile web services requester with WS-Security. The cryptography aspects of WS-Security are found to be the most challenging during the implementation of the mobile WS-Security configuration. The Java ME implementation of the configuration reveals that the BC libraries do not shield a developer enough from the complexities of cryptography. The generation of an RSA asymmetric encryption key with the BC libraries requires the developer to understand the types of values that may be set for the RSA

exponent and modulus parameters [Schneier, 1996].

A .NET CF developer using the SDF does not have to worry about setting the correct RSA parameter values because the SDF provides the `OpenNETCF.Web.Services2.DecodeCertKey` class that extracts the parameters from a certificate that is read in from the file system of the smartphone. However, a developer who avoids the SDF because of its poor documentation and develops a custom WS-Security API with the standard .NET CF classes needs to deal with such concerns. The developer also needs to develop cryptography algorithms not supported by the .NET CF and this is a complex task [Ferguson and Schneier, 2003]. Ferguson and Schneier [2003] suggest that custom implementations of cryptography algorithms should be avoided because the complexity of these algorithms makes it likely that custom implementations will contain errors. The process of applying WS-Security is also complicated by the challenge of debugging during the development of mobile web services secured with interoperable, end-to-end, security

### 6.2.3 The Challenge of Debugging

The implementation of the mobile WS-Security configuration was considered successful when the requester decoded a secured response message containing the result of the addition of two integers sent to the provider. Consequently an implementation attempt was considered unsuccessful when a response message containing a Java exception was received from the GlassFish server. An example of such a response is shown below:

```
<S:Envelope...>
    ...
    <S:Body>
        <ns2:Fault...>
        <faultcode>ns2:Server</faultcode>
        <faultstring>java.lang.NullPointerException
        </faultstring>
        <detail>
            <ns2:exception...>
            <message>java.lang.NullPointerException
            </message>
            ...
            <ns2:exception>
```

```
            </detail>
            </ns2:Fault>
        </S:Body>
    </S:Envelope>
```

The exception in the snippet was thrown because of a development error during the modification of the SDF libraries. A mistake in the implementation of the `GFXMLEncHandler` lead to the encryption of the contents of the `<wsse:UsernameToken>` element, instead of the entire element as required by WS-Security [Nadalin et al., 2006a]. The child elements of the `<wsse:UsernameToken>` element were received when the message was decrypted by the provider but the provider expected the entire element.

The problem evident in the snippet is that the exception gives little information to suggest what the error is. The stack trace of the exception is not shown in the snippet because it is extensive but it details that the exception was thrown by the WSIT classes that handle the username token. The stack trace information is insufficient because the encrypted username token is represented by cipher-text in the `<xenc:EncryptedData>` element. The cipher-text is scrambled and this prevents attempts to determine the structure of the `<wsse:UsernameToken>` attached in the message. The generated SOAP request message looks perfect because the malformed message part is concealed by encryption. This is not a failing of the encryption operation because its responsibility is to conceal plain-text data regardless of its correctness.

The failure to determine the error from the generated SOAP message lead to an investigation of other potential sources of the exception. One such potential source was a bug in WSIT itself and the author's efforts to locate a bug in the WSIT source code dealing with the username token resulted in a WSIT bug report being registered with the GlassFish Community [Gopal, 2007]. This bug was declared invalid once the source of the error was found in the C# code of the `GFXMLEncHandler` class and not in WSIT.

This example demonstrates that debugging with exceptions thrown by a provider is insufficient because exceptions may not adequately isolate the source of the error. The exception may be thrown because a bug in the requester code generates a malformed SOAP message or the provider may have a bug in its SOAP message handling code. This debugging problem is further aggravated when XML Encryption is applied to a message because errors in encrypted message parts are hidden by the encryption process.

This debugging challenge is common to web services development in general and not just to mobile web services. The GlassFish server would have returned the same exception if the same mistake had been made by a requester on another hardware platform, for example a .NET

Framework client on a desktop computer. However, it is contended that this challenge is likely to be more common in mobile web services development because development tools to implement WS-Security are unavailable. It is harder to make the error detailed in this section because the NetBeans IDE creates and encrypts the username token without the developer needing to write a single line of code.

The three challenges referred to in this section apply to both the .NET CF and Java ME platforms. The implementation of the mobile WS-Security configuration also reveals issues that apply to each platform in particular. These differences allow a comparison of the two platforms to be carried out.

## 6.3 Standard WS-Security Support

The .NET CF platform is considered better prepared for the provision of interoperable, end-to-end, mobile web services security than the Java ME platform. This consideration is based on the degree to which the standardisation efforts of both platforms support this type of security. The manner in which third party libraries are utilised on the platform is an indicator of the state of these standardisation efforts because the third party libraries fill the gaps left by the standard libraries. The reasons why the .NET CF is considered better prepared are discussed first.

### 6.3.1 .NET Compact Framework

Teder [2006] suggests that the .NET CF is more focused towards the enterprise environment than Java ME. This assertion is proved by the fact that the .NET CF has supported web services since its inception [Fox and Box, 2004].

It is unsurprising, when the web services focus of the .NET CF is considered, that the .NET CF provides standard XML and cryptography libraries that are mostly sufficient for third-party libraries to use for building WS-Security support. The SDF does not need to provide its own XML and most of its own cryptography classes for its implementation of WS-Security, XML Encryption and XML Signature. Its implementation is flawed because it addresses a limited set of scenarios that require SOAP message security. For example, the SDF cannot be used to implement the mobile WS-Security configuration without modification to its classes. However, the SDF and the modifications to its classes demonstrate that the underlying functionality of the .NET CF allows for the provision of interoperable, end-to-end, mobile WS-Security. The third-party libraries simply build upon this existing functionality. The gap these third-party libraries

mostly fill concerns the XML security and WS-Security APIs missing on the .NET CF . Third-party libraries on Java ME play a much more elementary role in the provision of interoperable, end-to-end, mobile web services security.

### 6.3.2 Java ME

The Java ME platform is considered less ready than the .NET CF platform for the provision of interoperable, end-to-end, mobile web services security because the standard libraries cannot be used to implement any type of WS-Security. Third-party libraries must be used for the XML and cryptography APIs that are provided by standard libraries on the .NET CF. There is also no sign of an intent to support WS-Security on the Java ME platform from any of the Java standardisation processes.

The fact that JSR 172 does not include any header support suggests that WS-Security was not considered by this JCP standardisation process [Ellis and Young, 2004]. It is stated earlier in section 5.7.3.1.1 that it is impossible to implement WS-Security without header support. JSR 172 also ignores the XML Signature standard because it does not provide support for XML attributes. However, adding attribute support for the sake of supporting the XML Signature standard may be a futile exercise when the JSR specifying the standard Java XML Signature API is not targeted at Java ME [Nadalin and Mullan, 2005]. The XML Encryption and XML Signature JSRs do not include the Java ME platform because they are targeted at Java Standard Edition JVMs only [Nadalin, 2005].

It is concluded on the basis of the status of these JSRs that the Java standardisation effort is not considering support for end-to-end, mobile web services security at this stage. The .NET CF platform does not have standard XML security or WS-Security APIs but its standard libraries have functionality built in such that the provision of such APIs should not be a problem. JSR 172 needs to be overhauled or deprecated if these APIs are to be standardised on Java ME. Although the .NET CF holds the advantage in terms of readiness for the deployment of interoperable, end-to-end, mobile web services security, this capability may not be transferable.

## 6.4 Operating System Independence

Java code is considered managed in the same way as .NET CF code because it runs on a JVM that performs a similar function to the CLR [Li and Knudsen, 2005]. The BC libraries and the SATSA library that provide symmetric encryption are written in managed code. This allows the BC to provide cryptography on any device that supports MIDP, and SATSA to provide symmetric

encryption on devices that support JSR 177 [JSR 177 Expert Group, 2004]. The same operating system independence does not apply to the .NET CF.

Although the .NET CF bears the promise of cross-platform development, its reliance on P/Invoke hinders the meeting of this promise where the underlying operating system is not controlled by Microsoft. It is suggested that the calling of cryptography operations with P/Invoke is the cause of the failure to implement the mobile WS-Security configuration on the Red Five Labs Net60 platform but proving this assertion is beyond the scope of this thesis. Nevertheless it is reasonable to suggest that the strong ties the Microsoft .NET CF has to underlying Microsoft operating systems lessens the potential to run .NET CF applications on CLRs for non-Microsoft operating systems.

The issue of cross-platform development is important because it partly answers the first research question posed in section 1.5, which asks whether the deployment of interoperable, end-to-end, mobile web services security hinders a requester from participating in a secure web services transaction? The answer to the question when it is applied to the .NET CF platform is that there exist cases when it does hinder the requester. It is suggested that a .NET CF requester may be hindered when it does not run on a CLR written by Microsoft. The failure to implement the mobile WS-Security configuration on Net60 and potential problems identified from a reliance on the underlying operating system are the reasons for this suggestion. The lack of secure message-size-reduction functionality may also hinder a mobile requester from participating in a secure web services transaction.

## 6.5 Message Optimisation

The issue of SOAP message-size optimisation is not unique to SOAP security because insecure SOAP messages are already considered verbose [Ng et al., 2005]. Although MTOM is useful for optimising the binary data resulting from cryptography operations, the MTOM standard may be taken to be independent of SOAP message security efforts because binary data in SOAP does not necessarily result from XML Encryption or XML Signature only. Data such as a bitmap image may be represented as binary data in a SOAP message. MTOM may be used to optimise this binary data too. The lack of standard WS-Security APIs on the .NET CF and Java ME platforms does not mean that there is no need for the standard platform libraries to support MTOM.

The lack of MTOM adoption, by .NET CF and Java ME web services, is surprising, because of the network constraints discussed in section 4.4.2. WS-Security may not be provided by both platform standardisation efforts but there are still increased message size costs to be borne from

other sources of binary data. Section 4.4.2 discusses that smaller message sizes are desirable because network operators typically bill on the amount of data transferred. The use of WLANs, like those used to transfer SOAP messages during the experiment, may mitigate this financial cost. However, section 4.4.1 also states the transfer of data consumes more battery power than the smartphone CPU and this resource cost can only be mitigated by reducing the amount of data transferred.

