Vulnerability in cloud computing.
Securing SOAP message using SESoap method

Hadi Razzaghi Kouchaksaraei

Submitted to the
Institute of Graduate Studies and Research
in Partial Fulfilment of the Requirements for the Degree of

Master of Science
in
Computer Engineering

Eastern Mediterranean University
July 2013
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Elvan Yılmaz
Director

I certify that this thesis satisfies the requirements as a thesis for the degree of Master of Science in Computer Engineering.

Assoc. Prof. Dr. Muhammed Salamah
Chair, Department of Computer Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Master of Science in Computer Engineering.

Assoc. Prof. Dr. Alexander G. Chefranov
Supervisor

Examining Committee

1. Assoc. Prof. Dr. Zeki Bayram

2. Assoc. Prof. Dr. Alexander G. Chefranov

3. Asst. Prof. Dr. Gürcü Öz

ABSTRACT

Cloud computing is a concept based on Internet, which delivers large scalable computing resources, as services over the Internet. The main benefit of this technology is the decrease of capital and operational costs, which has caused industrial companies and research communities pay attention to this technology, increasingly. A typical cloud computing systems has special characteristics. According to, five major characteristics have been considered for a typical cloud system. Having those characteristics, there are also three major service models, for each system, which are namely, Cloud Software as a Service (SaaS), Cloud Platform as a Service (PaaS), Cloud Infrastructure as a Service (IaaS).

It should be added here that although there are a lot of positive features counted for cloud systems, there are also some problems that slacken this technology's development. One of the most critical issues, is security, which threatens the successfulness of cloud computing.

It is known that the exchange of information between web applications is done by means of the SOAP protocol. Securing this protocol is obviously a vital issue for any computer network. However, when it comes to cloud computing systems, the sensitivity of this issue rises, as the clients of system, release their data to the cloud.

XML signature is employed to secure SOAP messages. However, there are also some weak points that have been identified, named as XML signature wrapping attacks, which have been categorized into four major groups; Simple Ancestry

Context Attack, Optional element context attacks, Sibling Value Context Attack, Sibling Order Context.

In this study, two existing methods, for referencing the signed part of SOAP Message to counter the mentioned attacks, named as ID and XPath method, are analyzed and examined. In addition, a new method is proposed and also tested, to secure the SOAP message.

In the new method, the XML signature wrapping attack is prevented by employing the concept of XML digital signature on the SOAP message. In this study a different way for signing is used, which is more efficient than the current methods .The results of conducted experiments show that the proposed method is approximately three times faster than the best method, which is currently available.

Keywords: Cloud computing, SOAP message, XML digital signature, Wrapping attack

ÖZ

Bulut bilişim internete dayanmakta olup internette geniş ölçeklenebilir programlama veri kaynağını ve hizmetleri sunan bir konsepttir. Bu sistemin başlıca faydaları sermayenin ve işletim maliyetinin düşürmesidir. Dolaysıyla sanayi kuruluşları ve araştırma topluluklarının bu sisteme gösterdiği ilgi gittikçe artmaktadır. Bulut bilişim sistemlerinin kendine özgü özellikleri vardır. Bir bulut bilişim sistemi genel olarak beş özelliğe sahiptir. Bu özelliklerin yanı sıra, her sistemin üç ana hizmet modeli de mevcuttur; Hizmet Olarak Yazılım (SaaS), Hizmet Olarak Platform (PaaS), Hizmet Olarak Altyapı (IaaS).

Unutulmaması gerekir ki bu sistemin birçok faydaya sahip olmasının yanı sıra, bu teknolojinin geliştirilmesine ilişkin bazı sorunlar da bulunmaktadır. En kritik konulardan birisi bu sistemin başarı oranının olumsuz bir şekilde etkileyebilen güvenliktir.

Bilindiği üzere, web uygulamaları arasında veri değişimi SOAP protokolü (Basit Nesne Erişim Protokolü) aracılığıyla gerçekleşmektedir. Bu protokolün güvenliği her bilişim ağı için hayati önem taşımaktadır. Fakat bulut bilişim sistemlerine gelindiğinde, sistem müşterileri kendi verilerinin buluta sürdükleri için güvenliğin önemi daha da artmaktadır.

XML imzası SOAP mesajlarının güvenliğinin sağlanması için kullanılır. Ancak XML imzasının bazı zayıf yönleri da tespit edilmiştir. Bunlar XML imzası saldırı paketi olarak adlandırılmakta olup dört kategoriye bölünür; Basit Geçmiş İçerik Saldırısı (Simple Ancestry Context Attack), Seçimli Bileşen İçerik Saldırısı

(Optional Element Context Attacks), Benzer Değer İçerik Saldırısı (Sibling Value Context Attack), Benzer Düzen İçerik Saldırısı (Sibling Order Context).

Bu çalışmada, SOAP mesajının imzalanmış kısmına yönelik sözü gecen saldırıları önlenmek için kullanılmakta olan URI ve XPath olmak üzere, iki kullanılmakta olan yöntem analiz edilmiş ve değerlendirilmiştir. Ayrıca SOAP mesajlarının güvenliğinin sağlanması için yeni bir yöntem önerilmeye ve test edilmeye çalışılmıştır.

Yeni yöntemde, XML imza sarma saldırısı SOAP mesajı uzerine XML sayısal imza yöntemlere söre daha fahli ve daha verimli bir ımzalame yüntemi kullanılmıştır. Deneyler göstermiştir ki öneilen yöntem varolan yüntemlere göre üç misli hızlıdır.

Anahtar Kelimeler: Bulut bilişim, SOAP mesajı, XML sayısal imza, Sarma saldırısı

ACKNOWLEDGMENTS

I would like to express my gratitude to my supervisor Assoc. Prof. Dr. Alexander Chefranov, for his precious guidance, valuable supervision and continuous encouragement throughout this thesis work.

My deep appreciations go to my dear family, especially mother, father, my dear brother and sister, whom without their compassionate support; this step could not be achievable for me.

To my dear mom, dad, brother and sister,

who have always supported and loved me

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

SOAP	Simple Object Access Protocol
SESoap	Signing Entire SOAP
XML	Extensible Markup Language
SaaS	Software as a Service
PaaS	Platform as a Service
IaaS	Infrastructure as a Service
NIST	National Institute of Standards and Technology
RPC	Remote Procedure Calls
DCOM	Distributed Component Object Model
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
DTD	Document Type Definition
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XPath	XML Path Language
CAs	Certificate Authorities
DOM	Document Object Model
JPEG	Joint Photographic Experts Group
PNG	Portable Network Graphics
JAX	Java API for XML

Chapter 1

INTRODUCTION

Cloud computing is a new technology [1], which provides greatly ascendable resources such as bandwidth, hardware and software, to be utilized as a service for consumers, over the internet. This concept has attracted wide attention in all kind of industries, recently [2]. One of the most significant advantages of using of this technology is that, consumers can save the cost of hardware deployment, software license and system maintenance. Consequently, the price of providing and using the systems will be reduced, significantly.

There are three main layers in cloud computing [2], depending on the type of provided resources, which are namely; Infrastructure-as-a-service (IaaS) [3], Platform-as-a-service (PaaS) [4] and Software-as-a-service (SaaS) [5]. Very common examples of this system are Google's App Engine, Amazon's EC2 and Microsoft Azure [2].

However, besides being absolutely beneficial, there are still particular unsolved problems [2], in order to implement this concept. It can be said that the most important challenges in cloud computing are security and trust. Since the consumer's data has to be released to the cloud, the system requires high security safety over them. The data in clouds could be very personal and sensitive and must not be unveiled to the unauthorized person. This fact results in the problematic sensitive data leakage or data loss from clouds [2]. In cloud computing, data are threatened during the transition as well. This problem reduces the reliability of the cloud systems [6].

