Policy-driven and Content-based Web Services Security Gateway

Zein Radwan, Camille Gaspard, Ayman Kayssi, and Ali Chehab

(Corresponding author: Ali Chehab)

Department of Electrical and Computer Engineering, American University of Beirut
Beirut 1107 2020, Lebanon (Email: chehab@aub.edu.lb)

(Received Feb. 19, 2008; revised May 5, 2008; and accepted June 13, 2008)

Abstract

Web Services are widely used to provide services and exchange data among business units, customers, partners and suppliers for enterprises. Although Web Services significantly improve the interaction and development of processes in the business world, they raise several security concerns, since they greatly increase the exposure of critical enterprise data. Web Services exchange data using SOAP messages that are based on the interoperable XML language. We have previously introduced XPRIDE as an enhanced security architecture for assuring confidentiality and integrity of SOAP messages. XPRIDE uses content-based encryption to secure SOAP messages based on their XML content, and depends on security policies to define the parts of the SOAP message that need to be encrypted. Security policies are defined by administrators for each Web Service that needs to be secured. This paper extends XPRIDE using a modular design approach to ensure extensibility, such that new modules can be developed and deployed to handle the security of different types of data. In addition, we show a new implementation of XPRIDE as a gateway capable of applying content-based security on attachments of SOAP messages, where a single gateway serves several web servers in a web farm. These new features significantly improve the security, scalability, and deployability of XPRIDE.

Keywords: Web services, SOAP, confidentiality, integrity, policy-based security, content-based security

1 Introduction

Web Services are becoming the most common way for enterprise applications to exchange data. This importance stems from the fact that Web Services are platform- and language-independent and easily discovered by clients. Enterprises deploy Web Services to be able to exchange data and share services with other enterprises, customers, partners and suppliers. Web Services are also needed for data collection, delivery, display, and processing. This has pushed the trend for enterprises to implement a Service-Oriented Architecture (SOA) in their enterprise networks. Although the Web-Services-based application-to-application architecture significantly improves the interaction and development of the processes in the business world, it raises many security concerns, since it increases the exposure of the critical and sensitive data that is being exchanged.

Web Services exchange information using the SOAP messaging protocol [1]. SOAP uses eXtensible Markup Language (XML) based documents to wrap the transmitted data. Several methods are used to ensure confidentiality by encrypting the sensitive data exchanged among Web Services. The encryption phase can be integrated into the application by modifying the code of the application itself to encrypt the content of the SOAP message. Encryption may also be achieved by relying on the transport layer security technologies, such as Secure Sockets Layer (SSL) [2] or Transport Layer Security (TLS) [3]; these technologies present a point-to-point security solution that provides confidentiality and integrity for the data exchanged. Security for Web Services may also be performed at the SOAP message level by using XML security techniques such as XML Encryption [4] and XML Digital Signature [5]. The Organization for the Advancement of Structured Information Standards (OASIS) presented the Web Services Security Specification (WS-Security) [6] as a way for Web Services to use different security models via SOAP extensions. This specification describes enhancements to SOAP messaging to provide message integrity and confidentiality by relying on existing specifications such as XML Encryption and XML Digital Signature. The specification also provides a general-purpose mechanism for associating security tokens with message content.

To avoid the unnecessary effort and delay of bulk encryption, PRIDE [7] was developed as an efficient content-based security solution [11] for protecting the privacy and integrity of Web traffic exchanged between enterprise application servers on the Internet and mobile wireless devices. PRIDE is a policy-driven security architecture that employs content-based encryption and hashing methodologies to secure network data based on sensitivity and relevance. XPRIDE [20] was suggested as a SOAP security solution for securing the XML SOAP messages exchanged over the network between SOAP processors. XPRIDE is a policy-driven security architecture that employs content-based encryption to secure the SOAP messages based on their XML content. XPRIDE performs content-based security depending on security policies that are defined by administrators to specify exactly the security required for each part of the data.

In this paper we significantly enhance the design and implementation of XPRIDE. The system is redesigned in a modular way to ensure extensibility and a clear separation of duties. Thus, security is handled by one module while other additional SOAP message-related tasks are provided by other, separate modules. Moreover, dividing the functionalities into modules enables the XPRIDE system to more easily adapt to new future requirements. The original XPRIDE was designed as a SOAP filter that intercepts SOAP messages going out and into a Web Service [20]. In this work, we show an additional implementation of the optimized and modular XPRIDE system as a reverse proxy gateway using a local Domain Name Server (DNS) and a HyperText Transfer Protocol (HTTP) listener. The use of reverse proxy also improves scalability and security since the XPRIDE gateway forms an additional layer of protection and security when used in front of hidden Web servers.

The rest of the paper is organized as follows. In Section 2, we review techniques used to secure Web Services. Section 3 presents previous work related to Web Services security. In Section 4, the general design of the XPRIDE system is shown. Section 5 presents the details of a prototype implementation of XPRIDE as SOAP filters. Section 6 includes the details of the other prototype implementation of XPRIDE as a reverse proxy. Conclusions are given in Section 7.

2 Overview of Web Services Security

2.1 Secure Sockets Layer (SSL)

SSL encrypts all data transmitted across a network and can thus prohibit eavesdropping and unauthorized access to Web Services and SOAP messages. This technology ensures point-to-point security by establishing a secure channel on top of the Transmission Control Protocol (TCP), and provides data integrity and confidentiality in addition to authentication. Point-to-point security techniques are appropriate for creating secure connections in which information can be transmitted directly. However, SOAP messages sent over the network may need to travel through numerous intermediaries before reaching their ultimate SOAP receiver. Thus, in a Web Services architecture, intermediaries can manipulate a message on its way to the receiver. But when using a transport layer security technology such as SSL, intermediaries will no longer be able to control the SOAP messages [8], since the message would need to be decrypted by the intermediary before being forwarded to the ultimate receiver using a new encrypted stream. In addition, SSL performs bulk encryption on the whole data (in this case the SOAP messages) as one stream, without separating sensitive data from the insignificant data that causes no threat whatsoever if exposed. Although most of the data in some cases need not be secured, SSL encrypts everything, thus causing additional delay and inefficiency in the system performance especially if significant amounts of data are transmitted. Therefore, protocols like SSL don't scale well to complex, high-volume transactions, like those in Web Services.

2.2 XML Encryption

XML Encryption provides security at the SOAP message level. XML Encryption describes the process for encrypting and representing the encrypted data in XML documents. The specification supports common encryption algorithms and techniques, and provides ways to encrypt all or just parts of the XML text in the message, by using XPath expressions to reach the XML elements that need to be encrypted. XML Encryption is efficient because information that is not confidential can be sent unencrypted.

2.3 XML Signature

XML Signature defines syntax and processing rules for representing digital signatures. XML Signatures provide integrity, message authentication, signer authentication and proof for non-repudiation of who created the message. Like XML Encryption, XML Signature is able to sign only specific parts of the XML file. This is relevant when the data is generated, edited, or viewed by several users, and when the integrity of specific portions of the data has to be ensured, but other portions may still be open to changes.