The lack of MTOM provision by third party libraries is less surprising given the nature of these libraries. Section 5.7 states that the SDF does not provide extra web services functionality other than that of the WSE. It is not surprising that the SDF 2.1 does not support MTOM because it implements WSE 2.0 which does not support MTOM either. The kSOAP and WSOAP APIs provide simple SOAP proxy functionality for serialising Java objects into SOAP but they do not support any advanced SOAP manipulation such as binary encodings of SOAP. The simplicity of these APIs does not lead to any high expectations of MTOM support from them.

The lack of support for MTOM alone may be accepted as an indicator of the sentiment towards WS-Security on the .NET CF and Java ME platforms. Although other reasons for optimising binary data exist, as already stated in this section, the growth of message sizes resulting from the implementation of WS-Security is a strong motivator for reducing SOAP message size. The size of the SOAP request message that invokes the calculator web service in the experiment grows from 1kb when it is insecure to 9kb when it is secured. Such growth in message size is more expensive when complex web services require multiple secure messages to be sent by the requester. The lack of MTOM support indicates that WS-Security is a standard not afforded any priority on the .NET CF and Java ME platforms.

## 6.6 Recommendations

The analysis carried out in this thesis with the .NET CF and Java ME platform allows for the formulation of recommendations to improve the state of interoperable, end-to-end, mobile web services security provision. These recommendations are grouped under the developer issues this chapter discusses.

### 6.6.1 Increased Developer Support

The mature state of developer support for WS-Security on traditional web services allows web services developers to focus on developing web services instead of learning how best to implement web services security. The application of cryptography, in particular, is discussed in this

thesis as complex and development tools for traditional web services assist developers to avoid errors that may result from this complexity.

The availability of development tools for securing mobile web services similar to those found in the NetBeans IDE for WSIT and Visual Studio IDE for WSE will offer mobile web services developers the same abstraction as traditional web services developers. The provision of more detailed documentation and code examples for third party libraries will also assist those developers who wish to go further than the abstractions of the development tools, when implementing WS-Security. The production of developer tools for WS-Security may be driven by the adoption of WS-Security by smartphone standardisation processes.

### 6.6.2 Standardisation of WS-Security APIs

The adoption of WS-Security by the smartphone standardisation bodies will improve the quality of developer support and the security of mobile web services. It was noticed, during the experiment, that more development tools supporting standard web services functionality were available than those supporting third party library functionality. The NetBeans IDE provides an automated stub creator that generates JSR 172 web services from WSDL but no such tool for kSOAP ships with NetBeans. The commercial version of the SDF leverages the code completion feature of Visual Studio, but this is not comparable to the GUI configuration support for the WSE. It is therefore assumed that the specification of a standard WS-Security API on a smartphone platform will lead to better development tools.

A standard WS-Security API may lead to the provision of better quality, end-to-end, security for mobile web services on platforms that currently force developers to build their own WS-Security libraries. These custom libraries may contain security flaws that may not necessarily deny a requester from interoperating with a provider but weaken the security deployed. An example of the weakening of the security applied to a SOAP message is the derivation of symmetric keys with a username token. A standard WS-Security API may completely reject this key generation strategy because of the security risk of deriving a symmetric key with a weak password discussed in section 5.6.3.3. A developer who is forced to build a custom WS-Security API is not prohibited from writing code that derives weak symmetric keys that may be easily guessed. Standard WS-Security APIs limit the security flaws that arise from poor developer decisions by making some security decisions for a developer, for example the selection of a strong key derivation technique.

The provision of standard WS-Security smartphone APIs may be in vain if they serve only to further fragment the smartphone environment. Platforms that provide mobile web services but

cannot use the standard WS-Security APIs still force a developer to build a custom WS-Security API. Operating-system-independent APIs will help reduce this fragmentation.

### 6.6.3 Operating System Independence

The testing of the SDF on Net60 reveals that a WS-Security API that calls operating system functionality may, in some cases, serve to fragment the provision of end-to-end security for mobile web services on the platform. This lesson is inapplicable to WS-Security APIs on platforms tied to the operating system, for example the Nokia S60 platform runs on Symbian only and applications developed with S60 run directly on Symbian [Hirsch et al., 2006]. The lesson was previously inapplicable to the .NET CF because it ran exclusively on Microsoft operating systems, but Net60 shows that this situation has changed.

A standard WS-Security API on a managed code platform, that utilises an operating system's functionality, is likely to benefit mobile web services that run on implementations of the platform for that operating system, instead of all implementations of the platform. This fragmentation seems to introduce a competitive advantage that will influence the buying decision of a smartphone to be used in the deployment of end-to-end secure, mobile web services. The prospect of purchasing a smartphone running a Microsoft operating system, for example, becomes more attractive than purchasing a Symbian smartphone with .NET CF support, if a standard .NET CF WS-Security API exclusively works on Microsoft CLRs. This exclusivity may be achieved by designing the API so that it makes calls to the underlying Microsoft operating system. However, this benefit to one mobile web services operating system manufacturer is not worth the costs it presents to the mobile web services field in general.

The limited provision of a standard WS-Security API across a platform means that developers of the platform that cannot use the API are forced into the practice of building custom WS-Security APIs. This activity is shown in the previous recommendation to be unfavourable because poor custom WS-Security APIs may weaken the security deployed to protect messages. The fragmentation of WS-Security APIs potentially results in some strong WS-Security implementations and some weak ones. However, a traditional provider may not be able to discern between platform implementations that employ strong security and those that are plagued by weak, custom WS-Security APIs. A provider, for example, can currently distinguish the SOAP messages generated by the Microsoft .NET CF 2.0 from those of Net60 through HTTP headers only. It is possible that Net60 could, in its later releases, produce the same HTTP headers as the Microsoft .NET CF 2.0. This may make it might be impossible to determine which .NET CF platform implementation created the SOAP messages. To this end, it is foreseeable that an

entire platform may be written off as insecure because some implementations of the platform are forced to host flawed, custom, WS-Security APIs. This is in effect a case of one apple spoiling the entire barrel because it inhibits the adoption of mobile web services on the entire platform instead of platform implementations that do not support the standard WS-Security API.

It is recommended that the standardisation of smartphone WS-Security APIs should result in an API that may used by the mobile web services provided across an entire platform. Managed code environments must use managed code within their API to foster operating-system-independence. For example, a standard .NET CF WS-Security API must apply the AES encryption with the managed code `AesManaged` class instead of the `AesCryptoServiceProvider` class that uses the underlying operating system's AES implementation. The final recommendation provided in this chapter is the support for message optimisation.

### 6.6.4 Message Optimisation Support

Support for a message optimisation standard must be provided in conjunction with a standard WS-Security API because of the increased message size resulting from WS-Security. Section 4.7.1 details that MTOM is the only interoperable, message optimisation strategy available. The EXI binary XML encoding introduced in section 4.4.2 may provide an alternative interoperable standard for message optimisation once it is approved.

The work being carried out on mobile web services specific binary encodings such as Xebu demonstrates that some fragmentation is beginning to take shape in the pursuit of message optimisation for mobile web services. However, the requester nature of mobile web services means that they rely on traditional providers for their realisation and must adapt to the requirements of the provider. This is part of the reasoning provided in section 1.4, for setting interoperability as a dominant theme of this thesis. Therefore, a smartphone platform's support of a non-standard message optimisation strategy is as good as supporting no optimisation strategy at all, if it is not supported by traditional providers. The mobile web services requester has to forgo message optimisation if the provider does not support the optimisation strategy used by the requester. The support for standard message optimisation strategies is recommended because these are likely to be adopted by traditional providers.

## 6.7 Summary

Third-party libraries are considered an inappropriate driver for the provision of interoperable, end-to-end, web services security. The third-party libraries analysed in this thesis provide poor

developer support and do not appropriately shield developers from the complexity of this type of security, which may lead to developers producing poor quality security implementations. Non-standard message optimisation strategies also provide inappropriate assistance for this type of security because they may lead to interoperability issues.

It is for this reason that standards are recommended to drive the provision of interoperable, end-to-end, web services security. Standards simplify the implementation of this type of security. The developer tools, that are built for standards, eliminate some of the security implementation mistakes developers may introduce. Standards also ensure that secure transactions are optimised to face mobile web services network constraints, while keeping the optimisations interoperable. It is important that these standards be applicable to an entire mobile web services platform to eliminate the reliance on third-party APIs by some implementations of the platform. The thesis is concluded in the following chapter.

# Chapter 7

# Conclusion

## 7.1 Introduction

The absence of a standard WS-Security API on a smartphone platform may seem inexcusable but the mobile environment constraints discussed in section 4.4 may provide a compelling reason for the absence of such an API. This concluding chapter begins with a discussion of this absence because it provides context to the conclusions reached in this thesis. The middle sections of this chapter conclude the research reported by this thesis, through a reflection on the research questions asked in section 1.5. The thesis culminates in a statement of its contributions and a suggestion of future work.

## 7.2 The Context to the Research Conclusions

The fact that third-party APIs such as the SDF exist shows a need for WS-Security on smartphones but this need has so far been ignored by the standardisation efforts of the .NET CF and Java ME platforms. Mobile environment constraints require that the size of these platforms be managed and the author is of the opinion that this is a reason for the absence of standard WS-Security APIs on these platforms [Fox and Box, 2004] [Li and Knudsen, 2005].

The size of the .NET CF and Java ME platforms is reduced by having them present only a subset of the functionality that is available on the .NET Framework and the Java Standard Edition respectively [Fox and Box, 2004] [Li and Knudsen, 2005]. The addition of a new standard API would increase the size of these platforms and therefore makes them less efficient on resource constrained devices. This balance between size and functionality applies to all smartphone platforms as they operate within the same mobile environment constraints.

The discussion and recommendations in chapter 6 of this thesis suggest that a standard WS-Security API is an important requirement for smartphone platforms. However, this may be only one of other important requirements, for example the need for more comprehensive cryptography APIs. The resource constraints of the mobile environment dictate that such requirements compete for inclusion as standard APIs because the size of a smartphone platform must be limited. Although a standard WS-Security API is an important requirement, it is possible that other requirements may have a higher priority for inclusion in a smartphone platform's standard libraries. For example, a more comprehensive cryptography API may be prioritised above a WS-Security API because cryptography may be used by many smartphone functions including web services. On the other hand, a WS-Security smartphone API is valuable to mobile web services only. To this end, standard WS-Security APIs for smartphones may become more pervasive when the hindrance to mobile web services participating in end-to-end secured web services transactions becomes more pronounced.