A popular protocol which used to exchange the data in cloud system is Simple Object Access Protocol (SOAP) [7]. This is a communication protocol, which has been used in cloud systems, for transferring data, between applications over the Internet, with the independent platform and language based on Extensible Markup Language (XML) [8]. Securing the data in SOAP messages is one of the main concerns related to security in cloud systems. It can be threatened by XML Signature wrapping attack, which causes the unveiling of sensitive data [9]. This attack is based on altering the structure of the original message from the genuine sender. So as to counter this attack, although diverse remedies have been proposed, none of them have been able to counter the attack completely [9].

The solution provided in this thesis, uses a new method, namely SESoap, to provide integrity for the messages, exchanging in a cloud system by SOAP protocol. In this technique, which is less complicated, more reliable and faster than the current methods, the entire SOAP message is signed by XML digital signature, instead of signing a part of that.

Due to importance of the information existing in SOAP message, finding a method in order to counter the XML signature wrapping attack seems to be a crucial issue. As an example to clarify the importance level of SOAP message information security, it can be assumed that it is containing the information of a bank account’s credit card. Obviously, achieving this information, by a third party, can be terrible for the owner.

Consequently, finding a safe way for transferring data can increase the level of trust and trustworthiness of the cloud system, which is actually the aim of this study. In other words, in the current research, it has been tried to suggest and offer a new way to provide a more secure way to transfer data over HTTP.

Layout of this thesis covers the following outline. In chapter 2, Literature review, some definitions and a summary of pervious works are brought. In chapter3, Experiments with Cloud Computing, some experimental works, in the field of cloud computing and different methods for signing a soap message have been discussed, with their relevant analysis, as well. Proposing and describing the method of SESoap, its analysis, and their results have been given in chapter4, entitled as Proposed SESoap Method to Counter XML Signature Wrapping Attack in SOAP Message. Finally the brief conclusions and achievements of this research have been given in chapter 5, Conclusion.

Chapter 2

LITERATURE REVIEW

One of the most fascinating aims that are being sought these days is to provide computing resources like bandwidth and storage capacity to all the online clients of a system. This aim is tried to be achieved by a new technology called cloud computing. In this chapter, it is mainly tried to explain about cloud computing, its features, and some topics which are mainly related to security issues of these systems.

To have a brief outline, the themes, which are covered in this chapter are firstly, cloud computing, its feature, different service models and deployment models of cloud systems. Secondly, Simple Object Access Protocol or SOAP message, its building block, skeleton and syntax rules are explained. XML signature is the succeeding topics of this chapter, which is followed by clarifications about XML signature wrapping attacks and describing the four major categories of this attack. Finally, in the last section of this chapter, some proposed countermeasures, offered to prevent the mentioned attacks are elucidated.

2.1 Cloud Computing

Cloud computing is a technology which enables the clients to have rapidly supplied, immediate access to a joint of computing resources conveniently, with minimum service provider communication. These computing resources can be networks, servers or even storage applications. This model of network advances the accessibility and is consisted of five important features, three layers, and four deployment models [1].

2.1.1 Essential Features

According to various literatures, between which, NIST (National Institute of Standards and Technology) is one of the most trusted ones, there are some characteristics mentioned for a typical cloud system. They can be categorized in five points.

1) Immediate self-service; meaning that the clients are able to get access to computing capabilities, for example server time and network storage, individually.
2) Wide network access: Features and capabilities of the cloud system should be available over the network (system) and these features also should be accessible for different consumer platforms like laptops and mobile phones.
3) Resource pooling: The provider's computing capabilities are shared, in order to serve several clients, by employing a multi-tenant model, which includes various physical as well as virtual sources, assigned according to clients' demands. Location independence is a characteristic of these systems, which can be explained in a way that the clients actually do not have any knowledge or control on the exact location of the supplied systems' resources, i.e. storage, network bandwidth, processing and etc., although may only be able to identify the country or state.
In other words, the cloud system's capabilities should be accessible for the clients from any location, by using any compatible device.
4) Rapid elasticity: the features and capabilities of cloud systems should be delivered rapidly and elastically, meaning that they are ought to be available unlimitedly, and

in certain cases, be able to scale outward and inward rapidly, depending on the demand.

5) Measured Service: The cloud system’s resources can be controlled and monitored and also reported, to both provider and consumer of the service, as a matter of transparency and also resource optimization. This feature can be fulfilled by employing an assessing capability, suitable to the type of service [1], [10].

2.1.2 Service Models

According to the literatures, there are three service models, defined for a typical cloud computing system, which are aimed to be explained in the following sections.

1) Cloud Software as a Service (SaaS): the cloud systems’ capabilities are available to the consumers, in a way that it lets them to utilize the applications or features, which are running on the cloud infrastructure. These applications are available to the consumers via a thin client interface (like a web browser). The clients do not have the access and also are not able to manage or control the substructure of the cloud system, like network or servers. However, there can be some exceptions in the form of some limited settings, specifically defined for users. A good example of this model is MicroSoft Office365, which is services the clients (software services, automatic backup, update and etc.), without any requirement to installation.

2) Cloud Platform as a Service (PaaS): in the cloud system, the provided capability to the clients is to use and arrange in the cloud infrastructure’s developed applications (either by clients or provider), which are created by employing programming languages and tools that are in fact supported by provider. This means that the clients do not control or manage the substructure of the cloud system, including network and

operating system. However, controlling capability has been provided over the used applications and perhaps over the configuration settings, for the application-hosting environments. This service is mostly used by software development companies, in order to execute their products.

3) Cloud Infrastructure as a Service (IaaS): in this model, storage, networks and some other essential computing resources are provided to the consumers. The consumers are capable of running random software, possibly even operating systems and other applications. In this model also, like the previous ones, clients do not have the permit to control or manage the cloud system's substructure, however, controlling permit of operating systems, storage and utilized applications and possibly some selected networking components (like host firewalls), have been provided. This model of service covers a wide range of services including disk drives, storage devices, and individual servers [1], [11].

2.1.3 Utilization Models

According to most of the publications, utilization of cloud computing systems is mainly categorized into four main models, which will be explained comprehensively, in the following paragraphs.

c) Public cloud: in this model, the cloud system is available to the public or a big group organization, and is owned by a specific organization, which provides and sells the service. Some examples of public cloud system are SunCloud, IBM Blue Cloud, and Google App Engine [1] & [12].

b) Community cloud: in community clouds, the cloud system’s infrastructure is actually shared between various users and organizations and is aimed to support the shared issues of definite association. Issues like specific missions, security, policy and etc. it can be managed and owned by the organizations or a third party and can be on premise or off premise [1].

d) Hybrid cloud: The hybrid system services in composed of several cloud systems that can be private or public. Although the systems are bound together to let data and applications transfer easily, each of these components are remained exclusive units. As an extra explanation, it can be said that a hybrid cloud can be a cloud system, in which a part of resources have been provided locally (internally) and the rest have been provided externally (public) [1], [13].

a) Private cloud: in these cloud systems, the cloud infrastructure is functioning only for a specific organization, which may include several consumers as well. The system can be managed and operated by the relevant institute or an intermediary. Amazon Virtual private cloud can be named as a private cloud system [1], [14]

2.2 SOAP

SOAP, which is the abbreviation of Simple Object Access Protocol [15], is a communication protocol, in order to communicate between various applications. Having the platform and language, independent and XML-based respectively, it works as a format for sending messages via internet, which also collaborates with the firewalls [16], [17], [15].