2.4 Security Assertion Markup Language (SAML)

SAML, developed by OASIS, defines a way to express security information in an XML format. SAML functions as a framework for exchanging authentication, attribute, and authorization assertions across multiple participants over the Internet using protocols such as HTTP and SOAP. Assertions provide proof of identity for users and computers, via SAML subjects, which contain identity-related information. In addition, assertions list transaction-related user information (such as credit limits for e-commerce). SAML can also indicate the authentication method that must be used with a message, such as a password, Kerberos authentication ticket, hardware token, or X.509 digital certificate.

2.5 WS-Security Family Specifications

To combine multiple security techniques and concepts together, Microsoft, IBM and VeriSign have developed the WS-Security specification that was submitted to OASIS to be standardized. The task was not to invent a new security mechanism, but rather to define ways to use the techniques that already existed in the Web Services world. WS-Security defines a set of SOAP security extensions that can be used to insure message content integrity and confidentiality, by applying existing standards and specifications such as XML Encryption and XML Signature among other techniques. What WS-Security adds to existing specifications is a framework to embed these mechanisms into a SOAP message through additional security headers. Therefore, WS-Security covers the use of a general-purpose technique to associate security tokens with messages. These tokens represent a collection of claims. Claims are the statements that a related client makes, for example name, identity, key, group, privilege, capability, etc. The security token is not limited to a specific type; the specification is designed to be extensible. For example, a client may provide proof of identity and proof that it has a particular business certification.

Other specifications that directly relate to security issues such as WS-SecurityPolicy, WS-Trust, WS-Privacy, WS-Authorization, and WS-Federation are developed based on WS-Security. In the protocol stack and right on top of WS-Security, we find the WS-Policy specifications (with its security attached WS-SecurityPolicy specification). WS-SecurityPolicy [9] specifies how to define security assertions that clearly state a Web Service's preferences and security requirements, such as the signature and encryption algorithms used, and the parts of the message (XML elements) that need to be secured. WS-SecurityPolicy builds on the foundation of WS-Policy that defines a general approach to specifying policies of all kinds for Web Services.

2.6 Web Services Firewalls

Traditional network firewalls are blind to Web Services traffic when TCP ports 80 and 443 are open to HTTP traffic, thereby allowing SOAP and XML messages to flow undetected into an enterprise internal network. Thus, a Web Services firewall is useful where a combination of a packet filter firewall and an application level firewall are combined. The packet filter firewall checks every packet and detects SOAP messages. Every SOAP message is then redirected to an application-level firewall that analyses the message based on predefined policies. Policies may be used for data integrity checks. After checks and countermeasures are applied, the SOAP message is routed to its destination. The Web Services firewall may be extended by several modules such as user access control, logging, accounting and load balancing modules.

3 Related Work

In this section we review work related to securing Web Services.

In [12], Bhargavan et al. indicate that despite the flexibility brought by the WS-Security specifications group, driving Web Services security from WS-SecurityPolicy is not necessarily the best solution. First, they argue that WS-SecurityPolicy drives low level mechanisms that build and check individual security headers although what is needed is a way to relate policies to more abstract, application-level goals such as message authentication or secrecy. Second, all XML files used in a SOAP-based system, including WS-SecurityPolicy files are vulnerable to XML rewriting attacks. Furthermore, an essential limitation of the policy language in their opinion is that it is stateless, meaning that its interpretation does not depend on previously-received messages. Thus, the authors propose in their work a new link language and two new tools to address these problems. The high-level link specification language is a simple notation that describes the required secrecy and authentication specifications for all messages flowing between SOAP processors, and could easily be generated from a simple user interface or a systems modeling tool. Their first tool compiles link specifications to WS-SecurityPolicy configuration files, while their second tool is an analyzer to check (prior to execution) whether the security goals of a link specification are achieved by a given set of WS-SecurityPolicy files and whether they are vulnerable to any XML rewriting attacks.

In [13], Fernandez proposes two security patterns for Web Services. First, a Security Assertion Coordination pattern that coordinates authentication and authorization using a Role-Based Control (RBAC) model for access to distributed resources; and second, a pattern for XML firewalls that filter XML messages according to institution policies. The second pattern ensures that a client can access a Web Service only if it is authorized by the policy and if the content of the message sent is considered to be safe. The policies for each application are centralized within the firewall that enforces the access control for the application, and checks the content and structure of the message. A Content Inspector is used to check the content of the XML messages sent from/to the application. The Content Inspector consists of a Harmful Data Detector and an XML Schema Detector. The Harmful Data Detector performs checks for harmful data embedded in the content of the message, while the XML Schema Detector checks the validity of the XML documents sent to the application, where the structure is validated through a database of valid XML schemas.

In [14], Khoo and Zhou recommend that a service provider should not choose security solutions randomly, given a variety of emerging Web Services security standards. However, they provide some guidance to assist service providers in choosing security solutions strategically by proposing a framework for matching the security standards to the business security goals and choosing the most suitable standard for each security requirement.

In [15], Cremonini et al. propose in their paper an integrated security model that combines Web Services and firewall security. They argue that the Web Services architecture has reached the phase where it is integrated with excellent message security standards although the network security community is strongly arguing against it since in their opinion, security holds back functionality. To handle the problem, the authors argue that firewalls should evolve to meet the characteristics of the Web Services architecture in order to combine the advantages of both solutions.

In [16], Swart et al. reviewed security literature and documented eight ways in which Web Services violate traditional assumptions about security and may expose corporate data to invasion if not properly addressed.

In [21], Damianou et al. present Ponder as a declarative, object-oriented language used to specify policies for the security and management of distributed systems. The Ponder language provides the components necessary to specify several types of policies, such as authorization policies, event-triggered obligation policies, refrain policies, and delegation policies.

PRIDE [7] is a security architecture that provides confidentiality and integrity for the Web traffic sent between wireless handheld devices and enterprise application servers. PRIDE is a policy-based security solution that is configured to provide the security services according to the content and sensitivity of the network data in a Web transaction, which gives PRIDE considerable performance gains over bulk encryption protocols such as SSL. PRIDE relies on a security policy to define the content that needs to be secured. The policy configuration is stored in an XML-formatted document that is developed for each web application secured by PRIDE. This policy is used by a server security engine while a compacted version is used on the client side to reduce the network traffic and avoid any performance degradation on the client due to XML parsing.

The original XPRIDE [20] acts as a SOAP filter that performs security operations in a transparent manner to the sender and receiver, and hides the processing details and therefore avoids the application-level security complexity. This approach avoids the unnecessary bulk encryption carried out by transport layer security. The advantage that this solution has over the existing content-based security provided by WS-Security, is that it supports the ability to secure specific pattern occurrences in the SOAP messages exchanged, while WS-Security, which relies on XML Encryption, enables whole XML element encryption, although in many cases only parts of the SOAP response element need to be encrypted. XPRIDE applies security dynamically on the data without the need to specify (at design time) the parts that need to be encrypted as in WS-Security. Furthermore, the security policy proposed in XPRIDE enables the use of content-based security on the attachments appended to SOAP messages, where the policy specifies the exact parts of the attached files that should be encrypted, without the need to encrypt the whole attachment. XPRIDE was first implemented using the SOAP filters provided by the WSE class library to access and secure the raw SOAP messages based on the content relevance and sensitivity of the data included.

