1 Introduction

1.1. Background and motivation

Web services provide a standard means of interoperation between different software applications running on diverse platforms and/or frameworks [1]. Figure 1 shows the service-oriented architecture of Web services. First, the service requester generates request message *X* in SOAP format according to the WSDL document obtained from the service provider or service broker (UDDI) (note that the WSDL is an XML-based language for describing Web services and how to access them). The service provider replies by verifying if the service requester is authorized to obtain the service according to its authentication policy. It then performs the operations specified in *X* and returns the execution results in another SOAP document, *R*. Web services employ a combination of XML [2], SOAP [3], WSDL [4], and UDDI [5] technologies.

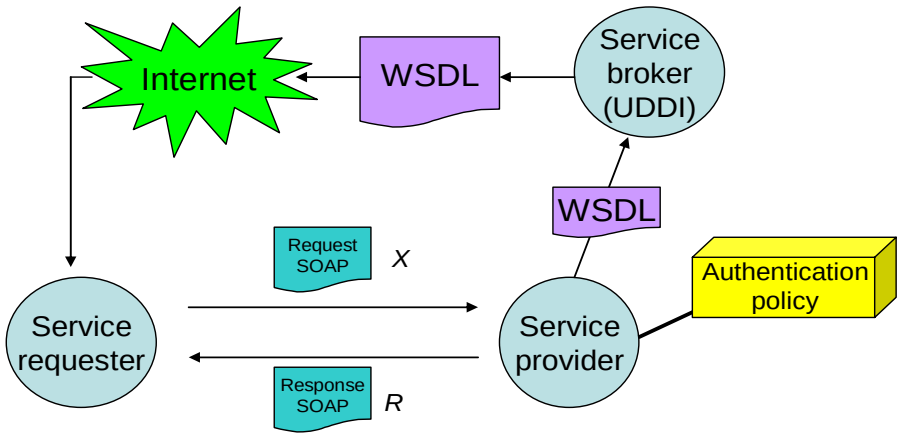

Figure 1: Service-oriented architecture of Web services

The Internet is a public network, and hence it is vital to protect against attacks and unauthorized access to sensitive information. The XML encryption working group of the World Wide Web Consortium (W3C) [6] delivered a recommendation specification for securing XML documents using XML encryption [7]. An encrypted

XML document specifies a process for encrypting data therein and representing the result in XML. The data may be arbitrary data, an XML element, or the content of an XML element. The W3C recommendation in [8] specifies the XML rules and syntax for processing digital signatures. Related work includes language support for securing XML documents, such as the document security language [9,10,11].

SOAP functions as a transport mechanism for XML messages by allowing programs running on one operating system to communicate with programs running on another, using the HTTP protocol and XML to exchange data. However, the original SOAP version of the W3C provided no security [3]. IBM, Microsoft, and Internet security vendor VeriSign developed Web Services Security (WS-Security) as a way for Web services to work with several different security models via SOAP extensions [12]. It allows applications to attach security data to the headers of SOAP messages, which can include security metadata like those found in the XML encryption and XML signature specifications [7,8]. The standard also uses security tokens—such as Kerberos authentication tickets or SAML assertions—that validate digital signatures, verifying that the sender has not been spoofed. WS-Security is generally used as the standard syntax for secured SOAP messages that contain digital signatures and encrypted XML elements.

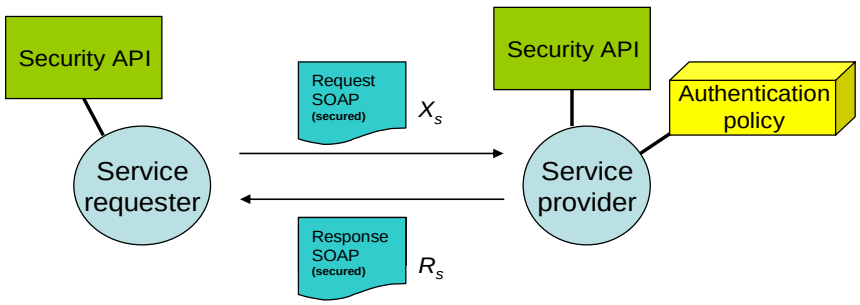

Figure 2: Hardwiring the security policy in the codes of the service requester and service provider

Figure 2 illustrates a scheme that employs WS-Security to secure the operation of Web services. The service requester and service provider invoke cryptographic algorithms in the security application programming interface (API). The security policy is hardwired in the codes of the service requester and service provider, and the request and response SOAPs, X_s and R_s, are secured in the form of WS-Security. Altering the security policy (e.g., the scope of encryption and signed data) requires changes to the programs in both the service requester and service provider, which is an obvious drawback. The JDeveloper environment developed by the Oracle Corporation employs this model for testing securing and adding self-signed key-pair integrity and encryption details to both the requester and provider of Web services so as to ensure that the service is used by only authorized users [13].

WS-SecurityPolicy provides a framework for defining security, privacy, and other policies on machines involved in Web services transactions [14]. The specification also allows a system to assess the requirements of policies, such as which type of authentication is needed, and is used by the service provider to publish the associated security policies. However, some technical details such as the locations and identities of the keys cannot be specified in WS-SecurityPolicy [15], and hence this information can only be obtained by prior negotiation between the service requester and service provider. This represents a type of implicit key definition.