Internet connection between programs is indeed an important requirement that is fulfilled these days by means of Remote Procedure Calls (RPC) (to connect objects such as DCOM and COBRA). However, there is another problem caused by non-compatibility of RPC with HTTP, which is not planned for this matter. Therefore, RPC is obviously blocked by firewalls. To overcome this problem, SOAP was created, which can communicate between applications over HTTP. It offers a path to communicate between applications, on different operating systems, with diverse programming languages [17].

2.2.1 SOAP Building Blocks

As it is also mentioned above, SOAP message's language is based on XML [16]. Moreover, it can be explained that the building block of SOAP is in fact a typical XML document, which is consisted of these items:

1) Envelope: this element recognizes the XML document as a SOAP message.

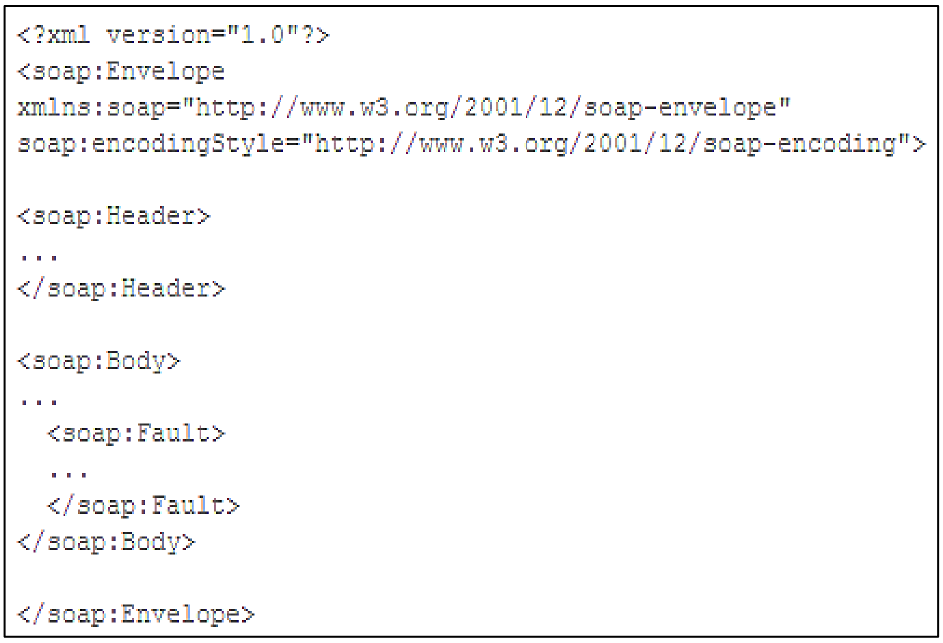
2) Header: this element includes the header information

3) Body: this element includes the actual SOAP message

4) Fault: Errors that occurred while processing message are included in this element [16], [18].

2.2.2 Skeleton of SOAP Message

A typical skeleton of SOAP message have been shown in Figure 2.1.



```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
...
</soap:Header>

<soap:Body>
...
  <soap:Fault>
  ...
  </soap:Fault>
</soap:Body>

</soap:Envelope>
```

Figure 2.1: Skeleton of A SOAP Message [18]

2.2.3 Syntax Rules

Essential syntax rules of SOAP messages can be summarized in the following guidelines. Encoding must be done by XML. SOAP encoding namespace and SOAP envelope namespace must be used in a typical SOAP message [18]. On the other hand, DTD reference and XML processing instructions should not be contained within in a SOAP message [15].

2.2.4 SOAP Message and Web Services

In order to clarify the relation between simple object access protocol (SOAP) and web service, each of them have been defined briefly separately in bellow paragraphs.

A Web service is defined as a designed software system to support interoperable machine-to-machine communication over a network. Its interface is described in a machine-processable format (specifically WSDL) [19].

Web Services Description Language (WSDL) is an XML-based language in which an interface of a Web service is described [20].

SOAP message, which have also been defined specifically in the previous section is in fact the communication protocol, for exchanging information between diverse applications.

According to World Wide Web Consortium, the relation between Web service and SOAP message can be explained as the following paragraph.

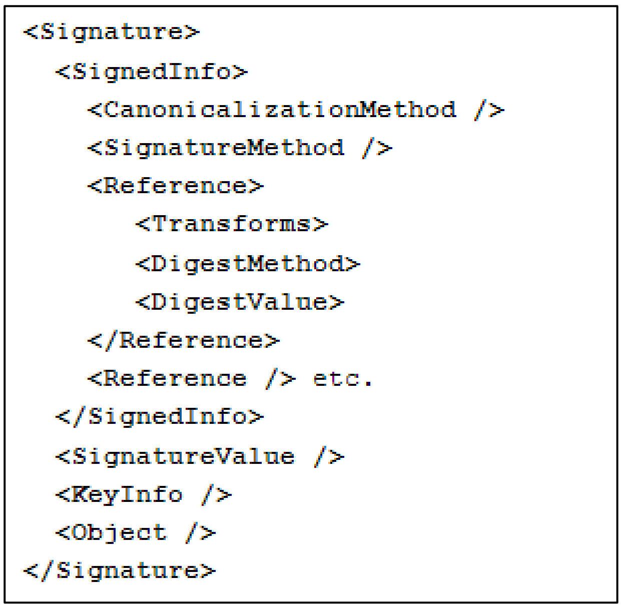
> “A Web service is a software system to support interoperable machine-to-machine interaction between computational resources over a network using Simple Object Access Protocol (SOAP) messages.” [20]

2.3 XML Signature

Xml signature is a technique, which is used to deliver reliability, integrity and message authentication, for various types of data [21]. By providing integrity to data, it is meant that once the data is signed; it cannot be altered later, without invalidating the signature. This technique is executed by employing asymmetric cryptography. It works in two ways: it can be used by the sender of a document, as evidence that the message is authored by the sender or also can be used by the receiver, indicating that the receiver has authored it. In addition, the second way, can deliver non-repudiation, in a way that, when a message is signed by a receiver, the authorization of the document cannot be denied later on by the receiver. The roles for signing a document are as follows [22].

$$M = D_{P_C}[E_{R_C}[M]] = D_{R_C}[E_{P_C}[M]]$$

Asymmetric encryption, uses two keys in order to encrypt and decrypt the function, which are named private (Rc) and public (Pc) keys. XML digital signature employs private key and public key to sign a message and validate the document, respectively. When signing the message, signature will be attached to the original document, and will be sent to the receiver. It should be noted that the document, is not hidden, since hiding the message is not the aim of XML digital signature. Since asymmetric encryption is time consuming, a hash function (f(M)) is calculated over the document and the result, which is called digest value, is considerably smaller than the document itself. The result of hash function is then encrypted by private key. Consequently, the time passed for encrypting data will be reduced, significantly. Figure 2.2 shows the structure of an XML signature.



```
<Signature>
  <SignedInfo>
    <CanonicalizationMethod />
    <SignatureMethod />
    <Reference>
       <Transforms>
       <DigestMethod>
       <DigestValue>
    </Reference>
    <Reference /> etc.
  </SignedInfo>
  <SignatureValue />
  <KeyInfo />
  <Object />
</Signature>
```

Figure 2.2: Structure of an XML Signature [22]

The signing process is done according to the following stages: for each xml element, which is ought to be signed, there is one reference element. The specified element is first canonicalized, and then hashed. The information of canonicolaization method goes to canonicolaization method element, and the result of hash function goes to digest value element. Also, signature value has kept the result of asymmetric cryptography of signed info element of the signature. To validate a signature, the receiver must apply two operations. First, apply the hash function, used by sender, over the document and compares it with the digest value of the signature, then, decrypt the encrypted data and compare it with the signed info element of the signature. If the results of two comparisons are true, the signature will be validated [22].