## 7.3 The Hindrance to the Realisation of Mobile Web Services

The first research question, introduced in section 1.5, asks whether the application of end-to-end security on traditional web services providers prevents mobile web services requesters from engaging in web services transactions with these providers? Section 1.4 states that the primary realisation of mobile web services is as requesters and this realisation depends on mobile web services being able to consume the web services offered by traditional providers. Answering this first research question establishes whether end-to-end web services security poses an obstacle to the further realisation of mobile web services. The meeting of the first two research goals, detailed in section 1.5, answers the research question.

The first research goal of identifying appropriate, interoperable, end-to-end, web services security mechanisms is met by the selection of WS-Security in chapter 3 and by the configuration presented in chapter 4. Meeting this goal exposes the fact that WS-Security may hinder mobile web services because of the increased message sizes that result from securing SOAP messages with WS-Security. WS-Security-secured mobile web services that do not employ message optimisation will operate with a decreased battery life penalty and increased billing costs from increased message size. These negative consequences may be considered by mobile web services users as too high a price to pay for engaging in end-to-end secured web services transactions. An unwillingness to engage in such transactions because of the performance and financial drawbacks hinders mobile web services from participating in these transactions.

The second research goal of determining the feasibility of implementing interoperable, end-to-end, mobile web services security is met in the experiment reported in chapter 5. Meeting this goal shows that third-party libraries may prevent mobile web services from participating in end-to-end, secured web services transactions. Chapter 6 argues that the poor developer support provided by third-party libraries and the complexity to which these libraries expose a developer, may lead to weak WS-Security implementations on smartphones. This would harm the reputation of mobile web services because traditional providers might cease to trust the quality of mobile web services security. It is possible that traditional providers may block mobile web services to protect themselves from the potentially poor message security that results from engaging with mobile web services. Such a practice would hinder the ability of mobile web services to interact with end-to-end secured traditional providers.

Therefore, the answer to the first research question is that end-to-end security on traditional providers may prevent mobile web services from interacting with these providers, when a standard message optimisation strategy is not provided and third-party libraries are relied upon. These obstacles however, may be circumvented by a standard WS-Security API that provides a message optimisation strategy.

## 7.4 A Standard WS-Security API as an Improvement

The second research question mentioned in section 1.5 requires the identification of improvements that will further the capability of mobile web services to interact with end-to-end secured traditional providers. Meeting the third research goal described in section 1.5 answers this question and this goal is met in chapter 6.

The recommendations issued in chapter 6 suggest that a standard WS-Security smartphone API that supports a standard message optimisation strategy will lead to better developer support and mitigate the developer errors that may result from using third-party libraries. The failed implementation on the Net60 platform provides an important lesson that a standard WS-Security API must not be operating-system-dependent when it is implemented in managed code.

The context provided in section 7.2 shows that the limited size nature of smartphone platforms means this API would need to compete with other types of API for inclusion in standard smartphone platform libraries. To this end, standard APIs that provide interoperable, end-to-end, mobile web services security may become more pervasive only as more traditional web services providers demand end-to-end security for their web services transactions. This would leave smartphone platform standardisation efforts with no option but to include a WS-Security

API because the failure to interact with traditional providers limits the realisation of mobile web services.

## 7.5 Contributions and Future Work

This thesis concludes with a mention of its contributions and future work that may be carried out as a result of the work reported in it. Three contributions are highlighted in this section and two sets of future work are suggested.

### 7.5.1 Contributions

The first contribution made by this thesis is the provision of a snapshot of mobile web services security. Although this snapshot is limited to two platforms, it highlights the challenges that must be met when providing interoperable, end-to-end, mobile web services security. This thesis is also the first study to experiment with a WS-Security-enabled mobile web services requester on a .NET CF CLR running on a non-Microsoft operating system. The testing with Net60 highlights the issues that may be faced when securing .NET CF web services running on a non-Microsoft CLR and operating system.

This thesis, secondly, contributes through its description of the state of readiness of smartphone platforms for end-to-end secure web services. It is the first study to question the status quo of the reliance on third-party APIs to provide interoperable, end-to-end, mobile web services security. It shows that such a reliance is not ideal for mobile web services that participate in end-to-end, secured web services transactions.

The third contribution of this thesis is the provision of a set of recommendations that may improve this state of readiness. It is acknowledged that a standard WS-Security API may not be a present priority for inclusion into the standard libraries of a smartphone platform. However, the recommendations issued may motivate and guide the development of such an API once the need for it becomes more acute. These three contributions will be of increasing importance as mobile web services evolve from a simple point-to-point environment into a more complex enterprise environment comprised of end-to-end secured providers.

### 7.5.2 Future Work

The need for a standard WS-Security smartphone API provides room for further work on the provision of interoperable, end-to-end, mobile web services security. The following work is

suggested as follow-on from the findings presented in this thesis:

- The EXI message optimisation approach was a work in progress at the time of writing. The promise of this approach warrants its consideration for mobile web services and the reduction of WS-Security message size in particular. The implementation of EXI for mobile devices, within the mobile environment constraints discussed in this thesis, is suggested as future work;
- The reason suggested for the absence of a standard WS-Security API on the .NET CF and Java ME platforms is the potentially increased size of the platforms once such an API is added. Some future work might look into how this increase could be minimised such that the inclusion of a standard WS-Security API might become more attractive for inclusion into the standard libraries of the .NET CF and Java ME platforms.

# Bibliography


3GPP. About 3GPP, 2007a. [Online]. Available WWW :http://www.3gpp.org/About/about.htm (Accessed 30 September 2007).

3GPP. 3G Security;Specification of the 3GPP Confidentiality and Integrity Algorithms; Document 2: KASUMI Specification (Release 7) . Technical Specification, Third Generation Partnership Project, 2007b.

Holt Adams, Dan Gisolfi, James Snell, and Raghu Varadan. Best Practices for Web services: Part 1, Back to the Basics. White Paper", year =, International Business Machines.

Africa Research Bulletin. Telecommunications: Morocco. *Africa Research Bulletin: Economic, Financial and Technical Series*, 43(11):17194A–17194C, 2007. doi: 10.1111/j.1467-6346.2007.00634.x. URL `http://www.blackwell-synergy.com/doi/abs/10.1111/j.1467-6346.2007.00634.x`.

Apache Software Foundation. Apache Synapse, 2007. [Online]. Available WWW:http://ws.apache.org/synapse/project-summary.html (Accessed 30 September 2007).

P.G. Argyroudis, R. Verma, H. Tewari, and D. OMahony. Performance Analysis of Cryptographic Protocols on Handheld Devices. *Proceedings of the Network Computing and Applications, Third IEEE International Symposium on (NCA'04)-Volume 00*, pages 169–174, 2004.

A. Ashkenazi and D. Akselrod. Platform independent overall security architecture in multiprocessor system-on-chip integrated circuits for use in mobile phones and handheld devices. *Computers & Electrical Engineering,*, 33(5-6):407–424, 0 2007.

Alex Biryukov, Adi Shamir, and David Wagner. Real time cryptanalysis of a5/1 on a pc. In *FSE '00: Proceedings of the 7th International Workshop on Fast Software Encryption*, pages 1–18, London, UK, 2001. Springer-Verlag. ISBN 3-540-41728-1.

G. Booch. Unifying Enterprise Development Teams with the UML. *Journal of Database Management*, 10(4), 2000.

D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. Web Services Architecture. W3C Working Group Note, World Wide Web Consortium, 2004. [Online].Available WWW :http://www.w3.org/TR/ws-arch/#service_oriented_architecture (Accessed 31 May 2006).

C. Borcea, L. Iftode, P. Kang, Peng Zhou, and N. Ravi. Smart phone: an embedded system for universal interactions. *Distributed Computing Systems, 2004. FTDCS 2004. Proceedings. 10th IEEE International Workshop on Future Trends of*, 2004.

John Boyer. Canonical XML Version 1.0. W3C Recommendation, World Wide Web Consortium, 2001. [Online].Available WWW :http://www.w3.org/TR/xml-c14n (Accessed 31 May 2006).

John Boyer, Donald E. Eastlake, and Joseph Reagle. Exclusive XML Canonicalization Version 1.0. W3C Recommendation, World Wide Web Consortium, 2002. [Online].Available WWW :http://www.w3.org/TR/xml-exc-c14n/ (Accessed 31 May 2006).

Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and FranÃ§ois Yergeau. Extensible Markup Language (XML) 1.0 (Fourth Edition). W3C Recommendation, World Wide Web Consortium, 2006. [Online].Available WWW : http://www.w3.org/TR/REC-xml/ (Accessed 30 September 2007).

Canalys. 64 million smart phones shipped worldwide in 2006, 2007. [Online]. Available WWW:http://www.canalys.com/pr/2007/r2007024.htm (Accessed 3 April 2007).

Y Chang and C Chen. Smartphone the choice of client platform for mobile commerce. *Computer Standards and Interfaces*, 27, 2005.

Annie Cheneau-Loquay. From networks to uses patterns: the digital divide as seen from africa. *GeoJournal*, 68:55–70, 2007.

Casey Chesnut. Compact framework and wse 2.0 release, 2004. [Online]. Available WWW : http://www.brains-n-brawn.com/default.aspx?vDir=cfwse2 (Accessed 30 September 2007).

Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana. Web services description language (wsdl) version 2.0 part 1: Core language. W3C Recommendation, World Wide Web Consortium, 2007. [Online]. Available WWW :http://www.w3.org/TR/wsdl20/ (Accessed 19 September 2007).

Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web services description language 1.1. Technical Specification, World Wide Web Consortium, 2001. [Online]. Available WWW :http://www.w3.org/TR/wsdl (Accessed 19 September 2007).

Marco Cremonini, Sabrina De Capitani di Vimercati, Ernesto Damiani, and Pierangela Samarati. An xml-based approach to combine firewalls and web services security specifications. In *XMLSEC '03: Proceedings of the 2003 ACM workshop on XML security*, pages 69–78, New York, NY, USA, 2003. ACM Press. ISBN 1-58113-777-X. doi: http://doi.acm.org/10.1145/968559.968571.