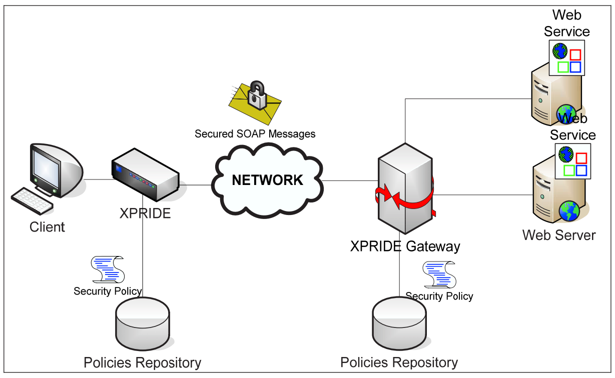


Figure 1: XPRIDE high-level architecture

4 Design and Architecture

An abstract view of the major components of the security architecture and their interrelationships is shown in Figure 1. XPRIDE is designed as an XML gateway that intercepts the SOAP messages exchanged between Web Services and their clients. Thus, it is located as a front-end to the Web server that it handles, where it intercepts all the traffic headed to that Web server.

XPRIDE provides content-based confidentiality and integrity for SOAP messages and attachments. Moreover, it is designed to provide a number of other XML security features related to XML content, such as XML content inspection, XML schema validation and XML access control. This is in addition to other XML functions not related to security such as XML transformation and XML compression. The framework is designed to be scalable and open to adapt to new features, where the functionality is divided into modules. Thus, in order to add a new feature to XPRIDE, it is sufficient to add the new module that conforms to the specification and handles this feature. An abstract view of the main components that make up XPRIDE framework is shown in Figure 2.

4.1 HTTP Handler

XPRIDE is placed as a front-end to the Web server and intercepts all the Web traffic coming to this Web server.

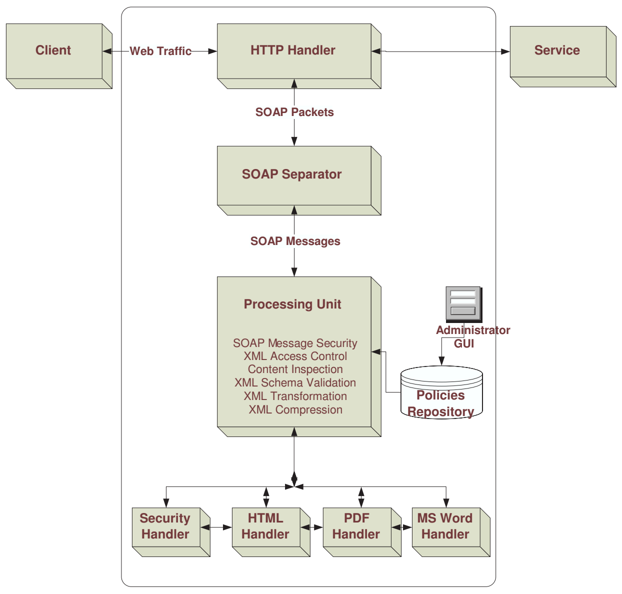


Figure 2: XPRIDE framework main structure

Thus an HTTP Handler unit is needed to handle all the HTTP transactions and deal with the HTTP message format, as it also separates the non-SOAP packets from the SOAP packets that need to be further processed.

4.2 SOAP Separator

The SOAP Separator is responsible for parsing and extracting the XML SOAP messages out of the HTTP message streams, where the rest of the processing will be performed later to the XML SOAP message itself.

4.3 Processing Unit

The Processing Unit represents the core of XPRIDE. It is responsible for performing content-based security and other XML-related functions on the SOAP messages and attachments. To ensure scalability, XPRIDE processing unit functions are divided into modules, where each module provides a specific function, with possible collaboration among modules. The major Security Handler module is responsible for ensuring confidentiality and integrity of the SOAP messages, by applying digital signatures and content-based encryption on the SOAP message and attachments. In order to decide which parts of the SOAP message content need to be secured; XPRIDE relies on the security configurations that are specified in a Policy File.

Separate modules may be included to handle the different types of attachment files (such as Microsoft Word documents, PDF files, HTML files) This modular design allows the set of features to be extended to handle various file types.

Other modules are also included to provide a number of XML security functions such as XML access control, content inspection, and XML schema validation, in addition to other non security-related operations such as XML transformation and XML compression.

4.4 Policies Repository

The repository is the persistent storage where all the policies are maintained. The policy definition controls the behavior of the processing unit, and represents the requirements stated by the system administrators.

4.5 The Security Policy and Scope

The Security Policy controls the overall security behavior of XPRIDE. The policy configuration is stored in an XML-formatted document for each Web Service secured by XPRIDE, and is used by the XPRIDE Security Engine on the sender side. A compacted version of the policy, presented as a reference table that indicates the locations of the encrypted parts, is included with the secured message in order for the client to be able to decrypt and validate the received data.

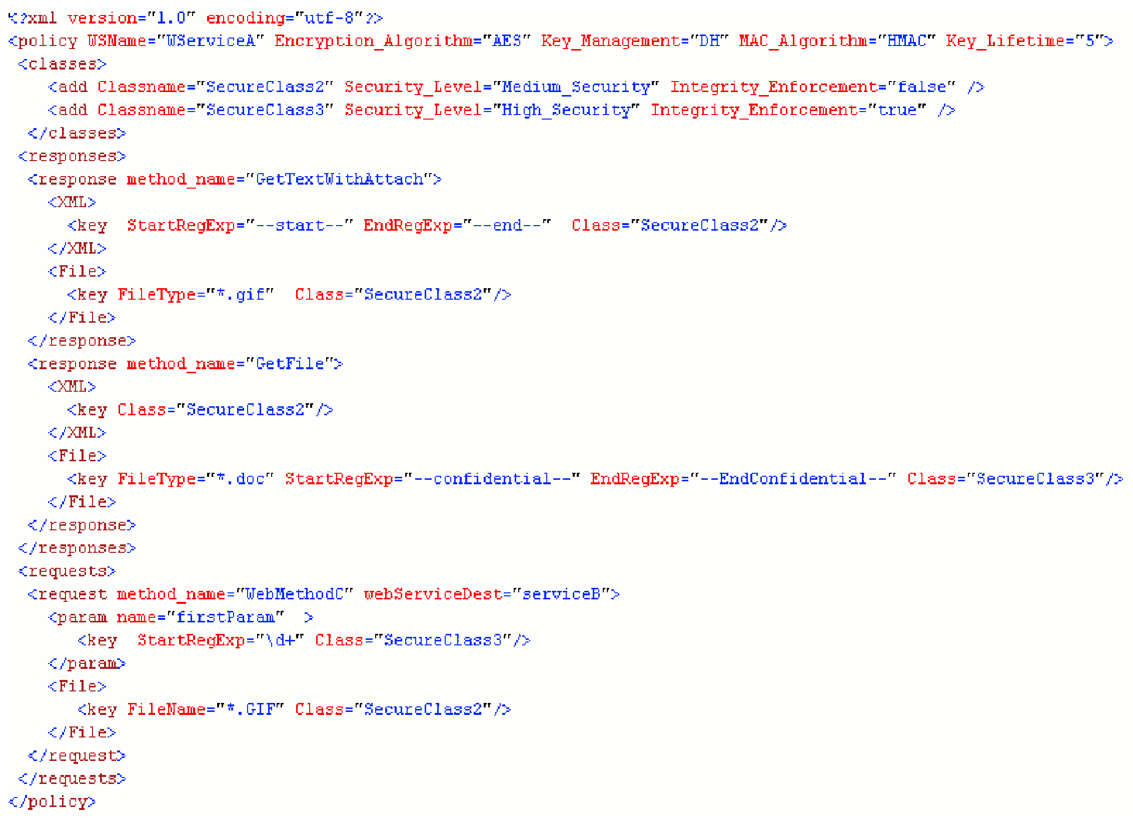
The Policy is divided into two main parts (see Figure 3): the first part specifies security-related attributes and parameters, while the second deals with how security mechanisms are applied to the SOAP message. The security-related attributes specified in the first part of the policy control the behavior of the Security Engine regarding how confidentiality, integrity and key management are performed. The attributes are the encryption algorithm, the hashing algorithm, the key management algorithm, and the session-keys life time.