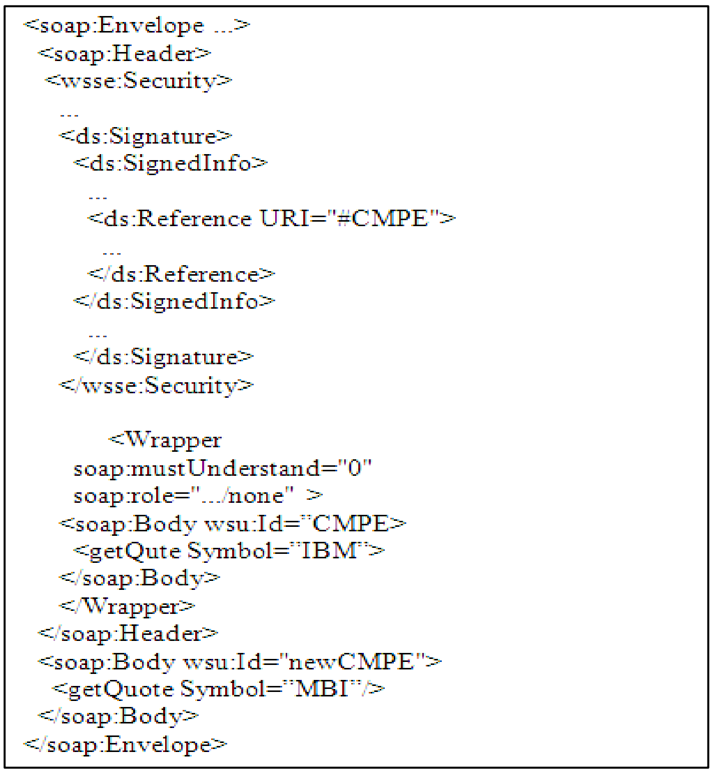
2.4 XML Signature Wrapping Attack

XML signature wrapping attack, can threaten the document because of the fact that the signature dose not convey any information to where the referenced element is placed [23]. This attack was introduced for the first time, in 2005 by McIntosh and Austel, stating different kind of this attack, including Simple Context, Optional Element, Optional Element in security header (sibling value) and Namespace injection (Sibling order) [24]. This attack happens in SOAP message, which transfers the XML document, over the internet.

2.4.1 Simple Ancestry Context Attack

In Simple Ancestry Context Attack, a request's SOAP body is signed by a signature, which is placed in the security header of the request. The recipient of message, checks if the signature is correct and legalizes trust in the signing credential. Lastly, the recipient controls to realize whether the required element was actually signed, by bringing the "id" of the soap body to the URI reference, in the signature [25].

A typical example of this attack is shown in Figure 2.3.This mechanism of this attack can be briefly explained in this way that, the SOAP body gets swapped with a malicious SOAP body. The original SOAP body is placed in a <wrapper> element, which is situated in the SOAP header and when the signature is validated, The XML signature confirmation algorithm, begins searching for the element, which has the id of "CMPE", as it is stated in the <Reference> element. Finally, <soap:Header> Element wrapped within the <wrapper> element, will be found by the algorithm. Signature verification will be implemented on the <soap:Header>, within the <wrapper> element. The verification will be positive, because it includes the original SOAP body, which is signed by the sender. The SOAP message will be passed to the logic of the application. In the application logic procedure, only the SOAP body, which is straightly positioned under the SOAP header, will be processed. In other words, all other SOAP body elements will be just ignored [25]. Figure 2.3 shows how this attack works.

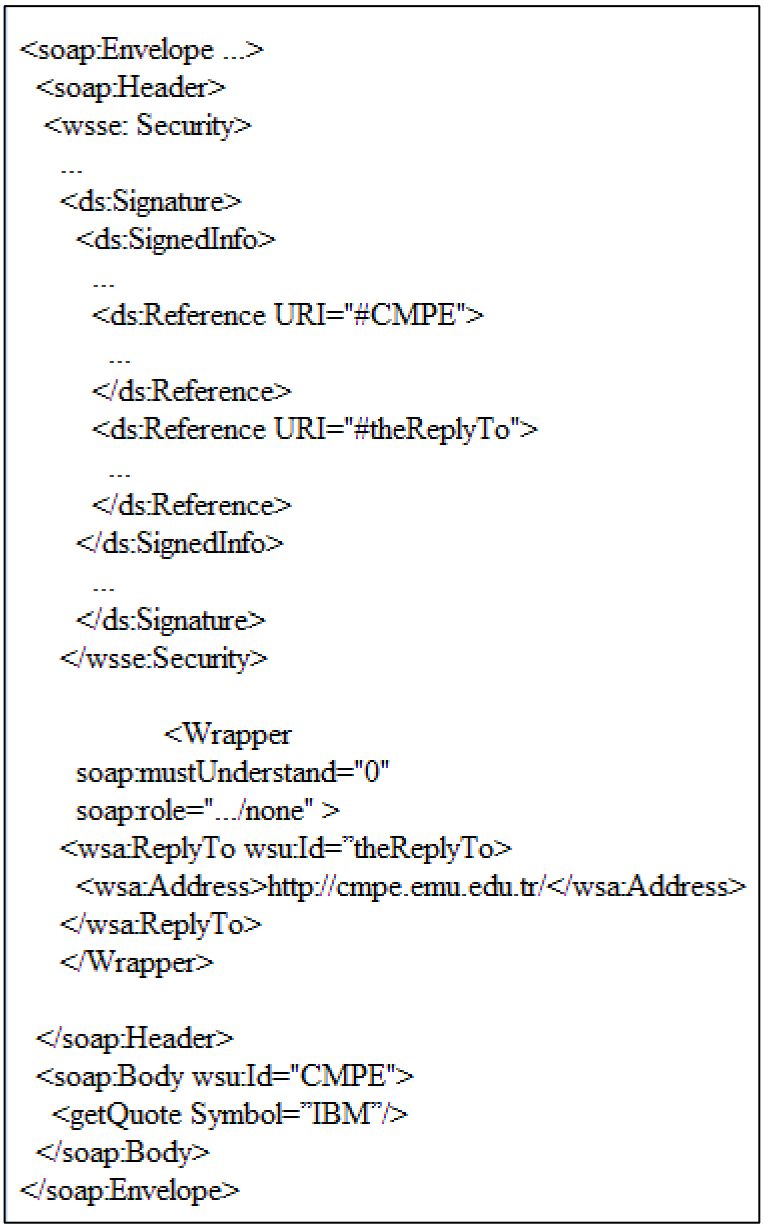


```
<soap:Envelope ...>
  <soap:Header>
   <wsse:Security>
    ...
    <ds:Signature>
      <ds:SignedInfo>
        ...
        <ds:Reference URI="#CMPE">
          ...
        </ds:Reference>
      </ds:SignedInfo>
        ...
      </ds:Signature>
    </wsse:Security>

         <Wrapper
      soap:mustUnderstand="0"
      soap:role=".../none"  >
    <soap:Body wsu:Id="CMPE>
      <getQute Symbol="IBM">
    </soap:Body>
    </Wrapper>
  </soap:Header>
  <soap:Body wsu:Id="newCMPE">
    <getQuote Symbol="MBI"/>
  </soap:Body>
</soap:Envelope>
```

Figure 2.3: Typical Simple Ancestry Context Attack [24]