T. Dierks and E. Rescorla. RFC 4346 - The Transport Layer Security (TLS) Protocol Version 1.1, 2006. [Online]. Available WWW : http://tools.ietf.org/html/rfc4346 (Accessed 30 September 2007).

D. Eastlake and P. Jones. RFC 3174 - US Secure Hash Algorithm 1 (SHA1) , 2001. [Online]. Available WWW : http://www.faqs.org/rfcs/rfc3174.html (Accessed 30 September 2007).

D. Eastlake, J. Reagle, D. Solo, et al. XML-Signature Syntax and Processing. W3C Recommendation, World Wide Web Consortium, 2005. [Online]. Available WWW :http://www.w3.org/Signature (Accessed 31 May 2006).

Jon Ellis and Mark Young. J2ME Web Services Specification. JCP Specification, Sun Microsystems, 2004. [Online]. Available WWW : http://jcp.org/en/jsr/detail?id=172 (Accessed 25 September 2007).

ETSI. Microsoft ws-i basic security profile 1.0 reference implementation: Final release for the .net framework version 2.0, 2007. [Online]. Available WWW :http://www.etsi.org/WebSite/AboutETSI/AboutEtsi.aspx(Accessed 30 September 2007).

Niels Ferguson and Bruce Schneier. *Practical cryptography*. Wiley, New York, 2003. ISBN 047122894X 9780471228943 0471223573 9780471223573. ID: 51568066.

R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Rfc 2616 - hypertext transfer protocol, 1999. [Online]. Available WWW :http://www.w3.org/Protocols/rfc2616/rfc2616.html (Accessed 30 September 2007).

R.T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, 2000.

Dan Fox and Jon Box. *Building solutions with the Microsoft .NET compact framework architecture and best practices for mobile development*. Addison-Wesley, Boston, 2004. ISBN 0321197887 9780321197887. ID: 52948769.

A. Freeman and A. Jones. *Programming. NET Security*. O'Reilly, 2003. ISBN 0-596-00442-7.

G. Gehlen and R. Bergs. Performance of mobile Web Service Access using the Wireless Application. *Proceedings of World Wireless Congress*, pages 427–432, 2004.

J. Gehtland, D. Almaer, and B. Galbraith. *Pragmatic Ajax: A Web 2.0 Primer*. Raleigh, N.C. : Pragmatic Bookshelf, 2006. ISBN 0976694085.

C. Geuer-Pollmann and J. Claessens. Web services and web service security standards. *Information Security Technical Report*, 10(1):15–24, 2005.

S Gindraux and Deloitte & Touche. From 2g to 3g: A guide to mobile security. In *Third International Conference on 3G Mobile Communication Technologies, 2002.*, pages 308–311, 2002.

Venu Gopal. NullPointerException when handling security header, 2007. [Online]. Available WWW : https://wsit.dev.java.net/issues/show_bug.cgi?id=600 (Accessed 30 November 2007).

Anders Grangard. ebXML Technical Architecture Specification. Technical Specification, OASIS Open and The United Nations Centre for Trade Facilitation and and Electronic Business, 2001. [Online]. Available WWW :http://www.ebxml.org/specs/ebTA.pdf(Accessed 30 September 2007).

M. Gudgin, M. Hadley, and T Rogers. Web services addressing 1.0 - core. Technical Specification, World Wide Web Consortium, 2006. [Online]. Available WWW : http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/ (Accessed 19 September 2007).

M. Gudgin, M. Hadley, N. Mendelsohn, JJ. Moreau, H. Frystyk Nielsen, Anish Karmarkar, and Yves Lafon. Soap version 1.2 part 1: Messaging framework (second edition). Technical Specification, World Wide Web Consortium, 2007. [Online]. Available WWW :http://www.w3.org/TR/2007/REC-soap12-part0-20070427/ (Accessed 19 September 2007).

Martin Gudgin, Noah Mendelsohn, Mark Nottingham, and Herve Ruellan. SOAP Message Transmission Optimization Mechanism. W3C Recommendation, World Wide Web Consortium, 2005. [Online].Available WWW :http://www.w3.org/TR/soap12-mtom/ (Accessed 31 May 2006).

V. Gupta and S. Gupta. Securing the wireless internet. *Communications Magazine, IEEE*, 39 (12):68–74, 2001.

Marc J. Hadley. Web Application Description Language (WADL). Technical Specification, Sun Microsystems, 2006. [Online]. Available WWW : https://wadl.dev.java.net/wadl20061109.pdf (Accessed 25 September 2007).

Johannes Helander and Yong Xiong. Secure web services for low-cost devices. In *ISORC '05: Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*, pages 130–139, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2356-0. doi: http://dx.doi.org/10.1109/ISORC.2005.50.

Frederick Hirsch, John Kemp, and Jani Ilkka. *Mobile Web Services : Architecture and Implementation*. John Wiley and Sons, 2006. ISBN 0-470-01596-9.

R. Housley, W. Ford, W. Polk, and D. Solo. RFC 2459 - Internet X.509 Public Key Infrastructure, 1999. [Online]. Available WWW:http://www.ietf.org/rfc/rfc2459.txt (Accessed 30 September 2007).

Hung-Yun Hsieh, Chung-Wei Li, Shuo-Wei Liao, Yu-Wen Chen, Tsung-Lin Tsai, and Hsiao-Pu Lin. Moving toward end-to-end support for handoffs across heterogeneous telephony systems on dual-mode mobile devices. *Computer Communications,*, In Press, Corrected Proof, 2007.

Tea Vui Huang. SIMless confidentiality. Technical Article, International Business Machines, 2006. [Online]. Available WWW :http://www-128.ibm.com/developerworks/wireless/library/wi-simless/ (Accessed 19 Septemer 2007).

T. Imamura, B. Dillaway, E. Simon, et al. XML Encryption Syntax and Processing. W3C Recommendation, World Wide Web Consortium, 2005. [Online].Available WWW :http://www.w3.org/Encryption/2001 (Accessed 31 May 2006).

Wassim Itani and Ayman Kayssi. J2me application-layer end-to-end security for m-commerce. *Journal of Network and Computer Applications,*, 27(1):13–32, 1 2004.

Ian Jacobs and Norman Walsh. Architecture of the World Wide Web, Volume One. W3C Recommendation, World Wide Web Consortium, 2004. [Online]. Available WWW :http://www.w3.org/TR/webarch/ (Accessed 31 May 2006).

Andreas Janecek and Helmut Hlavacs. Programming interactive real-time games over wlan for pocket pcs with j2me and .net cf. In *NetGames '05: Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, pages 1–8, New York, NY, USA, 2005. ACM. ISBN 1-59593-156-2. doi: http://doi.acm.org/10.1145/1103599.1103603.

Mario Jeckle and Erik Wilde. Identical principles, higher layers: Modeling web services as protocol stack. In *XML Europe 2004, Amsterdam*, 2004.

Steve Jobs. Third Party Applications on the iPhone, 2007. [Online]. Available WWW : http://developer.apple.com/iphone/devcenter/third_party_apps.php (Accessed 30 November 2007).

S. Josefsson. RFC 3548 - The Base16, Base32, and Base64 Data Encodings, 2003. [Online]. Available WWW :http://www.faqs.org/rfcs/rfc3548.html (Accessed 30 September 2007).

JSR 118 Expert Group. Mobile Information Device Profile for Java 2 Micro Edition Version 2.1. JCP Specification, Sun Microsystems, 2006. [Online]. Available WWW : http://jcp.org/en/jsr/detail?id=118 (Accessed 25 September 2007).

JSR 177 Expert Group. Security and Trust Services API for J2ME. JCP Specification, Sun Microsystems, 2004. [Online]. Available WWW : http://jcp.org/en/jsr/detail?id=177 (Accessed 25 September 2007).

JSR 248 Expert Group. Mobile Service Architecture Specification. JCP Specification, Sun Microsystems, 2006. [Online]. Available WWW : http://jcp.org/en/jsr/detail?id=248 (Accessed 25 September 2007).

B. Kaliski and J. Staddon. RFC 2437 - PKCS #1: RSA Cryptography Specifications Version 2.0, 1998. [Online]. Available WWW :http://www.faqs.org/rfcs/rfc2437.html (Accessed 30 September 2007).

Kangasharju. Efficient implementation of xml security for mobile devices. *IEEE International Conference on Web Services, 2007. ICWS 2007.*, 00:134–141, 2007. doi: http://doi.ieeecomputersociety.org/10.1109/ICWS.2007.81.

Jaakko Kangasharju, Tancred Lindholm, and Sasu Tarkoma. On encrypting and signing binary xml messages in the wireless environment. In *ICWS '06: Proceedings of the IEEE International Conference on Web Services (ICWS'06)*, pages 637–646, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2669-1. doi: http://dx.doi.org/10.1109/ICWS.2006.95.

Jaakko Kangasharju, Tancred Lindholm, and Sasu Tarkoma. Xml messaging for mobile devices: From requirements to implementation. *Computer Networks,*, 51(16):4634–4654, 11/14 2007.

Sumit Kasera and Mishit Narang. *3G Mobile Networks: Architecture, Protocols and Proceedures*. McGraw-Hill, 2005.

P Kearney. Message level security for web services. *Information Security Technical Reportl*, 10 (1):41–50, 2005.

P Kearney, J Chapman, N Edwards, M Gifford, and L He. An Overview of Web Services Security. *BT Technology Journal*, 22(1):27–42, 2004a.

P. Kearney, J. Chapman, N. Edwards, M. Gifford, and L. He. An Overview of Web Services Security. *BT Technology Journal*, 22(1):27–42, 2004b.

S. Kent and R. Atkinson. Rfc 2401 - security architecture for the internet protocol, 1998. [Online]. Available WWW :http://www.ietf.org/rfc/rfc2401.txt (Accessed 30 September 2007).

K. Khoo and L. Zhou. Managing web services security. *Journal of Information Technology Management*, 15(3-4):14, 2004.

Richard Kissel. Glossary of Key Information Security Terms. Glossary, National Institute of Standards and Technology, US Department of Commerce, 2006. [Online]. Available WWW :http://csrc.nist.gov/publications/nistir/NISTIR7298_Glossary_Key_Infor_Security_Terms.pdf (Accessed 3 Sep 2007).