1.2. The overview of the operation model

In this paper, we propose an operational model to support the security of Web services. In addition to satisfying the basic security requirements, including authentication, confidentiality, data integrity, and nonrepudiation [16], the proposed model supports security mechanisms such as element-wise encryption and temporal-based

element-wise digital signatures [9,10,11]. Furthermore, our model supports a flexible key specification scheme called *explicit key definition*, which can be used to define three different types of keys: static keys, dynamically selected keys, and keys applied to digital signatures[1]. The service requester does not need to negotiate with the service provider to determine the identity of the keys used. The proposed operational model is designed to reduce the cost of system development and maintenance in two ways: (1) by separating service implementation and specification of the security policy for Web services, and (2) by using a specially designed API to support the proposed operational mode.

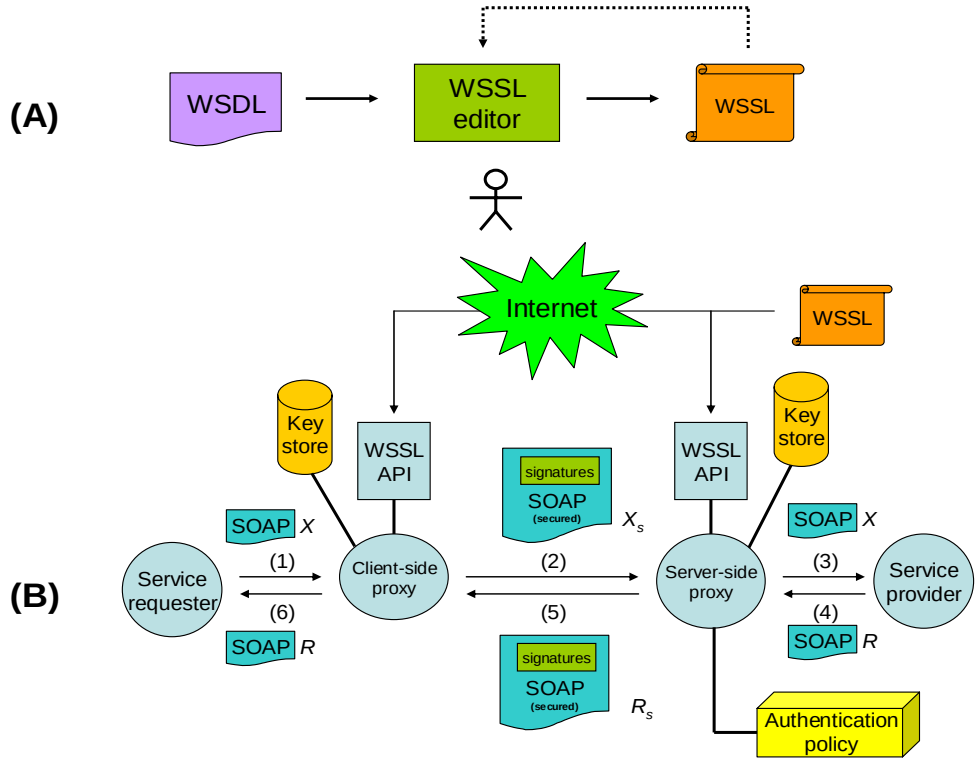

Figure 3: Proposed operational model

Assume that the WSDL is available and the service requester and service provider have been implemented, as shown in Figure 1. The operational model works as follows:

[1] Explicit key definition is explained in detail in Section 3.

First, the system administrator specifies the security policy in a WSSL (Web services security language) editor that reads the original WSDL document. This editor provides an interactive graphical user interface for setting up the security policy, such as the scopes of encryption and signed data, keys used, and cryptographic algorithms, and generates a document in the WSSL (see Figure 3A). The WSSL therefore constitutes the core of the proposed operational model.

The service requester and service provider then have to write the client-side and server-side proxy programs, respectively. The proxy programs invoke subroutines in the WSSL API to interpret the security policy stored in the WSSL document. The API provides a convenient way for the proxy programs to set up the required keys. Since the WSSL supports explicit key definition, the required keys can be identified without negotiation between the two proxy programs. After all the required keys are obtained, the proxy program can then instruct the WSSL API to secure or unsecure SOAP messages according to the instructions in the WSSL document. The WSSL API is designed so that the programmer does not have to understand the syntax or any details of the WSSL (see Figure 3B). There are six steps for securely invoking Web services in the proposed operational model:

Step 1: The service requester sends SOAP document X to the client-side proxy.

Step 2: The client-side proxy invokes routines in the WSSL API to generate secured SOAP document X_s.

Step 3: The server-side proxy receives X_s and invokes the WSSL API to unsecure it. The unsecuring process includes decrypting cipher data in X_s and verifying all the digital signatures embedded in X_s. The WSSL API supports obtaining the identity of the service requester from the embedded digital signatures according to the explicit key definition in the WSSL. It can then check the

obtained identity against its authentication policy. The unsecured document *X* is sent to the service provider.