2.4.2 Optional Element Context Attacks

In optional element context attacks, the signed data is contained in the SOAP header and it is arbitrary. Comparing this attack to the simple context attack, which is explained above, reveals that the main problem is not the place of the signed data in the SOAP header [26]. In fact, the optional nature of signed data is the main issue [24]. The <ReplyTo> element, which specifies where to send the reply to, can be given as an example, which is shown in Figure 2.4. The mechanism of this attack can be explained as follows; it can be seen that the element of <wsa:ReplyTo> is placed in the <wrapper> element, while, the element of <wrapper> is also positioned underneath the <wsse:security>. In addition, by means of soap:mustUnderstand="0", in <wrapper>, this element has become optional and by using soap:role=".../none", it is destined that the SOAP node (application logic) should not process this header element. These modifications in the SOAP message, result in the <wsa:ReplyTo> to become completely disregarded by the application’s logic. Having these attributions, when the signature gets legalized, the verification algorithm of XML signature begins to search for the element, which has the id of "theReplyTo" (specified in the <Reference>) and <wsa:ReplyTo>, which is in the <wrapper> element, will be found. At this stage, signature confirmation will be done on the <wsa:ReplyTo>, in the <wrapper>, and because it is including the original <wsa:ReplyTo>, signature confirmation will be positive. Consequently, SOAP message body and the descendants, which are understood, will be handed to the application logic while the <wrapper>, will not be passed to it. Thus, the application logic will ignore the <wsa:ReplyTo> element and as the result, the original message sender will get the reply [26].



```
<soap:Envelope ...>
  <soap:Header>
   <wsse: Security>
     ...
     <ds:Signature>
       <ds:SignedInfo>
         ...
         <ds:Reference URI="#CMPE">
           ...
         </ds:Reference>
         <ds:Reference URI="#theReplyTo">
           ...
         </ds:Reference>
       </ds:SignedInfo>
         ...
       </ds:Signature>
     </wsse:Security>

                   <Wrapper
       soap:mustUnderstand="0"
       soap:role=".../none" >
     <wsa:ReplyTo wsu:Id="theReplyTo>
       <wsa:Address>http://cmpe.emu.edu.tr/</wsa:Address>
     </wsa:ReplyTo>
     </Wrapper>

  </soap:Header>
  <soap:Body wsu:Id="CMPE">
    <getQuote Symbol="IBM"/>
  </soap:Body>
</soap:Envelope>
```

Figure 2.4: Typical Optional Element Context Attacks [24]

2.4.3 Sibling Value Context Attack

Sibling value context attack covers the following scenario. In this attack, the security header includes a signed element, which is in fact an alternative sibling of <Signature>. A common model for this attack can be the element of <Timestamp>, which together with <Signature>, are direct descendants of SOAP security header.

The difference between this attack and the previously discussed attacks is in the signed data, which in this attack is the sibling of <Signature> [27].

According to what is also shown in the Figure 2.5, it can be seen that the element of <wsu:Timestamp> that states the time period, during which the SOAP message is considered to be valid, is the signed part. < wsu:Timestamp> is assumed to be an optional element and it should be perceived that the signed data is referred through an XPath expression. All the <wsu:Timestamp> elements, which are descendants of <wsu:Security> will be selected by the XPath. The element of <wsu:Security> is also a descendant of soap:Header, and the soap:Header is also a descendant of all the elements that are called soap:Envelope [24], [27].

As it also can be seen in the Figure 2.5, this attack is performed by removing the element of <wsu:Timestamp>, from its authentic original place and has been placed in the second added element of <wsse:Security>. Moreover, two new characteristics have also been added to the <wsse:Security> element. The first one is soap:mustUnderstand="0", and the second one is soap:role=".../none". By means of these attributes, the header element will be ignored and that is the application's logic should not process it [24].

And when the signature gets validate, during the process of SOAP message, in the verification process of XML signature, the algorithm, starts to look for <wsu:Timestamp> as it is stated in Xpath. By finding this element within the second element of <wsse:Header>, the signature confirmation will be done on the <wsu:Timestamp>, which is placed in the second element of <wsse:Security>. Consequently, the signature confirmation will be positive, as the XPath finds the original timestamp. Apart from the second element of <wsse:Security>, all the SOAP message body and its descendants will be passed to the application logic. <wsse:Security> will not be passed, because it is containing information, which result in its negligence. In other words, the optional element of <wsu:Timestamp> is totally ignored and an attacker will be successful by carrying out a reply attack [27].

```
<soap:Envelope ...>
 <soap:Header>
  <wsse:Security>
    ...
    <ds:Signature>
      <ds:SignedInfo>
         ...
         <ds:Reference URI="#theBody">
           ...
         </ds:Reference>
         <ds:Reference URI="">
           <ds:Transforms>
             <ds:Transform Algorithm=".../REC-xpath-19991116">
               <ds:XPath ...>
                  /soap:Envelope/soap:Header/wsse:Security/wsu:Timestamp
               </ds:XPath>
              </ds:Transform>
              ...
             </ds:Transforms>
             ...
        </ds:Reference>
      </ds:SignedInfo>
       ...
    </ds:Signature>
  </wsse:Security>
  <!-- 2nd SOAP Security Header added by attacker-->
  <wsse:Security
    soap:mustUnderstand="0"
    soap:role=".../none">
    <wsu:Timestamp wsu:Id="theTimestamp">
       <wsu:Created>2005-05-29T08:45:00Z</wsu:Created>
       <wsu:Expires>2005-05-29T09:00:00Z</wsu:Expires>
    </wsu:Timestamp>
  </wsse:Security>
 </soap:Header>
 <soap:Body wsu:Id="theBody">
  <getQuote Symbol="IBM"/>
 </soap:Body>
</soap:Envelope>
```

Figure 2.5: Typical Sibling Value Context Attack [24]

2.4.4 Sibling Order Context

According to McIntoch and Austel, 2005, this attack is dealing with the protection of the sibling elements that are individually signed.

> Their semantics are related to their order relative to one another, from reordering by an adversary. More work is required to define appropriate countermeasures that do not prevent the addition and removal of siblings that do not impact the ordering semantics [24].

2.5 Proposed Countermeasures

The requirements of a service-side security policy, in order to detect an attack were shown by McIntosh and Austell, 2005 [24]. These necessities are being improved by each attack, which is able to bypass the previous provided security policy. In continuance, some of the improvements in the policy will be explained.

1) In the wsse:security header element, a signature "A" , XML signature, should be placed, having a clear soap:role attribute and value of "…/ultimateReceiver".

2) From signature "A", The element, which are identified by /soap:Envelope/soap:Body, must be referenced.

3) In the case of having any elements, which are matching with /soap:envelop/soap:Header/wsse:Security[@role="…/ultimateReceiver"] wsu:Timestamp and /soap:Envelop/soap:Header/wsa:ReplyTo, it should be noted that these elements must be referred through an absolute path, Xpath expression, from signature "A".

4) Verification key of signature "A" must be issued and provided by a trusted Certificate Authorities (CAs) and the certificate of X.509v3, respectively [24].

The first example of XML signature wrapping attack, which was indicating that the controls suggested by McIntosh and Austell are not satisfactory to notice XML signature wrapping attack, was shown by Gruschka and Lo Iacono, in 2009 [28]. It is also claimed in their research that the timestamp has to be referenced by an extra XPath expression, which is not fulfilled in Figure 2.5. Although, it can be added easily, it should be noted that the XPath references result in further problems. It is known that XPath expressions are more difficult to be evaluated, comparing to IDs, this issue is especially important in the context of streaming SOAP message. Another more important issue is that employment of XPath references may indicate security issues, so they are not suggested by basic security profile [29].

In a new method, which was proposed in 2006 and is named as inline method, a new element called SOAP account was introduced. Some characteristic information are gathered together and inserted in the SOAP account element [28]. Protection of some key features of SOAP message structure is aimed in this technique. The properties, which are aimed to be protected, are listed as below.