Andre N. Klingsheim, Veborn Moen, and Kjell J. Hole. Challenges in securing networked j2me applications. *Computer*, 40(2):24–30, 2007. ISSN 0018-9162. doi: http://dx.doi.org/10.1109/MC.2007.49.

Doug Kohlert and Arun Gupta. The Java API for XML-Based Web Services (JAX-WS) 2.1. JCP Specification, Sun Microsystems, 2007. [Online]. Available WWW : http://jcp.org/en/jsr/detail?id=224 (Accessed 25 September 2007).

H. Krawczyk, M. Bellare, and R. Canetti. Rfc 2104 - HMAC: Keyed-Hashing for Message Authentication, 1997. [Online]. Available WWW :http://www.faqs.org/rfcs/rfc2104.html (Accessed 30 September 2007).

kSOAP Community. kSOAP 2, 2006. [Online]. Available WWW:http://ksoap2.sourceforge.net/ (Accessed 3 April 2007).

kXML Community. About kXML, 2005. [Online]. Available WWW:http://kxml.sourceforge.net/about.shtml (Accessed 3 April 2007).

Legion of the Bouncy Castle. Legion of the Bouncy Castle Java cryptography APIs, 2007. [Online]. Available WWW:http:http://www.bouncycastle.org/java.html (Accessed 3 April 2007).

Sing Li and Jonathan Knudsen. *Beginning J2ME from novice to professional*. Apress, Berkeley, Calif., 2005. ISBN 1590594797 9781590594797. ID: 60518509.

International Business Machines. Developer Works: IBM's resource for developers, 2007. [Online]. Available WWW : http://www.ibm.com/developerworks/(Accessed 30 November 2007).

T.L. Martin. *Balancing Batteries, Power and Performance: System Issues in CPU Speed-Setting for Mobile Computing*. PhD thesis, Carnegie Mellon University, 1999.

Michael McIntosh, Martin Gudgin, K. Scott Morrison, and Abbie Barbir. Basic Security Profile Version 1.0. Technical Specification, The Web Services-Interoperability Organization, 2007. [Online]. Available WWW : http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0.html#ProcessingOrder (Accessed 31 May 2006).

M. Mealling and R. Denenberg. URI, URL, and URN: Clarificiations and Recommendations, 2002. [Online]. Available WWW : http://www.ietf.org/rfc/rfc3305.txt (Accessed 30 September 2007).

Microsoft Corporation. How to install root certificates on a windows mobile-based device, 2007a. [Online]. Available WWW :http://support.microsoft.com/kb/915840 (Accessed 30 September 2007).

Microsoft Corporation. Web services enhancements, 2007b. [Online]. Available WWW :http://msdn2.microsoft.com/en-us/webservices/Aa740663.aspx (Accessed 30 September 2007).

Microsoft Corporation. Web services enhancements architecture, 2007c. [Online]. Available WWW :http://msdn2.microsoft.com/en-us/library/ms826813.aspx (Accessed 30 September 2007).

Microsoft Corporation. Microsoft WS-I Basic Security Profile 1.0 Reference Implementation: Final Release for the .NET Framework version 2.0, 2007d. [Online]. Available WWW :http://www.microsoft.com/downloads/details.aspx?familyid=40e3d4c5-2105-47f1-ba26-9e4c29ba6990I&displaylang=en(Accessed 30 September 2007).

Microsoft Corporation. What's new in web services enhancements (wse) 3.0, 2005. [Online]. Available WWW :http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwse/html/newwse3.asp (Accessed 30 September 2007).

Mono Project Community. About Mono, 2007. [Online]. Available WWW:http://www.mono-project.com/Mono:About (Accessed 3 December 2007).

T. Moyo, B. Irwin, and M. Wright. Securing mobile commerce interactions through secure mobile web services. In *8th Annual Conference on WWW Applications: South Africa*, 2006.

MSDN Forums. Can wse 3.0 be used used with the compact framework 2.0 on mobile 5.0 devices?, 2006. [Online]. Available WWW :http://forums.microsoft.com/MSDN/ShowPost.aspx?PostID=243245&SiteID=1 (Accessed 30 September 2007).

Paul Muschamp. An Introduction to Web Services. *BT Technology Journal*, 22(1), 2004. ISSN 9-18.

A Nadalin. SOAP Message Security: Minimalist Profile (MProf) . Draft Technical Specification, OASIS Open, 2003. [Online]. Available WWW :http://xml.coverpages.org/WSS-MinimalistProfile-20030307.pdf (Accessed 30 September 2007).

A Nadalin, C Kaler, R Monzillo, and P Hallam-Baker. Web Services Security:SOAP Message Security 1.0 (WS-Security 2004). Technical Specification, OASIS Open, 2004. [Online]. Available WWW :http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf (Accessed 31 May 2006).

A Nadalin, C Kaler, R Monzillo, and P Hallam-Baker. Web Services Security:SOAP Message Security 1.1(WS-Security 2004). Technical Specification, OASIS Open, 2006a. [Online]. Available WWW :http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf (Accessed 31 May 2006).

A Nadalin, C Kaler, R Monzillo, and P Hallam-Baker. Web Services Security Kerberos Token Profile 1.1. Technical Specification, OASIS Open, 2006b. [Online]. Available WWW :http://www.oasis-open.org/committees/download.php/16788/wss-v1.1-spec-os-KerberosTokenProfile.pdf (Accessed 30 September 2007).

A Nadalin, C Kaler, R Monzillo, and P Hallam-Baker. Web Services Security:SAML Token Profile 1.1. Technical Specification, OASIS Open, 2006c. [Online]. Available WWW :http://www.oasis-open.org/committees/download.php/16768/wss-v1.1-spec-os-SAMLTokenProfile.pdf (Accessed 30 September 2007).

A Nadalin, C Kaler, R Monzillo, and P Hallam-Baker. Web Services Security UsernameToken Profile 1.1. Technical Specification, OASIS Open, 2006d. [Online]. Available WWW :http://www.oasis-open.org/committees/download.php/16782/wss-v1.1-spec-os-UsernameTokenProfile.pdf ((Accessed 30 September 2007).

A Nadalin, C Kaler, R Monzillo, and P Hallam-Baker. Web Services Security X.509 Certificate Token Profile 1.1. Technical Specification, OASIS Open, 2006e. [Online]. Available WWW :http://www.oasis-open.org/committees/download.php/16785/wss-v1.1-spec-os-x509TokenProfile.pdf (Accessed 30 September 2007).

A Nadalin, Marc Goodner, Martin Gudgin, Abbie Barbir, and Hans Granqvist. WS-SecurityPolicy 1.2). Technical Specification, OASIS Open, 2007. [Online]. Available WWW :http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.pdf (Accessed 1 November 2007).

Anthony Nadalin. XML Digital Encryption APIs. JCP Specification, Sun Microsystems, 2005. [Online]. Available WWW : http://jcp.org/en/jsr/detail?id=106 (Accessed 25 September 2007).

Anthony Nadalin and Sean Mullan. XML Digital Signature APIs. JCP Specification, Sun Microsystems, 2005. [Online]. Available WWW : http://jcp.org/en/jsr/detail?id=105 (Accessed 25 September 2007).

NanoXML Community. About NanoXML, 2007. [Online]. Available WWW:http://nanoxml.cyberelf.be/index.html(Accessed 3 April 2007).

Srirama Satish Narayana, Jarke Matthias, and Wolfgang Prinz. Security analysis of mobile web service provisioning. *International Journal of Internet Technology and Secured Transactions*, 1:151–171(21), 2007. URL `http://www.ingentaconnect.com/content/ind/ijitst/2007/00000001/F0020001/art00008`.

A. Ng, P. Greenfield, and S. Chen. A Study of the Impact of Compression and Binary Encoding on SOAP Performance. *Proceedings of the Sixth Australasian Workshop on Software and System Architectures (AWSA2005)*, 2005.

”NIST”. Specification for the Advanced Encryption Standard (AES). Federal Information Processing Standards Publication, National Institute of Standards and Technology, US Department of Commerce, 2001. [Online]. Available WWW :http://www.csrc.nist.gov/publications/fips/fips197/fips-197.pdf (Accessed 3 Sep 2007).

”NIST”. Digital Signature standard (DSS). Federal Information Processing Standards Publication, National Institute of Standards and Technology, US Department of Commerce, 2000. [Online]. Available WWW :http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf (Accessed 3 Sep 2007).

Nokia Corporation. Device Details- Nokia N80, 2007. [Online]. Available WWW : http://www.forum.nokia.com/devices/N80(Accessed 30 November 2007).

Open Mobile Alliance. OMA Web Services Enabler (OWSER) : Overview. Technical Specification, Open Mobile Alliance, 2006.

OpenNetCF Consulting. Smart device framework, 2007. [Online]. Available WWW :http://www.opennetcf.com/Default.aspx?tabid=67 (Accessed 30 September 2007).

C. Enrique Ortiz. Web Services APIs for J2ME, Part 1: Remote service invocation API. Technical article, International Business Machines, 2004. [Online]. Available WWW : http://www.ibm.com/developerworks/wireless/library/wi-jsr/ (Accessed 30 November 2007).

Enrique Ortiz. The security and trust services api (satsa) for j2me: The security apis, 2005. [Online]. Available WWW :http://developers.sun.com/mobility/apis/articles/satsa2/ (Accessed 30 September 2007).

S. Pal, J. Marsh, and A. Layman. A case against standardizing binary representation of xml. In *Workshop on Binary Interchange of XML Information Item Sets*, 2003.

Dhiru Pandey. Connected Limited Device Configuration 1.1. JCP Specification, Sun Microsystems, 2006a. [Online]. Available WWW : http://jcp.org/en/jsr/detail?id=139 (Accessed 25 September 2007).

Dhiru Pandey. Implementing Enterprise Web Services. JCP Specification, Sun Microsystems, 2006b. [Online]. Available WWW : http://jcp.org/en/jsr/detail?id=109 (Accessed 25 September 2007).