The second part of the policy is divided into three sections. The first section specifies the security classes that are to be used by the policy. Each class holds a set of security specifications that can be applied later to different parts of data by simply referring to the name of the specific class that holds the data without the need to repeat the specifications again. These security classes are characterized by the following attributes: class name, security level, and integrity enforcement.

XPRIDE supports three security levels: a High Security level equivalent to an AES [10] 256-bit key length; a Medium Security level equivalent to an AES 192-bit key length; and a Low Security level equivalent to an AES 128-bit key length. The three keys are derived from one key that is initially generated during the key exchange phase. The integrity enforcement attribute in the Policy specifies if data integrity (using digital signatures) needs to be applied.

The second and third sections of the Policy deal with the SOAP message content. These sections specify the security level and scope for each part of the message, by assigning a certain security class for the part of the message that needs to be secured.

The body of a SOAP message can be carrying either the set of parameters needed to invoke a Web method,



```
<?xml version="1.0" encoding="utf-8"?>
<policy WSName="WServiceA" Encryption_Algorithm="AES" Key_Management="DH" MAC_Algorithm="HMAC" Key_Lifetime="5">
 <classes>
    <add Classname="SecureClass2" Security_Level="Medium_Security" Integrity_Enforcement="false" />
    <add Classname="SecureClass3" Security_Level="High_Security" Integrity_Enforcement="true" />
  </classes>
 <responses>
  <response method_name="GetTextWithAttach">
    <XML>
      <key  StartRegExp="--start--" EndRegExp="--end--"  Class="SecureClass2"/>
    </XML>
    <File>
      <key FileType="*.gif"  Class="SecureClass2"/>
    </File>
  </response>
  <response method_name="GetFile">
    <XML>
      <key Class="SecureClass2"/>
    </XML>
    <File>
      <key FileType="*.doc" StartRegExp="--confidential--" EndRegExp="--EndConfidential--" Class="SecureClass3"/>
    </File>
  </response>
 </responses>
 <requests>
  <request method_name="WebMethodC" webServiceDest="serviceB">
    <param name="firstParam"  >
       <key  StartRegExp="\d+" Class="SecureClass3"/>
    </param>
    <File>
       <key FileName="*.GIF" Class="SecureClass2"/>
    </File>
  </request>
 </requests>
</policy>
```

Figure 3: Example of XPRIDE policy

in case the message is a SOAP request, or the response content in case the message is a SOAP response. In order to cover these two cases, and knowing that a policy is assigned to one Web Service, the second section contains the information needed to secure the responses leaving this Web Service, while the third section contains the information needed to secure the requests made by this Web Service.

Since a Web Service offers a number of methods that can be invoked, the second section (responses) may contain an entry for each Web method that is distinguished by name. The third section (requests) includes entries for the methods that belong to other Web Services that may be requested and invoked by this Web Service, and whose parameters need to be secured. In this case, the request is distinguished by the requested method name and the name of the Web Service it belongs to. Therefore, in case XPRIDE is installed at the client side to secure the outgoing SOAP messages and this client is not a Web Service itself, the policy file at the corresponding XPRIDE does not need to include any entries in the "responses" section since responses are never sent by this client.

The data exchanged between a Web Service client and server includes the data that is serialized and enclosed inside the SOAP message, in addition to any attachment files. Therefore, the policy tag "XML" declares that the security specifications should be applied on the SOAP message, while the policy tag "File" declares that the security specifications should be applied on the SOAP attachments. Accordingly, data can be identified using the following three mechanisms: filetype, regular expression, or filetype and regular expression.

Filetype is used to specify the type of the SOAP attachment file that needs to be secured. It can be any valid file type extension, or an asterisk (*) for all files. Regular expressions (REs) are used to match and secure sensitive patterns that may appear inside the SOAP message. REs allow securing specific pattern occurrences in the SOAP messages exchanged, that are embedded inside the XML elements. The WS-Security specification on the other hand, relies on XML encryption to accomplish content based encryption on the tag level, where XML encryption relies on XPath expressions to reach a specific XML element; thus, the minimum part that WS-Security can reach is an XML element inside the SOAP message. However, using XPRIDE security policy, the data sent can be any text, and by using REs in the security policy we are still able to encrypt specific pattern occurrences where the regular expressions specified in the security policy are matched against the SOAP message, and the matches found are encrypted using the appropriate settings.

The XPRIDE Security Engine, after finding a match, encrypts the entire data where the match occurred.