1) Number of header element descendants
2) Number of soap:envelop, descendent elements
3) Amount of references in every signature
4) The descendants and antecedents of every signed item

By means of this approach, with the above properties, if in an attack, each of these properties is changed, the attack will be easily identified [30]

However, it is important to add that this method has some weak points and disadvantages as well. As an example, two of the main disadvantages will be explained in continuance.

The main problem with this method is that it does not provide a general protection, from XML signature wrapping attack. In other words, if an attacker manages to change the SOAP message structure in a way that the inline method structure properties does not get changed, this technique can be easily dodged [31].

Another model for policy confirmation was developed in 2005, by Bhargavan, Fournet and Gordon. A policy advisor checklist, for generating security policies has been derived by them, which includes compulsory and signed elements, and a recommendation. Mandatory elements are wsa:To, wsa:Action, soap:Body, signed elements are all the mandatory elements plus wsa:messageID and Timestamp. It is also suggested that the certificate of X.509, to be employed for authentication [32]. The second issue about inline method is that the newly introduced element of SOAP account, and its verification, is not standardized [33].

Additionally, Gajek et al. proposed some solution ideas for protecting against the signature wrapping attacks. Having the core idea of using the verification component as a filter, this method returns the result of canonicalization and transformation step (while in common methods, Boolean value are returned). This difference certifies that the subsequent components, which are inside the web service framework, function accurately on the originally signed message [31]. There are also some problems about this method, which were already mentioned by the other authors. One weak point is about the operation of web service on SOAP envelope. The SOAP envelope cannot be operated as a single well-formed message document. To solve this issue, the signed elements and their parent nodes can be passed together to the business logic part of web service, as a spanning DOM. However, to have the problem solved with this method, the content of elements should not be changed by the signature transformations. However, this solution can also be inadequate if both signed and unsigned parts of SOAP message are operated by the web service, which will not allow the signature components to operate as filter [28].

In addition, fastXPath method was proposed by Gajek et al., in 2009. This method is employed to increase the speed of XPath function, and to point to the signed subtree. However, this method also could not solve the identified issues about XPath expression [34]. A comparison between runtime of different methods, ID, fastXPath and XPath methods, have been also done in this article. The comparison's relevant graph is shown in Figure 2.6.

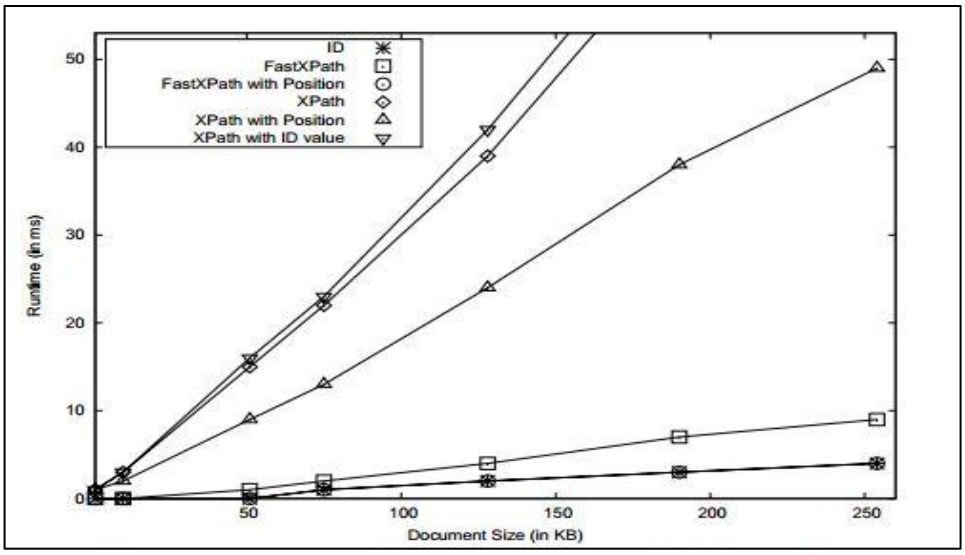


Figure 2.6: Runtime Comparison of Different Referencing Methods [28]

2.6 Problem Definition

It can be easily found out from the topics, which were discussed and explained in this chapter (especially the countermeasure section), that although there are rapid improvements about the concept of cloud computing and especially about security issues in these systems, there are still missing methods, that should be provided to have better and well developed cloud systems. Specifically, as it has been also explained about the proposed methods to prevent XML signature wrapping attacks, the most advanced method is only avoiding two attacks out of total four attacks. In other words, there is not any method to prevent Sibling Order Context and Sibling Value Context attack.

During this investigation, going through the previous literatures has been done in the literature review chapter. In the 3rd chapter, experiments with Cloud Computing, an example of available cloud computing systems was chosen to work and get familiar with these systems. Finally, it has been attempted to propose a new method in order to avoid all the mentioned attacks.

Chapter 4, Proposed SESoap Method to Counter XML Signature Wrapping Attack in SOAP Message, includes the proposed method is also implemented and verified, to check its efficiency. Implementation of the purposed method was done by using C#.net programming language in Visual Studio 2010 framework. Finally, conclusion remarks of this study have been also collected briefly in chapter 5, Conclusions.

Chapter 3

EXPERIMENTS WITH CLOUD COMPUTING

As the experimental section of this research, it was attempted to work with available examples of cloud systems, to find out more about the positive features, negative points and generally their features. A famous example of cloud computing system is Google App Engine, which lets the users perform applications on the infrastructure of Google [35]. In this investigation, it has been tried to work with this system, in order to get familiar with the concept of cloud computing systems and their features. Google App Engine has been used to write some programs, in Python environment, which were later shared over the Web, again by means of this engine, to investigate how a cloud computing system works.

Outline of this chapter covers the following headings. Firstly, Google App Engine have been chosen as a typical cloud computing system, a few of its features and capabilities, including the environment, sandbox, services and the Python runtime environment (as example), have been explained. In the next step, available methods of referring signed part of SOAP message, for securing SOAP message, have been explained briefly and finally, the mentioned methods have been tested experimentally, by means of Visual Studio 2010 framework and C#.net programming language.

3.1 Google App Engine

As it is also mentioned above, Google App Engine is provides its users with the amenity of building web applications, based on an ascendable system, on Google infrastructure. Web application, can be defined as an application or service, which is accessible over the web [36]. Usually with a web browser there is no preserving service for this system and the size of it grows, as there is a requirement to the growth of traffic and data storage.

This system has the ability to support applications, which are written by mean of various programming languages, such as Java, Go, and Python. In addition, in the free version of this system, there are limitations for the storage capacity CPU and bandwidth [37]. In this version, all the applications can use the storage of maximum 1GB, bandwidth and CPU for an application which serves around 5 million page views per one month. The applications can be served from the programmer's domain name, or also, it is possible to publish the applications by creating free accounts [35].

3.1.1 The Application Environment

The following points are some of the system's important advantages:

- Dynamic web serving
- Persistent storage with queries, sorting and transactions
- automatic scaling and load balancing
- APIs for authenticating users and sending email using Google Accounts
- a fully featured local development environment that simulates Google App Engine on your computer
- task queues for performing work outside of the scope of a web request
- scheduled tasks for triggering events at specified times and regular intervals [35]

Three runtime environments, with standard protocols and shared technologies, have been provided for the applications, Go, Python and Java environments.

3.1.2 The Sandbox

Sandboxing also lets App Engine run several applications on the same server, separately, without being affected by each other [36]. A secure environment is provided for the applications, which allows the applications to have limited access to the original operating system. By means of these limitations, App Engine will be able to allocate and manage the web demands of the applications, across the various servers. Also it will be able to start and stop servers, to satisfy the traffic requirements.