Mike Papazaglou and Willem-Jan van-den Heuvel. Service Oriented Computing: State-of-the-Art and Open Research Issues. *Information Security Technical Report*, 9(3):99–109, 2004. URL `http://www.michalek.org/appsec/appsecxmlschemas.pdf`.

Jonathan B. Postel. Rfc 821 - simple mail transfer protocol, 1982. [Online]. Available WWW :http://tools.ietf.org/rfc/rfc821.txt (Accessed 30 September 2007).

Dave Raggett, J. Reagle, D. Solo, et al. XML-Signature Syntax and Processing. W3C Recommendation, World Wide Web Consortium, 2005. [Online]. Available WWW :http://www.w3.org/Signature (Accessed 31 May 2006).

Nick Ragouzis, John Hughes, Rob Philpott, and Eve Maler. Security Assertion Markup Language (SAML) V2. 0 Technical Overview. Technical Specification, OASIS Open, 2006. [Online]. Available WWW :http://www.oasis-open.org/committees/download.php/20645/sstc-saml-tech-overview-2%200-draft-10.pdf (Accessed 30 September 2007).

Ramesh Nagappan, Robert Skoczylas, and Rima Patel Sriganesh. *Developing Java Web Services: Architecting and Developing Secure Web Services Using Java*. Wiley Publishing, 2003.

B Ramsdell. Rfc 2633 - s/mime version 3 message specifica, 1999. [Online]. Available WWW :http://www.faqs.org/rfcs/rfc2633.html (Accessed 30 September 2007).

Red Five Labs. Net60 - Opening Symbian devices to .NET development. White Paper, Red Five Labs, 2007. [Online]. Available WWW :http://www.redfivelabs.com/content/whitepaper.aspx (Accessed 2 Aug 2007).

Leonard Richardson and Sam Ruby. *RESTful Web Services*. O'Reilly Media, 2007. ISBN 0-596-52926-0.

JonathanB Rosenberg, DavidL Remy, and Inc NetLibrary. *Securing Web services with WS-Security demystifying WS-Security, WS-Policy, SAML, XML Signature, and XML Encryption*. SAMS Indianapolis, IN, 2004. ISBN 0768663547 9780768663549.

J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2(4):277–288, 1984. ISSN 0734-2071. doi: http://doi.acm.org/10.1145/357401.357402.

Geoff Sanders, Lionel Thorens, Manfred Reisky, Oliver Rulik, and Stefan Deylitz. *GPRS Networks*. John Wiley and Sons, 2003.

John Schneider and Takuki Kamiya. Efficient XML Interchange (EXI) Format 1.0. W3C Working Draft, World Wide Web Consortium, 2007. [Online].Available WWW :http://www.w3.org/TR/2007/WD-exi-20070716/ (Accessed 10 November 2007).

B. Schneier. *Applied Cryptography Second Edition: Protocols, algorithms, and source code in C.* John Wiley & Sons, Inc, 1996. ISBN 0-471-12845-7.

Jerry Schwarz, Bret Hartman, Anthony Nadalin, Chris Kaler, Mark Davis, , K. Scott Morrison, and Frederick Hirsch. Security Challenges, Threats and Countermeasures Version 1.0. Technical Specification, The Web Services-Interoperability Organization, 2007. [Online]. Available WWW :http://www.ws-i.org/Profiles/BasicSecurity/SecurityChallenges-1.0.pdf (Accessed 3 August 2007).

N. Scott, S. Batchelor, J. Ridley, and B. Jorgensen. The Impact of Mobile Phones in Africa. Report, Commission for Africa, London, 2004. [Online]. Available WWW :http://www.commissionforafrica.org/french/report/background/scott_et_al_background.pdf (Accessed 2 Aug 2007).

Scott Seely. Understanding WS-Security. Technical article, Microsoft Corporation, 2002. [Online].Available WWW :http://msdn2.microsoft.com/en-us/library/ms977327.aspx (Accessed 30 September 2007).

Kevin Sharp. Series 40 5th Edition Announced, 2007. [Online]. Available WWW : http://blogs.forum.nokia.com/index.php?op=ViewArticle&blogId=31249&articleId=510 (Accessed 30 November 2007).

George Shepherd. Using SOAP Extensions in ASP.NET. *MSDN Magazine*, 2004.

S. Shirasuna, A. Slominski, L. Fang, and D. Gannon. Performance comparison of security mechanisms for grid services. *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 360–364, 2004.

R Shirey. RFC 2828 - Internet Security Glossary, 2000. [Online]. Available WWW :http://www.faqs.org/rfcs/rfc2828.html (Accessed 30 September 2007).

Shu Fang Rui. Designing mobile Web services. Technical article, International Business Machines, 2006. [Online]. Available WWW : http://www-128.ibm.com/developerworks/java/library/wi-websvc/index.html?ca=drs- (Accessed 30 July 2007).

Bilal Siddiqui. Building a secure SOAP client for J2ME, Part 1: Exploring Web Services APIs (WSA) for J2ME. Technical Article, International Business Machines, 2006. [Online]. Available WWW :https://www6.software.ibm.com/developerworks/education/ws-soa-securesoap1/ws-soa-securesoap1-a4.pdf (Accessed 19 Septemer 2007).

F. Siegemund, R. Sugar, A. Gefflaut, and F. van Megen. Porting the .NET Compact Framework to Symbian Phones. *Technology*, 5(3), 2006.

A Singhal, T Winograd, and K Scarfone. Guide to Secure Web Services: Recommendations of the National Institute of Standards and Technology. Report, National Institute of Standards and Technology, US Department of Commerce, 2007. [Online]. Available WWW :http://csrc.nist.gov/publications/nistpubs/800-95/SP800-95.pdf (Accessed 3 Sep 2007).

Maarten Struys. Building .NET Compact Framework 2.0 Applications That Have .NET Compact Framework 1.0 Code Compatibility. Technical Article, "Microsoft Corporation", 2006. [Online]. Available: WWW:http://msdn2.microsoft.com/en-us/library/bb435027.aspx.

Sun Microsystems. The Java Community Process Program - Introduction - Program Overview , 2007a. [Online]. Available WWW:http://jcp.org/en/introduction/overview (Accessed 3 December 2007).

Sun Microsystems. The Java ME Platform the Most Ubiquitous Application Platform for Mobile Devices, 2007b. [Online]. Available WWW:http://java.sun.com/javame/index.jsp (Accessed 3 April 2007).

Sun Microsystems. 3rd Party Tools and Downloads, 2007c. [Online]. Available WWW : http://developers.sun.com/mobility/allsoftware/index.html (Accessed 30 November 2007).

Sun Microsystems. Web Application Description Language, 2007d. [Online]. Available WWW : https://wadl.dev.java.net/ (Accessed 25 September 2007).

Sun Microsystems. The Java ME Device Table, 2007e. [Online]. Available WWW : http://developers.sun.com/mobility/device/pub/device/list.do?filterIds=1034 (Accessed 30 November 2007).

Sun Microsystems. Web Services Interoperability Technology (WSIT) Module, 2007f. [Online]. Available WWW : http://websvc.netbeans.org/wsit/ (Accessed 30 November 2007).

Sun Microsystems. Sun Java System Application Server, 2007g. [Online]. Available WWW : http://www.sun.com/software/products/appsrvr/index.xml (Accessed 25 September 2007).

Sun Microsystems. WSIT Tutorial. Technical tutorial, Sun Microsystems, 2007h. [Online]. Available WWW : http://java.sun.com/webservices/interop/reference/tutorial/doc/WSITTutorial.pdf (Accessed 25 September 2007).

Sun Microsystems. GlassFish Communuty, 2007i. [Online]. Available WWW :https://glassfish.dev.java.net/ (Accessed 25 September 2007).

Sun Microsystems. What is Metro?, 2007j. [Online]. Available WWW : https://metro.dev.java.net/discover/ (Accessed 25 September 2007).

Sun Microsystems. XML and Web Services Security 3.0, 2007k. [Online]. Available WWW :https://xwss.dev.java.net/overview.html(Accessed 25 September 2007).

Symbian Software. Certificate management overview, 2007. [Online]. Available WWW :http://www.symbian.com/developer/techlib/v9.3docs/doc_source/guide/Security-subsystem-guide/SecurityGuide/CertMan/CertManOverview.html#certman%2eoverview (Accessed 30 September 2007).

Yoshiaki Takahashi. Mobile Commerce. Report, Organisation for Economic Co-operation and Development, 2006. [Online]. Available WWW :http://www.oecd.org/dataoecd/22/52/38077227.pdf (Accessed 2 Aug 2007).

Kezhe Tang, Shiping Chen, David Levy, John Zic, and Bo Yan. A performance evaluation of web services security. *edoc*, 0:67–74, 2006. ISSN 1541-7719. doi: http://doi.ieeecomputersociety.org/10.1109/EDOC.2006.12.

William Tay. Enveloped signatures - xmldsig and ws-security, 2004. [Online]. Available WWW :http://www.softwaremaker.net/blog/EnvelopedSignaturesXMLDSIGAndWSSecurity.aspx (Accessed 30 September 2007).

Andres Teder. The problem sets of Java Micro Edition technology. Master's thesis, University of Tartu, 2006.

Third Generation Partnership Project. Technical Specification Group Terminals;Specification of the Subscriber Identity Module -Mobile Equipment (SIM - ME) interface (Release 5). Technical Specification, Third Generation Partnership Project, 2001a.

Third Generation Partnership Project. Subscriber Identity Modules (SIM),Functional characteristics (Release 4). Technical Specification, Third Generation Partnership Project, 2001b.

M. Tian, T. Voigt, T. Naumowicz, H. Ritter, and J. Schiller. Performance considerations for mobile web services. *Computer Communications*, 27(11):1097–1105, 2004.

Sameer Tyagi. RESTful Web Services. Technical article, Sun Microsystems, 2006. [Online]. Available WWW : http://java.sun.com/developer/technicalArticles/WebServices/restful/index.html (Accessed 25 September 2007).

Pieter Ben van der Merwe. Mobile Commerce over GSM: A Banking Perspective on GSM. Master's thesis, University of Pretoria, 2003.