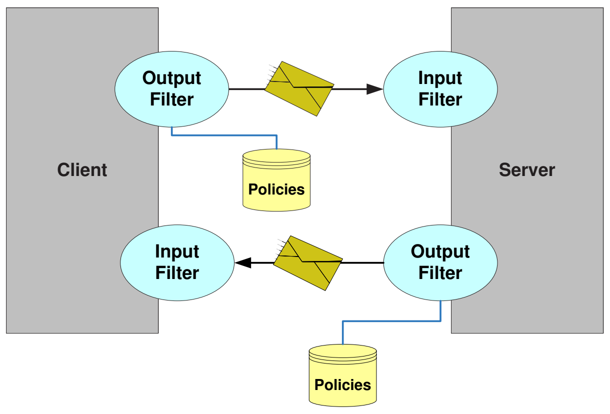


Figure 4: XPRIDE filter implementation structure

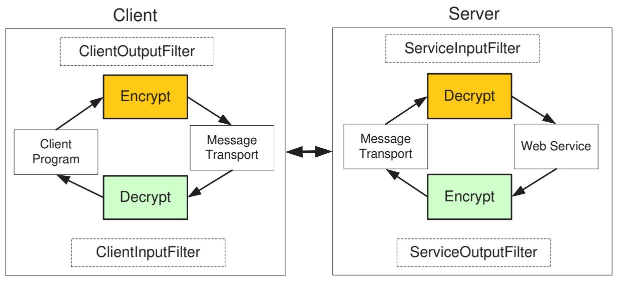


Figure 5: SOAP filters in XPRIDE implementation

"StartRegExp" is used to match a pattern occurrence, and is used with "EndRegExp" to match and secure all data that falls between these two regular expressions. When an entry is declared in the policy, but no regular expression is specified in it, it means that the whole entry is to be secured and no content-based matches are applied. File type and regular expressions are combined together in a policy to find and secure an occurrence of a specific pattern in a file that is being attached; an example is securing a specific match in an HTML or a Microsoft Office Excel file. This way, the security policy enables content-based security application on the attachments appended to SOAP messages as well. All the above identification mechanisms are used to specify the encryption boundaries and security levels that are to be applied when enforcing the policy.

4.6 Policy Caching

Policy resolving and caching are important features that are included in the XPRIDE design. Caching improves the policy loading time by using a compacted binary copy of the security policy instead of parsing it every time it is requested. However, due to the possible modification of the policy configuration by administrators, a hash is used to make sure that the policy information in the cache is always up-to-date.

4.7 SOAP Confidentiality and Integrity

The Security Engine is the component responsible for providing data confidentiality and integrity based on the sensitivity of the data. To support a flexible encryption scheme, XPRIDE identifies the content to be secured in the security policy and provides a filter through which the Web Service and its client can communicate and interact securely. Application developers need not be concerned about embedding security functions into the application itself and are only required to supply the Security Engine with identification of the data to be secured, how this data is to be secured and to what degree. It is the responsibility of the XPRIDE Security Engine on the sender side to perform the encryption and hashing operations on the data and to construct the secure response or request based on the incoming SOAP message and the security policy.

After the secured SOAP message is received, the Security Engine on the recipient side intercepts the received secure messages, performs the necessary integrity verification and decryption operations based on the reference table sent along with the secured message, reconstructs the original message and delivers it to the recipient.

The following steps summarize the overall request/response model of XPRIDE, both on the client side and on the service side. On the service side, XPRIDE (see Figure 4 and Figure 5):

1) Receives an incoming request from a client
2) Decrypts and validates the request depending on the attached reference table
3) Forwards the request after reconstructing it to the appropriate Web Service
4) Gets the generated response from the Web Service
5) Checks the policy store to see if a policy exists for the corresponding Web Service, and resolves the security settings by parsing the security policy
6) Applies confidentiality and integrity settings on the generated response
7) Attaches to the response an encrypted version of the reference table that indicates what parts of the message were encrypted at what locations, and
8) Forwards the response back to the client.

On the other hand, XPRIDE on the client side:

1) Gets a request from the client
2) Checks the policy store to see if the request should be secured, and resolves the security settings by parsing the corresponding security policy

3) Applies confidentiality and integrity settings on the request as indicated by the policy

4) Attaches to the request an encrypted version of the reference table

5) Sends the request to the appropriate Web Service

6) Waits for the response to arrive

7) Decrypts and validates integrity of the incoming response depending on the attached reference table

8) Delivers the plain response to the client.

5 XPRIDE Implementation as a SOAP Filter

XPRIDE is implemented as a SOAP filter using the Microsoft .Net 2.0 Framework with Visual Studio 2005, the WSE 2.0 (Web Services Enhancement) class library and IIS 5.1.

The implementation relies mainly on using the SOAP filters model, provided by the WSE class library to access the raw SOAP messages exchanged over the network between Web Services and their clients. The output filters intercept the SOAP messages going out of an application, while the input filters intercept the incoming SOAP messages, as depicted in Figure 4. Therefore, SOAP filters seem as the ideal candidates to implement custom security for Web Services, by applying XPRIDE security techniques on the outgoing SOAP messages that are being intercepted by these filters.

The general concept is depicted in Figure 5. The Output Filter on the sender side:

1) Intercepts the outgoing SOAP message

2) Parses the security policy relevant to the sender if it exists in the security policies repository

3) Parses the SOAP message to be sent

4) Defines the parts to be encrypted based on the policy

5) Applies encryption by using classes from the XPRIDE class library (XPRIDELib)

6) Combines the compacted security policy, after encrypting it, with the secured message to be sent in order for the ultimate receiver to be able to decrypt the received SOAP message.

The Input Filter on the receiver side:

1) Intercepts the incoming SOAP message

2) Extracts the reference table that represents the compacted security policy in case the message has been secured

3) Parses the secured SOAP message

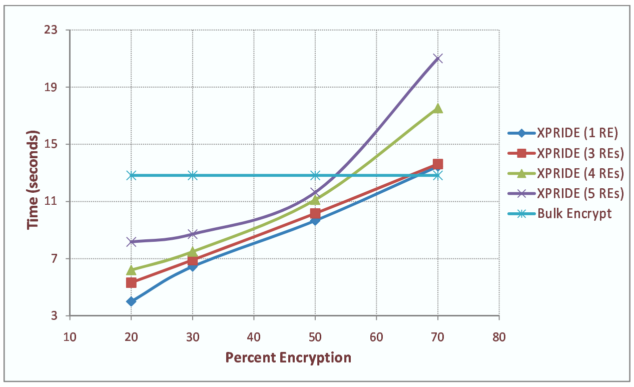


Figure 6: XPRIDE SOAP filter implementation results

4) Decrypts the secured parts

5) Generates the original message that will be delivered to the ultimate receiver.

5.1 Testing and Results

The tests were performed on a PC with the following specifications: DELL Inspiron 510 with 1.6 GHz Centrino CPU, 512 MB RAM and 40 GB hard drive. The operating system used was Windows XP Professional SP2.

The prototype XPRIDE implementation was tested on a sample data file of size 11 MB. In order to compare the performance of content-based security to the performance of bulk encryption, we implemented separate testing filters, both on the client and on the server sides. These filters simply encrypt and decrypt the content of every SOAP message to simulate a bulk encryption security system.

The first test was derived assuming no encryption at all; the time in this case is considered to be a reference time, which we refer to as t_0. Then, the same test was repeated assuming bulk encryption by using the bulk encryption testing filters. The additional time in this case, due to the bulk encryption operations, is the difference value t_1 = time taken with bulk encryption $-t_0$.

XPRIDE's content-based encryption was tested next, where the policy was repeatedly written to gradually increase the percentage of the data that needs to be secured. This percentage was varied as follows: 20%, 30%, 50%, and up to 70% of the data. For each percentage value, one regular expression is used in the policy. The data to be encrypted is enclosed between two distinctive words in the expression, which denote the start and end of the sensitive data. To study the effect of increasing the number of regular expressions in a policy on the performance of XPRIDE, the number of regular expressions used in the policy was incremented while keeping the overall percentage of the encrypted data constant. The number of regular expressions was increased to 3, 4, and 5. For every test, we compute t_2 = time taken with content-based encryption $-t_0$.