Step 4: The response message is stored in SOAP document *R*, and this is sent to the server-side proxy.

Step 5: The server-side proxy invokes routines in the WSSL API to generate secured SOAP document R_s.

Step 6: The client-side proxy receives R_s and invokes routines in the WSSL API to unsecure it. In addition to decrypting the cipher data in R_s, the digital signatures embedded in R_s should be verified.

The WSSL is defined so as to increase the flexibility in setting the security policy and to separate service implementation and the setting of security policy. The WSSL is used to specify the security policy in the request and response SOAP documents. First, it has to describe the required keys and the cryptographic algorithms used. Second, it should support element-wise encryption and temporal-based element-wise digital signatures [9,10,11]. Element-wise encryption allows different portions in the SOAP documents to be encrypted using different keys and algorithms. Also, each encrypted entity may have different scopes in term of XML elements. There may be several digital signatures embedded in the SOAP document, which sign different portions of it. The digest message can be obtained before or after the encryption, which is referred to as the implementation of temporal-based element-wise digital signatures.

Another important feature supported by the WSSL is explicit key definition, which can be used to specify static keys, dynamically selected keys, and keys applied to digital signatures. A static key is usually the key of the service provider that can be determined statically during the construction of the WSSL document, whereas the

latter two key types are usually those of the service requester that cannot be determined because the service provider offers a many-to-one service model; that is the possible service requesters are dynamic. The service-oriented architecture of Web services almost always comprises multiple service requesters that are able to make an invocation to a single service provider (see Figure 4). Also, the allowed service requesters are always dynamic. The service provider has to check the identity of the service requester according to the authentication policy to determine if it is an allowable transaction. It is unreasonable to store the group information in the WSSL document, since this information may change. In our operational model, the authentication policy supports verification of the group information. With the dynamic key definition in the WSSL, the service requester can choose to use its private key to embed digital signatures in the SOAP message (i.e., X_s in Figure 3). It is obvious that the service provider needs to obtain the public key of the service requester in order to unsecure requested messages, with this public key being used to secure response SOAP document *R*.

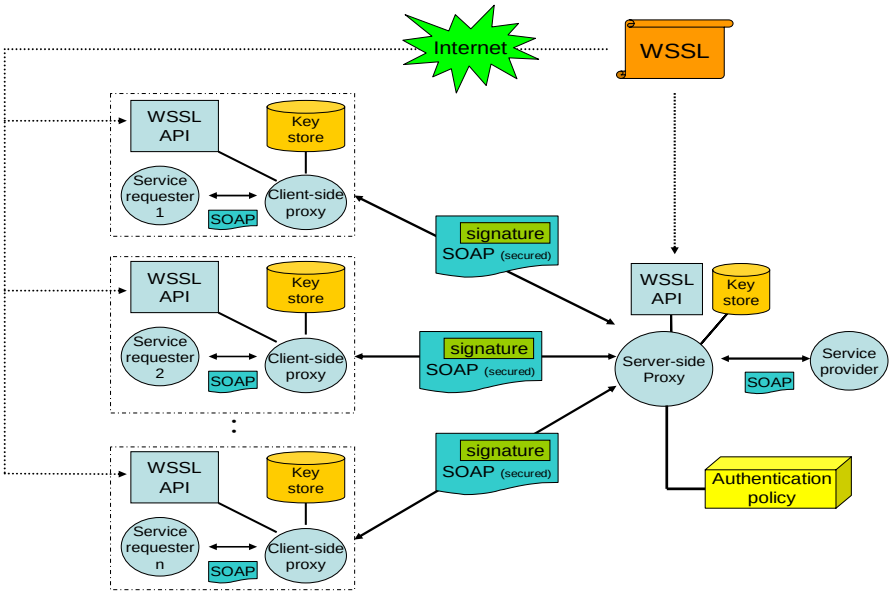

Figure 4: Multiple service requests to a single service provider (many-to-one service model)

In addition to the WSSL itself, we provide a facilities to support the proposed operational model. We design the WSSL API with the goal of minimizing the code that the programmer has to write in order to construct the client-side and server-side proxy programs. The WSSL API performs the actual parsing of the WSSL documents. The programmer can instruct the WSSL API to access the desired key stores (e.g., Java key store [23]). The occurrence of multiple errors during the operations of securing and unsecuring SOAP documents also needs to be handled. The securing system should be capable of enduring certain fault situations, and subsequently activating error handling routines to deal with them. For example, when one of the cryptographic keys is not available during decryption, the security application may still be able to decrypt most of the cipher texts. Since multiple endurable errors can be encountered during a single execution, the data structure of the WSSL API allows such errors to be stored in a vector. The programmer can design error handling routines based on an examination of the recorded errors. Finally, since a secured SOAP document may contain many digital signatures, the WSSL API should be able to verify all of them and report the individual results to the proxy program.

The remainder of this paper is organized as follows: Section 2 discusses related work, Section 3 describes the syntax of WSSL documents, Section 4 presents the syntax of secured SOAP documents, Section 5 presents the function of the WSSL API, Section 6 presents our implementation and experimental results, and conclusions are drawn in Section 7.