A Vedamuthu, David Orchard, Frederick Hirsch, Maryann Hondo, Prasad Yendluri, Toufic Boubez, and Umit Yalcinalp. Web Services Policy 1.5 - Attachment. W3C Recommendation, World Wide Web Consortium, 2007a. [Online].Available WWW :http://www.w3.org/TR/2007/REC-ws-policy-attach-20070904/ (Accessed 25 October 2007).

A Vedamuthu, David Orchard, Frederick Hirsch, Maryann Hondo, Prasad Yendluri, Toufic Boubez, and Umit Yalcinalp. Web Services Policy 1.5 - Framework. W3C Recommendation, World Wide Web Consortium, 2007b. [Online].Available WWW :http://www.w3.org/TR/2007/REC-ws-policy-20070904/ (Accessed 25 October 2007).

Andy Wigley. Migrating symbian os applications to windows mobile-based smartphones. Technical article, 2005.

Hervey Wilson. Hervyw's blog - multiple security headers, 2004. [Online]. Available WWW :http://www.dynamic-cast.com/mt-archives/000069.html (Accessed 30 September 2007).

Jim Wilson. Differences in Microsoft .NET Compact Framework Development between the Pocket PC and and Windows CE .NET. Technical Article, "Microsoft Corporation", 2003. [Online]. Available WWW:http://msdn2.microsoft.com/en-us/library/aa446544.aspx.

Jim Wilson. What's New in the .NET Compact Framework 2.0. Technical Article, "Microsoft Corporation", 2005. [Online]. Available WWW:http://msdn2.microsoft.com/en-us/library/aa446574.aspx.

Madeleine Wright. A Detailed Investigation of Interoperability for Web Services. Master's thesis, Rhodes University, 2005.

Wangming Ye. Web services programming tips and tricks: Improve interoperability between J2EE technology and .NET,Part 1. Technical Article, International Business Machines, 2004. [Online]. Available WWW :http://www.ibm.com/developerworks/webservices/library/ws-tip-j2eenet1/ (Accessed 19 Septemer 2007).

Michael Yuan. Securing your J2ME/MIDP apps. Technical Article, International Business Machines, 2002. [Online]. Available WWW :http://www.ibm.com/developerworks/library/j-midpds.html(Accessed 19 Septemer 2007).

Michael Juntao Yuan. What Is a Smartphone, 2005. [Online]. Available WWW : http://www.oreillynet.com/pub/a/wireless/2005/08/23/whatissmartphone.html (Accessed 30 July 2007).

P. Zheng and LM Ni. Spotlight: the rise of the smart phone. *Distributed Systems Online, IEEE*, 7(3), 2006.

H. Zimmermann. Osi reference model the iso model of architecture for open systems interconnection. *Innovations in Internetworking*, pages 2–9, 1988.

Michael zur Muehlen, Jeffrey V. Nickerson, and Keith D Swenson. The security risks of ajax/web 2.0 applications. *Network Security*, 40(1):9–29, 2005. URL `http://www.sciencedirect.com/science/article/B6V8S-4CF5FWK-1/2/9414f6196f8c6d6ffe0c8efca30a0c83`.

# Glossary

**AES:** The Advanced Encryption Standard is a NIST encryption standard based on the Rijndael encryption algorithm.

**cdma2000:** Cdma2000 is a cellular network radio technology that is named after the code division multiple access radio transmission technique on which it is based.

**e-commerce:** The Organisation for Economic Co-operation and Development definition of e-commerce as the sale or purchase of goods or services, whether between businesses, households, individuals, governments, or other public or private organisations, conducted over computer-mediated networks is employed in this thesis.

**HMAC:** The Keyed-Hashing for Message Authentication mechanism is a type of MAC that is generated with a hash function, for example the SHA-1 algorithm.

**HTTP:** The Hypertext Transfer Protocol is used to transfer information on the Internet. It utilises a request/response pattern where a client sends a request message to a server and the server sends back a reponse message.

**m-commerce:** The Organisation for Economic Co-operation and Development definition of m-commerce as "a business model that allows a consumer to complete all steps of a commercial transaction" using a mobile device is employed in this thesis.

**NIST:** The US National Institute of Standards and Technology is an agency under the US Department of Commerce. The agency's Computer Security Division provides security standards for the US goverment and industry.

**OMA:** The Open Mobile Alliance is a consortium of mobile industry companies. It develops open standards for the provision of interoperable mobile services.

**Quality of the security:** Ths is similar to the quality of service as it refers an agreed minimum standard of security for a transaction.

**RSA Algorithm:** The RSA algorithm is named after its inventors: Ron Rivest; Adi Shamir; and Len Adleman.

**SAML:** The Security Assertion Markup Language is an XML based language for providing authentication and authorisation information.

**SHA-1:** The Secure Hash Algorithm-1 is a hash function standardised by NIST.

**SOAP:** SOAP is an XML based protocol for exchanging information. SOAP is no longer acronym according to the SOAP Version 1.2 Primer.

**TLS:** The Transport Layer Security protocol provides a secure channel on top of a reliable transport protocol such as TCP. This channel is always between two parties.

**Traditional Web Services:** The term traditional in this context is used to describe pervasive devices within the web services environment. These are mostly interconnected with fixed line network connections and larger than handheld mobile devices.

**Triple DES:** Triple DES is a NIST symmetric encryption standard and is the successor to the Digital Encryption Standard. It has been superseded by AES.

**W3C:** The World Wide Web Consortium provides standards for Web related technologies. These standards facilitate the interoperability of technologies used on the Web.

**WS-I:** The Web Services-Interoperability Organization is a consortium that works towards the provision of interoperable web services.

**WSIT:** The Web Services Interoperability Technologies provide for interoperability between Java and .NET web services.

**XML:** The Extensible Markup Language is a W3C recommendation for a subset of the Standard Generalized Markup Language .

# Appendix A

# WSDL file for insecure Calculator Web Service

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss
-wssecurity-utility-1.0.xsd" xmlns:wsp="http://schemas.xmlsoap.org
/ws/2004/09/policy"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://tham.org/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://tham.org/" name="CalculatorWSService">
<wsp:UsingPolicy></wsp:UsingPolicy>
<wsp:Policy wsu:Id="CalculatorWSPortBinding_add_WSAT_Policy">
<wsp:ExactlyOne>
<wsp:All>
<ns1:ATAlwaysCapability
xmlns:ns1="http://schemas.xmlsoap.org/ws/2004/10/wsat"
wsp:Optional="false">
</ns1:ATAlwaysCapability>
<ns2:ATAssertion
xmlns:ns3="http://schemas.xmlsoap.org/ws/2002/12/policy"
xmlns:ns2="http://schemas.xmlsoap.org/ws/2004/10/wsat"
```

```
ns3:Optional="true"
wsp:Optional="true">
</ns2:ATAssertion>
</wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>
<wsp:Policy
wsu:Id="CalculatorWSPortBinding_subtract_WSAT_Policy">
<wsp:ExactlyOne>
<wsp:All>
<ns4:ATAlwaysCapability
xmlns:ns4="http://schemas.xmlsoap.org/ws/2004/10/wsat"
wsp:Optional="false">
</ns4:ATAlwaysCapability>
<ns5:ATAssertion
xmlns:ns6="http://schemas.xmlsoap.org/ws/2002/12/policy"
xmlns:ns5="http://schemas.xmlsoap.org/ws/2004/10/wsat"
ns6:Optional="true" wsp:Optional="true">
</ns5:ATAssertion>
</wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>
<types>
<xsd:schema>
<xsd:import
namespace="http://tham.org/"
schemaLocation="http://146.231.121.204:8080/CalculatorWSService
/CalculatorWS?xsd=1">
</xsd:import>
</xsd:schema>
</types>
<message name="add">
<part name="parameters" element="tns:add"></part>
</message>
```

```
<message name="addResponse">
<part name="parameters" element="tns:addResponse">
</part>
</message>
<message name="subtract">
<part name="parameters" element="tns:subtract">
</part>
</message>
<message name="subtractResponse">
<part name="parameters" element="tns:subtractResponse">
</part>
</message>
<portType name="CalculatorWS">
<operation name="add">
<input message="tns:add">
</input>
<output message="tns:addResponse"></output>
</operation>
<operation name="subtract">
<input message="tns:subtract">
</input>
<output message="tns:subtractResponse">
</output>
</operation>
</portType>
<binding name="CalculatorWSPortBinding" type="tns:CalculatorWS">
<soap:binding
transport="http://schemas.xmlsoap.org/soap/http" style="document">
</soap:binding>
<operation name="add">
<wsp:PolicyReference URI="#CalculatorWSPortBinding_add_WSAT_Policy">
</wsp:PolicyReference>
<soap:operation soapAction="">
</soap:operation>
```

```
<input>
<soap:body use="literal">
</soap:body>
</input>
<output>
<soap:body use="literal">
</soap:body>
</output>
</operation>
<operation name="subtract">
<wsp:PolicyReference URI="#CalculatorWSPortBinding_subtract_WSAT_Policy">
</wsp:PolicyReference>
<soap:operation soapAction="">
</soap:operation>
<input>
<soap:body use="literal">
</soap:body>
</input>
<output>
<soap:body use="literal">
</soap:body>
</output>
</operation>
</binding>
<service name="CalculatorWSService">
<port name="CalculatorWSPort" binding="tns:CalculatorWSPortBinding">
<soap:address
location="http://146.231.121.204:8080/CalculatorWSService/CalculatorWS">
</soap:address>
</port>
</service>
```

# Appendix B

# WADL file for insecure Calculator Web Service

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<application xmlns="http://research.sun.com/wadl/2006/10">
    <resources base="http://146.231.121.204:8080
    /RestfulCalc/resources//resources">
       <resource path="add">
          <method name="GET">
            <request>

              <param name="i" type="xsd:string"
              style="query" required="true"/>
              <param name="j" type="xsd:string"
              style="query" required="true"/>
            </request>
            <response>
            <representation mediaType="application/xml"
            element="result" />"
            </response>
          </method>
       </resource>
       <resource path="subtract">
```

```
            <method name="GET">
              <request>
              <param name="i" type="xsd:string"
              style="query" required="true"/>
              <param name="j" type="xsd:string"


              style="query" required="true"/>
              </request>
              <response>
              <representation mediaType="application/xml"
              element="result" />"
              </response>
              </method>
        </resource>
    </resources>
</application>
```