Table 1: XPRIDE time performance comparison (times are in seconds)

	1 Reg. Exp.	3 Reg. Exp.	4 Reg. Exp.	5 Reg. Exp.	No Encr.	Bulk Encr.
20% Encr.	6.020	7.342	8.224	10.193	2.017	14.845
30% Encr.	8.464	8.926	9.512	10.741		
50% Encr.	11.681	12.187	13.138	13.656		
70% Encr.	15.468	15.636	19.550	23.029		

Tests were repeated several times and the average values were recorded. The values of the averaged times (t_2) using XPRIDE content-based encryption filters, for the different cases listed above, are summarized in Table 1. Figure 6 compares the bulk encryption time with the time of content-based security for different data percentages, when using one, three, four, or five regular expressions in the security policy.

By examining the results shown in Table 1 and Figure 6, we can conclude that content-based Web Services security, as performed by XPRIDE, remains efficient as long as the percentage of the data that needs to be encrypted in relation to the whole data does not exceed 50%. Since, in most cases less than 20% of the Web data traffic is classified as sensitive and needs to be encrypted, XPRIDE is very efficient in such cases, and can reduce the encryption overhead time by a factor of 2.5. This can be the case in many Web Services, such as music download, image download, and news and forecast Web Services.

5.2 Comparison with WS-Security

To compare the performance of XPRIDE with that of WS-Security, we propose the following business scenario. A large company headquarters connects to each of its branches using Web Services to synchronize 11 MB of data and keeping it up-to-date. The Web Services that the business uses are described in Table 2.

The Web Services should transmit data securely since they may contain sensitive information. The first scenario is to use WS-Security and WS-SecurityPolicy to secure the traffic of these Web Services. In this scenario, the policy defines the XML elements that should be encrypted statically. Thus, by using the WS-SecurityPolicy, and because WS-Security pays no attention to content, all the traffic sent by Web Service A should be encrypted even though it may contain non-sensitive documents that need no encryption such as images and catalogs. Also, no part of the traffic sent by Web Service B is to be encrypted because it includes international exchange rates that are public data. Finally, a fraction of 35% of the traffic sent by Web Service C is secured: this percentage represents the XML elements that are defined as sensitive in the policy. As a result, the percentage of the encrypted data in the WS-Security scenario will be around 69% (see Table 3.)

XPRIDE, on the other hand, allows the encryption to be dynamically applied depending on the sensitivity of the content, using regular expressions. We assume therefore that the sensitive documents comprise 25% of the overall documents sent by Web Service A, and 30% of the HR traffic sent by Web Service C. This reduces the fraction of encrypted data in XPRIDE to 22.5% of the overall traffic (see Table 3.) From Figure 6, and considering the case of one regular expression, we can see that the time taken by XPRIDE (at 22.5%) for 11 MB is 4.5 seconds. The time taken by WS-Security corresponds to a 69% percentage, and is around 13 seconds. Therefore, XPRIDE takes 65% less time to secure the 11 MB of Web Services traffic described above. Using the same approach, we estimated the percentage of time decrease for three, four, and five regular expressions. The results are shown in Figure 7.

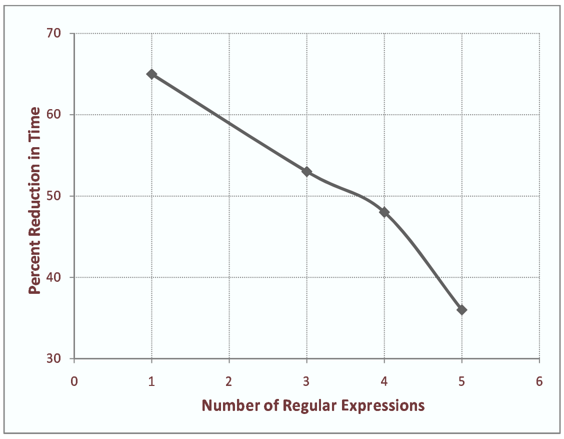


Figure 7: XPRIDE and WS-Security comparison

6 XPRIDE Implementation as a Reverse Proxy

The implementation of XPRIDE using SOAP filters is not the only approach to utilize the XPRIDE architecture. We have in fact implemented another XPRIDE prototype using a reverse proxy. Such a proxy receives Web client requests on behalf of a protected Web server, gets the response from the (or one of the) Web servers, and sends the response back to the client. The XPRIDE implementation in this case inserts code that is executed during the phase between getting the response from the Web server and sending it back to the client. The response stream is

Table 2: Web Services description

	Service description	% of total traffic
Web Service A	Transmits documents such as purchase orders, invoices and reports	60
Web Service B	Continuously transmits the currency exchange rates	15
Web Service C	Performs human resources related transactions	25

Table 3: XPRIDE and WS-Security comparison

	% Encrypted WS-Security	**% Encrypted XPRIDE**
Service A	100	25
Service B	0	0
Service C	35	30
Total % Encrypted	69	22.5
Time Taken (seconds)	13	4.5

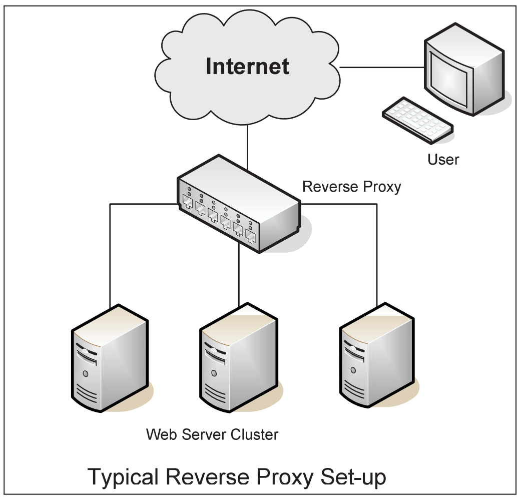


Figure 8: Reverse proxy structure

passed to the XPRIDE module to be secured before it is sent back to the client.

6.1 Reverse Proxy

A reverse proxy is typically utilized to reduce the load on a busy Web server by using a proxy between the server and the Internet. The proxy responds on behalf of the backend server. Figure 8 shows a typical reverse proxy structure. From an outside client's point of view, the reverse proxy is the actual HTTP server [19]. The benefit of using a reverse proxy includes improved scalability, improved security since the proxy server forms an additional layer of defense and protection for the hidden Web servers, improved load distribution, and caching of static content.

When a client browser makes an HTTP request, the domain name system (DNS) will point the request to the reverse proxy machine instead of the actual Web server. In the prototype implementation of XPRIDE as a reserve proxy, the requests initiated by clients to Web Services that reside on the protected Web server are redirected by the DNS to an HttpListener process on the proxy instead. The HttpListener forwards the client's request to the appropriate Web server and waits for the response. In this fashion, one XPRIDE reverse proxy may be shared by multiple Web Services running on one or more Web servers. After receiving the response, the data stream is passed as is to an XPRIDE module that applies the suitable transformations on it. Then, the resulting secured stream is sent back to the client. Figure 9 shows the structure employed in this implementation of XPRIDE.