Environment of the sandbox is not dependent of hardware, operating system and of course the location of web server [35], [36]. Some of the limitations of the secure sandbox are explained in the below paragraphs.

1. There is a limited time interval of 60 seconds, for an application to execute and bring the response data to a scheduled task or web request. After when a response is sent, the request handlers are not able to run code or start a sub-process.
2. The accessibility of applications to the computers over the internet is provided via the afforded email services and URL fetch. On the other hand, the access of other computers to the applications have been made by making HTTP (or HTTPS) requests, on the standard ports.
3. The applications can only read those files, which are uploaded including the application code. Meanwhile, for any data that persists between requests, the applications must use the provided services in the App Engine, such as memcache. It

should be added that in the environment of Python 2.7, the facility of reading, writing and modifying bytecode have been provided [35], [36].

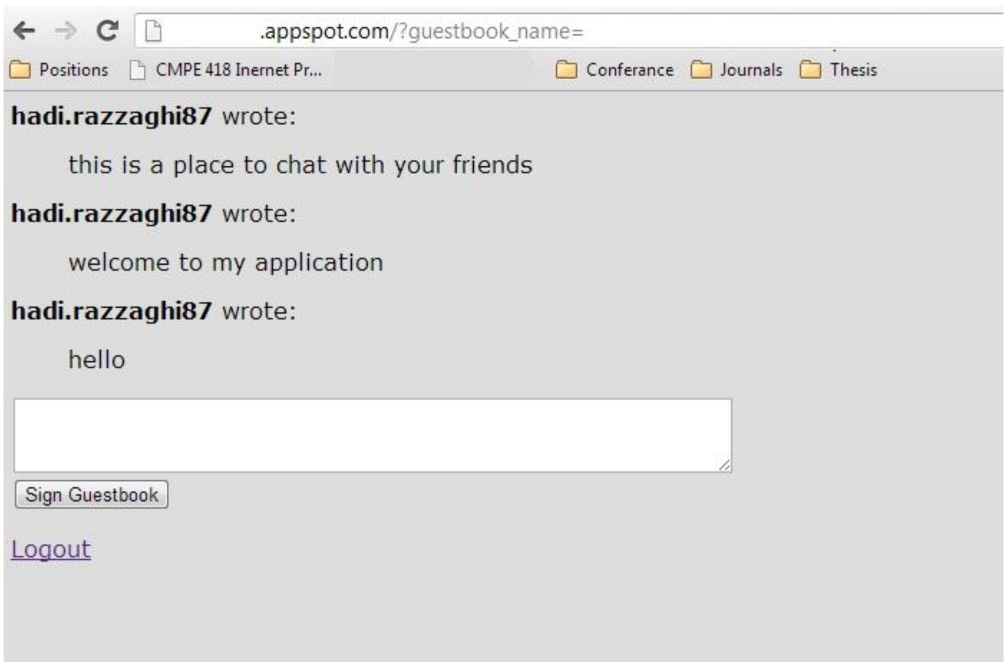
3.1.3 The Python Runtime Environment

In the Python Runtime Environment of Google App Engine, applications are implemented by using Python programming language. The application written in Python 2.5 (the language), are run, employing the official Python interpreter [35], [36]. The facility to utilize and take advantages from various frameworks and libraries of Python web application, such as web2py, has been potentially provided [35].

Python standard Library has been made available through the Python environment, although there are exceptions. These exceptions are such as trying to write a file or open a socket. The elements that are not supported by the runtime environment have been disabled and their corresponding codes, to import them, are set to raise error [36]. In addition, webapp2, which is a simple web application framework, have been also provided in App Engine that provides a more convenient way for application building [35].

3.1.3.1 A Practical Example of Python Environment

As an attempt to work with Google App Engine, an application has been written, using Paython. A screen shot of it have been shown in Figure 3.1. The clients can chat by using this application, in two ways. They both can chat as a guest user, having “guest” identity and also they can enter their email identity and chat by their own username.



hadi.razzaghi87 wrote:

this is a place to chat with your friends

hadi.razzaghi87 wrote:

welcome to my application

hadi.razzaghi87 wrote:

hello

Figure 3.1: Practical example of Python environment

3.1.4 App Engine Services

Several services have been provided in Google App Engine, in order to enable the users, to execute common processes, during application handling [35]. The following items are some of the provided application programming interfaces (APIs).

3.1.4.1 URL Fetch

Google App Engine's URL fetch have been provided as a facility to enable applications get access to the internet resources. This service works by using Google infrastructure, which is the same infrastructure of other Google products [35].

3.1.4.2 Memcache

The Memory cache or shortly, the Memcache service [36], is a storage service, which besides having the advantage of being short-term and a key-value memory, also, it has a remarkable superior property over the datastore that is being much faster than it, which is indeed a positive point especially for simple data retrieval and storing [36]. It is absolutely useful for temporary data or data copied from the datastore, as they don’t need features of transaction or persistence [35].

3.1.4.3 Image Manipulation

This API lets the users work with or correct image files. Images in the formats of JPEG and PNG can easily be resized, rotated, cropped or flipped by means of this service [36].

According to Rudominer, 2011, it is possible to build a SOAP server and a SOAP client, by means of java.xml.soap, JAX-B and JAX-WS, respectively [38]. It should be added here that the security issue of SOAP message, which have been introduced by McIntosh and Austel in 2005 [39], was also noticed in Google App Engine, during the literature review and experimental analyses of the current research work.

3.2 Available Methods To Refer Signed Part of SOAP Messages

SOAP message is defined as a protocol that is utilized for conveying information over HTTP, between the web applications [7]. The skeleton of a SOAP message is shown in Figure 3.2.

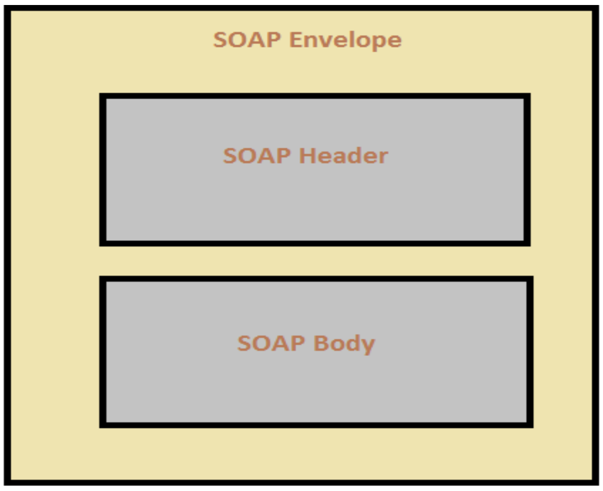


Figure 3.2: Skeleton of a Soap Message

The major element of SOAP message is the Envelope, which includes two sub elements, i.e. SOAP Header and SOAP Body. The element of SOAP Header is actually an optional element that includes specific information of applications, such as authentication. On the other hand, SOAP Body, is known as a compulsory element, incorporating the real SOAP message, which is aimed to reach to the ultimate endpoint [40].