# Appendix C

# Secure Calculator SOAP Header

```
<S:Header>
    <To xmlns="http://www.w3.org/2005/08/addressing"

       wsu:Id="5006">http://146.231.121.204:8080/
       CalcWS/SecureCalcService
    </To>
    <Action

       xmlns="http://www.w3.org/2005/08/addressing"
       wsu:Id="5005">
       http://me.org/SecureCalc/addRequest
    </Action>
    <ReplyTo xmlns="http://www.w3.org/2005/08/addressing"

       wsu:Id="5004">
          <Address>
          http://www.w3.org/2005/08/
          addressing/anonymous
       </Address>
    </ReplyTo>
    <MessageID
    xmlns="http://www.w3.org/2005/08/addressing"
```

```
wsu:Id="5003">
   uuid:dadf631f-37c6-4ddd-a2e8-c05892e9ccf2
</MessageID>
<wsse:Security S:mustUnderstand="1">
   <wsu:Timestamp
   xmlns:ns10="http://www.w3.org/2003/05/
   soap-envelope" wsu:Id="3">
      <wsu:Created>2007-06-18T11:43:31Z
      </wsu:Created>
      <wsu:Expires>2007-06-18T11:48:31Z
      </wsu:Expires>
   </wsu:Timestamp>
   <xenc:EncryptedKey xmlns:ns10=
   "http://www.w3.org/2003/05/soap-envelope"
   Id="5002">
      <xenc:EncryptionMethod
      Algorithm="http://www.w3.org/2001/04
      /xmlenc#rsa-oaep-mgf1p"/>
      <ds:KeyInfo
      xmlns:xsi="http://www.w3.org/2001/
      XMLSchema-instance"
      xsi:type="keyInfo">
      <wsse:SecurityTokenReference>
      <wsse:KeyIdentifier
      ValueType="http://docs.oasis-open.org
      /wss/2004/01/
      oasis-200401-wss-x509-token-
      profile-1.0#X509SubjectKeyIdentifier"
      EncodingType="http://docs.oasis-open.org/
      wss/2004/01/oasis-200401-wss-
      soap-message-security-1.0#Base64Binary">
      dVE29ysyFW/iD11a3ddePzM6IWo=
      </wsse:KeyIdentifier>
      </wsse:SecurityTokenReference>
```

```
   </ds:KeyInfo>
   <xenc:CipherData>
   <xenc:CipherValue>
   ixQT5Esd8rauBiMwfaaB3fz8Zd8Ywf1nZpP1g
   +04UI1klrTpGxy7yyDTRyMspNjvlgik5vrWNL
   5vDbX+OEAeyJyxSZ/dhiNrFLx6T4B4AoC7KLbF
   n8NN++LK+tF+hI+7vtKLcQx+rwEiTFVrmcvaMR
   tEK9wck5mbfJocZnahtIM=
   </xenc:CipherValue>
   </xenc:CipherData>
</xenc:EncryptedKey>
<xenc:ReferenceList xmlns:ns16=
"http://www.w3.org/2003/05/soap-envelope">

   <xenc:DataReference URI="#5008">
   </xenc:DataReference>
   <xenc:DataReferenceURI="#5009">
   </xenc:DataReference>
</xenc:ReferenceList>
<xenc:EncryptedData xmlns:ns10=
"http://www.w3.org/2003/05/soap-envelope"
Type="http://www.w3.org/2001/04/xmlenc
#Element" Id="5009">
   <xenc:EncryptionMethod
   Algorithm="http://www.w3.org/2001/04/
   xmlenc#aes128-cbc"/>
   <ds:KeyInfo
   xmlns:xsi="http://www.w3.org/2001/
   XMLSchema-instance"
   xsi:type="keyInfo">
   <wsse:SecurityTokenReference>
   <wsse:Reference ValueType=
   "http://docs.oasis-open.org/wss/
   oasis-wss-soap-message-security-1.1
```

```
   #EncryptedKey" URI="#5002"/>
   </wsse:SecurityTokenReference>
   </ds:KeyInfo>
   <xenc:CipherData>
   <xenc:CipherValue>
   cjQAvjeK6JkZgdfAHt9evNGIyM8sJONtrz0Dc
   Rk6dn08/THovGoq9BHc4yYGpQ7k8xfhG5DOwP
   ZPOqYOAjdeXyyyaCiw3ALV9Her5VElto3IYtv
   as606/+xpanTILo3wSVhejyOJ/3JlyP6gmz1D
   2X1/felN8VrXIAiGGl/FkLL/qh84XmgnMnUJi
   +K0xQyc+3uIIpG3eybbO0OcAX66uxaAGOkUoB
   UYjHn9/F1kmQb9nZaDx3cRM/Q19ZPX+wODc2N
   Zh7wPzpKJmdZhz0s1g9kB1+87CHT85EGauhZ
   4yC6n8Pu7hhG3SHX/h8Rg9NcuGIkupwMn1fx
   viwFoVXLs2s+jFnNKBgEpNCHBjyzWqyPgCtK
   Wp5p5w7g5dEuYacGQTT5DXzhW/qfdOBmOUqP
   VfLAr/uk3f5Yy0A9z9+y6fN/anIdc7vAG8AJ
   VZcq+twkLr2HMgLH88UhWX0FT0lQ5KukLPPi
   bX0kQm1QY5z6vQXCYsSO0Ulcfm2+wFNjgvYlI
   </xenc:CipherValue>
   </xenc:CipherData>
</xenc:EncryptedData>
<ds:Signature xmlns:ns10=
"http://www.w3.org/2003/05/soap-envelope"
Id="1">
   <ds:SignedInfo>
   <ds:CanonicalizationMethod Algorithm=
   "http://www.w3.org/2001/10/xml-exc-c14n#">
   <exc14n:InclusiveNamespaces PrefixList="wsse"/>
   </ds:CanonicalizationMethod>
   <ds:SignatureMethod Algorithm=
   "http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
   <ds:Reference URI="#5003">
   <ds:Transforms>
```

```
<ds:Transform Algorithm=
"http://www.w3.org/2001/10/xml-exc-c14n#">
<exc14n:InclusiveNamespaces PrefixList="S"/>
</ds:Transform>
</ds:Transforms>
<ds:DigestMethod Algorithm=
"http://www.w3.org/2000/09/xmldsig#sha1"/>
<ds:DigestValue>
bpXwHdieG5wnXGp25lrm1kBKJzc=
</ds:DigestValue>
</ds:Reference>
<ds:Reference URI="#5004">
<ds:Transforms>
<ds:Transform Algorithm=
"http://www.w3.org/2001/10/xml-exc-c14n#">
<exc14n:InclusiveNamespaces PrefixList="S"/>
</ds:Transform>
</ds:Transforms>
<ds:DigestMethod Algorithm=
"http://www.w3.org/2000/09/xmldsig#sha1"/>
<ds:DigestValue>
WdIQVaynkwqFa/LTJNuUmAHYF+k=
</ds:DigestValue>
</ds:Reference>
<ds:Reference URI="#5005">
<ds:Transforms>
<ds:Transform Algorithm=
"http://www.w3.org/2001/10/xml-exc-c14n#">
<exc14n:InclusiveNamespaces PrefixList="S"/>
</ds:Transform>
</ds:Transforms>
<ds:DigestMethod Algorithm=
"http://www.w3.org/2000/09/xmldsig#sha1"/>
<ds:DigestValue>
```

```
bppSweb++YZbNfVe3imIOas0YVg=
</ds:DigestValue>
</ds:Reference>
<ds:Reference URI="#5006">
<ds:Transforms>
<ds:Transform Algorithm=
"http://www.w3.org/2001/10/xml-exc-c14n#">
<exc14n:InclusiveNamespaces PrefixList="S"/>
</ds:Transform>
</ds:Transforms>
<ds:DigestMethod Algorithm=
"http://www.w3.org/2000/09/xmldsig#sha1"/>
<ds:DigestValue>
q3/TzdPB5IIrK6DMZ6RKASQ4d18=
</ds:DigestValue>
</ds:Reference>
<ds:Reference URI="#5007">
<ds:Transforms>
<ds:Transform Algorithm=
"http://www.w3.org/2001/10/xml-exc-c14n#">
<exc14n:InclusiveNamespaces PrefixList="S"/>
</ds:Transform>
</ds:Transforms>
<ds:DigestMethod Algorithm=
"http://www.w3.org/2000/09/xmldsig#sha1"/>
<ds:DigestValue>
3Zz/vjHwt9MlnkyPJijrC6lElRQ=
</ds:DigestValue>
</ds:Reference>
<ds:Reference URI="#3">
<ds:Transforms>
<ds:Transform Algorithm=
"http://www.w3.org/2001/10/xml-exc-c14n#">
<exc14n:InclusiveNamespaces
```

```
PrefixList="wsu"/>
</ds:Transform>
</ds:Transforms>
<ds:DigestMethod Algorithm=
"http://www.w3.org/2000/09/xmldsig#sha1"/>
<ds:DigestValue>
YSfcUDFkhCNFN4oCWXUZ8qZrr00=
</ds:DigestValue>
</ds:Reference>
<ds:Reference
URI="#5a6a4a91-6f9c-469e-9b06-dd6b24d71d9f">
<ds:Transforms>
<ds:Transform Algorithm=
"http://www.w3.org/2001/10/xml-exc-c14n#">
<exc14n:InclusiveNamespaces PrefixList="wsu wsse S"/>
</ds:Transform>
</ds:Transforms>
<ds:DigestMethod Algorithm=
"http://www.w3.org/2000/09/xmldsig#sha1"/>
<ds:DigestValue>
AksTWmnVTjcj4pxlCvMPMksDxQ4=
</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue>
FE3fvk+o3wCa7Jd5byueferAW+4=
</ds:SignatureValue>
<ds:KeyInfo>
<wsse:SecurityTokenReference
wsu:Id="1783316d-9d98-4b66-a09e-c542b92d4ebb">
<wsse:Reference ValueType=
"http://docs.oasis-open.org/wss/
oasis-wss-soap-message-security-
1.1#EncryptedKey" URI="#5002"/>
```

```
            </wsse:SecurityTokenReference>
            </ds:KeyInfo>
         </ds:Signature>
      </wsse:Security>
   </S:Header>
```