6.2 XPRIDE Security Module

Once the response stream is received by the HttpListener, it is passed to the XPRIDE Security Module in order to perform the security operations. The stream is initially transformed to its Direct Internet Message Encapsulation (DIME) [17] records structure. DIME is used to send SOAP messages along with additional attachments, such as binary files, XML fragments, and even other SOAP messages, using standard protocols like HTTP [18]. In case a DIME attachment is added, the SOAP response message becomes the first DIME record, and all the other records will follow the SOAP record.

The XPRIDE Security Module is used to secure the DIME records that follow and which represent the attachment files. For scalability, each attachment file type (jpg, pdf, txt, etc.) is handled by its own security handler to apply content-based encryption. In this prototype implementation, only text attachment files are considered for content-based security, while for other types of files the decision is taken upon the policy to encrypt or not encrypt the entire file. The text files are considered for content-based security by relying on the regular expressions defined in the security policy.

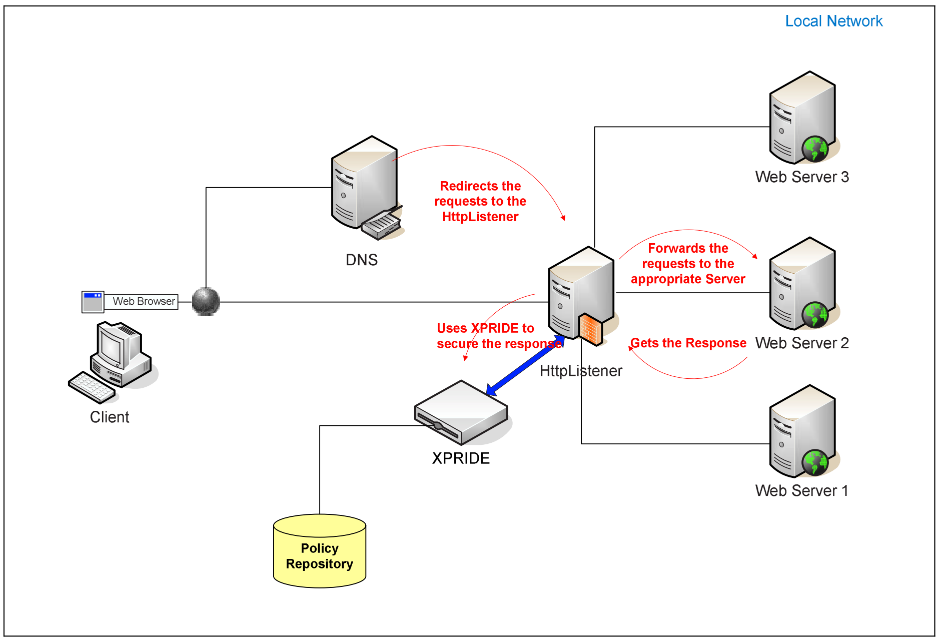

Local Network
Web Server 3
Redirects the requests to the HttpListener
DNS
Forwards the requests to the appropriate Server
Web Server 2
Web Browser
Uses XPRIDE to secure the response
Gets the Response
HttpListener
Client
XPRIDE
Web Server 1
Policy Repository

Figure 9: XPRIDE implementation as a reverse proxy

After securing all the records, the records are serialized again into a stream using a DIMEWriter process, and the resulting stream is returned to the HttpListener which sends it in turn back to the client. A reference table that indicates the number of DIME records that have been secured along with the security level for each record is formed and sent to the client as a new XPRIDE HTTP header.

In order to perform verification and decryption operations on the client, a plug-in decryption module was integrated with the client application for testing purposes. This module is responsible for receiving the secured stream, parsing it into DIME records, and applying the decryption and verification operations needed on the DIME records.

6.3 Testing and Results

The tests for the reverse proxy implementation of XPRIDE were performed on a client PC with the following specifications: 1.6 GHz Pentium 4 CPU, 256 MB RAM, and 40 GB hard drive. The server PC has the following specifications: 1.6 GHz Centrino CPU, 512 MB RAM and 40 GB hard drive. The operating system used was Windows XP Professional SP2 on the two computers.

The objective of the tests in this implementation was to examine the impact on latency caused by applying XPRIDE security on attachment files, since the first implementation (using SOAP filters) did not consider attachment files. The testing operations compare the performance of content-based security achieved by XPRIDE with the performance of bulk encryption and that of WS-Security.

The first test was derived assuming no encryption at all; the time in this case is considered to be a reference time, which again we refer to as t_0. Then, the same test was repeated assuming bulk encryption. The additional time in this case, due to the bulk encryption operations, is t_1 = time taken with bulk encryption $-t_0$.

The same test was repeated assuming WS-Security encryption. To simulate the WS-SecurityPolicy, XPRIDE policy is configured to encrypt the body element of the SOAP message without encrypting the attachments, since the implementation of the WS-SecurityPolicy, provided in WSE does not support attachment security. The additional time taken is t_2 = time taken with WS-Security encryption $-t_0$.

XPRIDE content-based encryption was tested next, where the policy was repeatedly written to gradually increase the number of attachments that need to be secured from one to four attachments of the following different types: image, text, pdf, and Microsoft Office Word document with the following sizes, respectively: 1.2 MB, 850 KB, 1.1 MB, and 1.2 MB. The number of regular expressions used in the policy to secure the SOAP message was increased from zero to two regular expressions where each entry corresponds to almost 25% of the data. For every test, we compute t_3 = time taken with XPRIDE content-based encryption $-t_0$.

Tests were repeated several times and the average values were recorded. The values of the averaged times (t_3) using XPRIDE content-based encryption, for the different cases listed above, are summarized in Table 4.

Figure 10 compares the XPRIDE times (t_3) with bulk encryption (t_1) and WS-Security (t_2) for different number of attachments, when one or two regular expressions,

Table 4: XPRIDE proxy performance comparison (times are in seconds)

	1 Reg. Exp.	2 Reg. Exp.	3 Reg. Exp.	No Encr.	WS-Security	Bulk Encr.
1 attachment	6.184	8.036	9.292			
2 attachments	6.472	8.756	9.908	4.044	8.932	11.960
3 attachments	7.200	9.056	11.680			

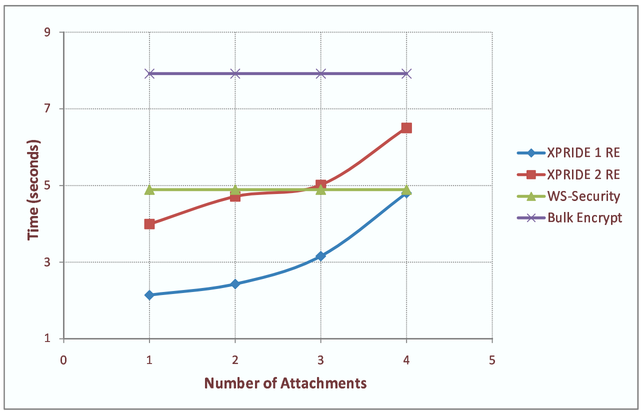


Figure 10: XPRIDE reverse proxy results

respectively, are used in the security policy.

The results presented in Table 4 and Figure 10 show that XPRIDE content-based encryption on the attachment results in a performance improvement, as measured by the latency reduction, when compared with bulk encryption and WS-Security. However, when the number of attachments increases, and/or when the number of regular expressions increases, the overhead of XPRIDE erodes its latency reduction, as compared to WS-Security.

We note that WS-Security is able to define using WS-SecurityPolicy that a SOAP attachment should be secured. However, it is neither able to apply content-based security on it, nor able to specify a certain type of attachments that need to be secured.

7 Conclusions

We presented in this paper the XPRIDE system, a policy-driven Web Services security solution for securing the privacy and integrity of SOAP messages and attachments exchanged between SOAP processors. XPRIDE depends on security policies to define the parts of the message or attachments that need to be encrypted.

The implementation of XPRIDE was presented in two versions. The first version depends on SOAP filters, while the second is designed as a reverse proxy gateway. In both cases, the results show that XPRIDE provides an important performance improvement as long as the number of regular expressions specified in the policy is limited. For typical Web traffic, XPRIDE is twice as fast as bulk encryption, and 50% more efficient than WS-Security.

Acknowledgments

This work was supported by the Lebanese National Council for Scientific Research, the University Research Board of the American University of Beirut, ACT/Sun Microsystems, and the Rathmann Family Foundation.

References

[1] W3C Recommendation "SOAP Version 1.2 Part 1: Messaging framework," Apr. 2007. (http://www.w3.org/TR/soap12-part1/)

[2] A. Freier, P. Karlton, and P. Kocher, "The SSL protocol Version 3.0," Internet-Draft, 1996.

[3] T. Dierks and C. Allen, "The TLS protocol - Version 1.0," RFC 2246, 1999.

[4] T. Imamura, B. Dillaway, and E. Simon, "XML encryption syntax and processing," W3C Recommendation, 2002. (http://www.w3.org/TR/xmlenc-core/)

[5] M. Bartel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon, "XML-signature syntax and processing," W3C Recommendation, 2002. (http://www.w3.org/TR/xmldsig-core/)

[6] OASIS Standard Specification, "Web services security: SOAP message security 1.1" (WS-Security 2004), Feb. 2006. (http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf)

[7] W. Itani, C. Gaspard, A. Kayssi, and A. Chehab, "PRIDE: Policy-driven Web security for handheld wireless devices," *Proceedings of IEEE GLOBECOM 2006*, pp. 1-6, Nov. 2006.

[8] IBM, Microsoft, "Security in a Web services world: A proposed architecture and roadmap," Apr. 1, 2002. (http://www.ibm.com/developerworks/library/specification/ws-secmap/)

[9] G. Della-Libera et al., "Web services security policy language (WS-security policy)," 2008. (http://specs.xmlsoap.org/ws/2005/07/securitypolicy/ws-securitypolicy.pdf)

[10] J. Daemen and V. Rijmen, "Rijndael, the advanced encryption standard," *Dr. Dobb's Journal*, vol. 26, no. 3, pp. 137-139, 2001.

[11] W. Itani and A. Kayssi, "SPECSA: a scalable, policy-driven, extensible, and customizable security architecture for wireless enterprise applications," *Computer Communications*, vol. 27, no. 18, pp. 1825-1839, 2004.

[12] K. Bhargavan, C. Fournet, and A. Gordon, "Verifying policy-based security for Web services," *Proceedings of 11th ACM Conference on Computer and Communications Security*, pp. 268-277, 2004.

[13] E. B. Fernandez, "Two patterns for Web services security," *Proceedings of International Symposium on Web Services and Applications*, June 2004.

[14] K. Khoo and L. Zhou, "Managing Web services security," *Journal of Information Technology Management*, vol. 14, no. 3-4, 2004.

[15] M. Cremonini, S. Vimercati, E. Damiani, and P. Samarati, "An XML-based approach to combine firewalls and Web services security specifications," *Proceedings of 2003 ACM Workshop on XML Security*, pp. 69-78, Oct. 2003.

[16] R. S. Swart, B. Marshall, M. E. Harris, K. A. Forcht, and D. Olsen, "Security at the edge: rethinking security in light of Web services," *Issues in Information Systems*, vol. 6, no. 2, pp. 103-109, 2005.

[17] H. Nielsen, H. Sanders, R. Butek, and S. Nash, "Direct Internet message encapsulation," Internet Draft, June 2002.

[18] J. H. Gailey, "Sending files, attachments, and SOAP messages via direct Internet message encapsulation," *MSDN Magazine*, Dec. 2002. (http://msdn.microsoft.com/en-us/magazine/cc188797.aspx)

[19] Visolve Squid Team, "Implementing reverse proxy using squid," Feb. 2002. (http://www.visolve.com/squid/whitepapers/reverseproxy.php)

[20] Z. Radwan, C. Gaspard, A. Kayssi and A. Chehab, "XPRIDE: policy-driven Web services security based on XML content," *Proceedings of IEEE GLOBECOM 2007*, pp. 553-558, Nov. 2007.

[21] N. Damianou, N. Dulay, E. Lupu and M. Sloman, "Ponder: a language for specifying security and management policies for distributed systems," *Imperial College Research Report DoC 2000/1*, 2000.

Zein Radwan received her Bachelor's degree in Information Technology Engineering from Damascus University, Syria in 2004, and her Master's degree in Computer and Communications Engineering from the American University of Beirut, Lebanon in 2007.

Camille Gaspard received his Bachelor's degree in Computer Engineering from Aleppo University, Syria in 2004, and his Master's degree in Computer and Communications Engineering from the American University of Beirut, Lebanon in 2006. He is currently pursuing a PhD degree in Computer Science at Purdue University, USA. His research interests include network security, distributed systems security, and applied cryptography.

Ayman Kayssi received his BE with distinction in 1987 from the American University of Beirut, Lebanon and an MSE in 1989 and PhD in 1993 from the University of Michigan, Ann Arbor, USA, all in Electrical Engineering. He is currently professor of Electrical and Computer Engineering at the American University of Beirut, where he has been working since 1993. His research interests are in the areas of information security and trust, and digital system testing.

Ali Chehab received his Bachelor's degree in Electrical Engineering from the American University of Beirut (AUB), Lebanon in 1987, the Master's degree in Electrical Engineering from Syracuse University, and the PhD degree in Electrical and Computer Engineering (ECE) from the University of North Carolina at Charlotte, USA in 2002. From 1989 to 1998, he was a lecturer in the ECE Department at AUB. He rejoined the AUB ECE Department as Assistant Professor in 2002. His research interests include digital system testing and information security.