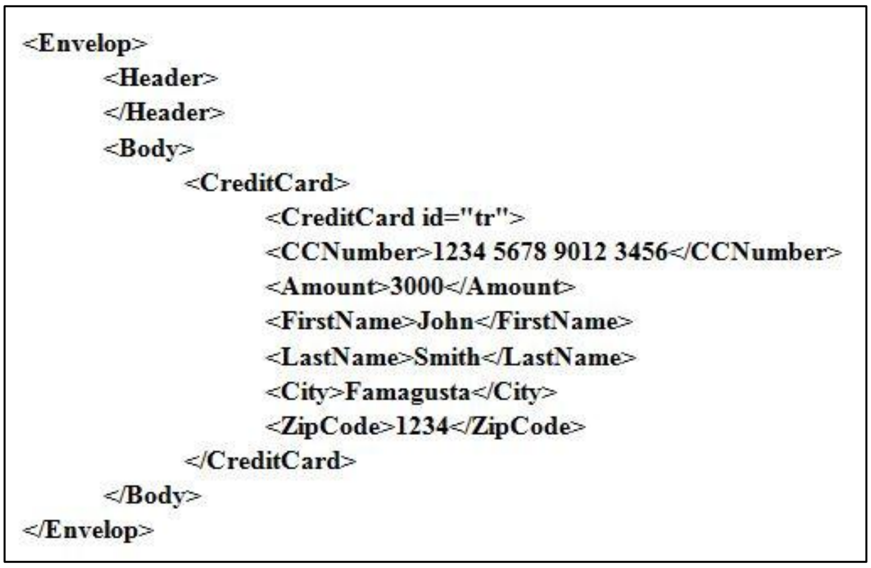
In order to secure the information inside the SOAP body, XML digital signature is utilized [41], [42]. However, vulnerable signed documents might occur, caused by hidden alterations, done by a hacker (adversary). These vulnerabilities are caused by a so-called wrapping attack [39]

As it is also explained comprehensively in the previous chapter, there are four major wrapping attacks, which are called as Simple Context, Optional element, Optional Sibling Value and Sibling Order Context [39].

Aiming to refer to the SOAP Body of a SOAP document, ID reference is being used by XML digital signature [43].Despite the fact that it is the fastest method [44]; it is also susceptible to XML signature wrapping attack [39]. Another method for detecting the signing element of SOAP message is XML Path Language (XPath) [45], which is also employed to counter optional element and simple context attack [39]. Unfortunately, this method is time consuming [44] and is remained vulnerable against Optional Sibling Value and Sibling Order Context attacks [9].

3.3 Examination of ID and Xpath Methods

In order to examine the mentioned attacks, C#.net programing language, was employed. The figure below (Figure 3.3) shows the SOAP message context of this study.



```
<Envelop>
        <Header>
        </Header>
        <Body>
                <CreditCard>
                        <CreditCard id="tr">
                        <CCNumber>1234 5678 9012 3456</CCNumber>
                        <Amount>3000</Amount>
                        <FirstName>John</FirstName>
                        <LastName>Smith</LastName>
                        <City>Famagusta</City>
                        <ZipCode>1234</ZipCode>
                </CreditCard>
        </Body>
</Envelop>
```

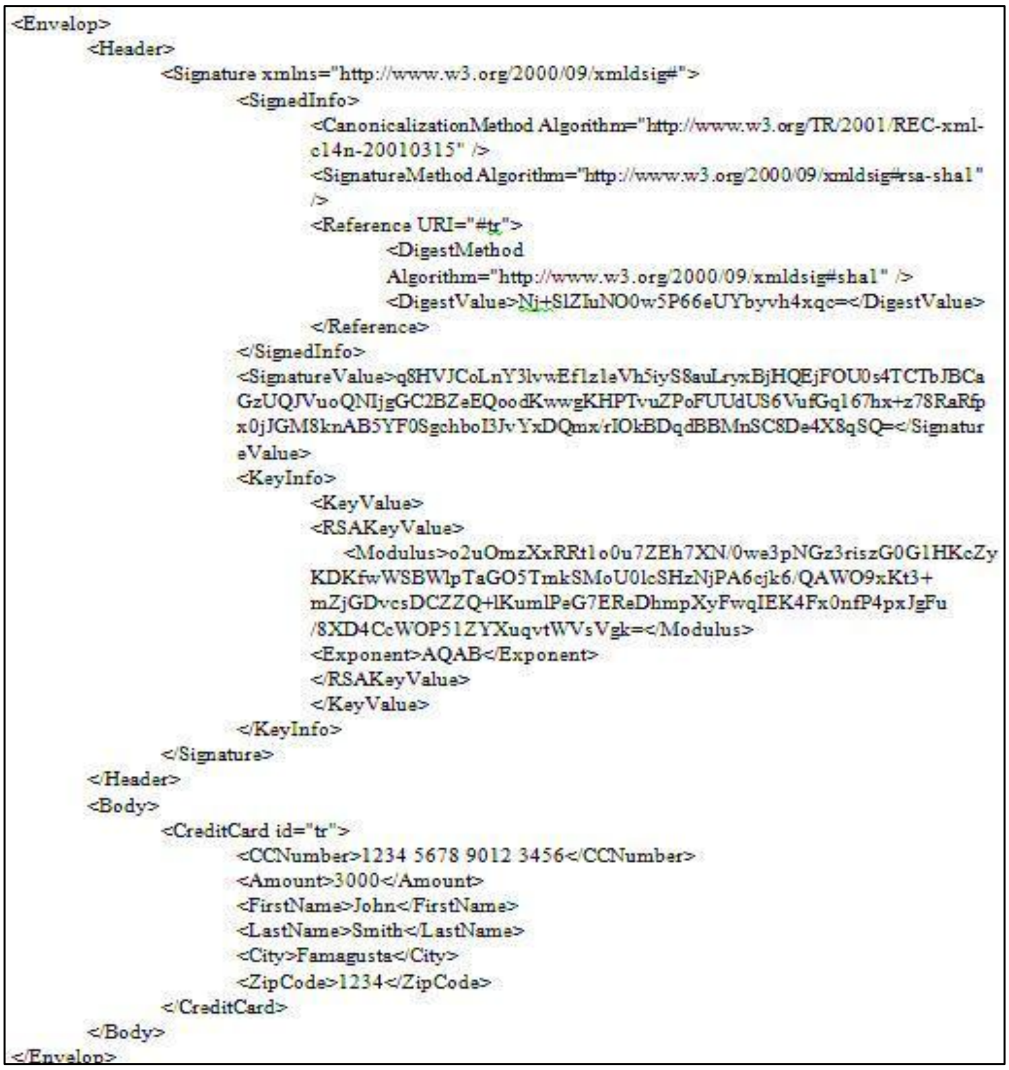
Figure 3.3: SOAP Message Context

After signing the Soap message, by using ID (shown in Figure 3.4), for referring the SOAP Body element, the result indicated susceptibility to XML wrapping attack (See also Appendix A). However, in this method, signing and verification were implemented very rapidly. The consumed time for finding the specified element by using ID method is shown in Table 3.1.

Table 3.1: Time Durations to Detect the Signing Element Using ID Method

Size (Kb)	Time consumed by ID (ms)
50	0
100	0
150	1.0003
250	1.0006
750	3.002
1050	4.0003
1350	5.0003
1650	6.0004
1950	7.0003
2250	8.0004
3150	11.0034

In the evaluation, the location of the reference element was always kept exactly at the middle of the document. The process of evaluation and running the program was done by using Laptop, with 2.00 GHz Core2Duo CPU and 1.00 GB memory. The results are given out in the Figure 3.4.



```
<Envelop>
        <Header>
                <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
                        <SignedInfo>
                                <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-
                                c14n-20010315" />
                                <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"
                                />
                                <Reference URI="#tr">
                                        <DigestMethod
                                        Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
                                        <DigestValue>Nj+SlZIuNO0w5P66eUYbyvh4xqc=</DigestValue>
                                </Reference>
                        </SignedInfo>
                        <SignatureValue>q8HVJCoLnY3lvwEf1z1eVh5iyS8auLryxBjHQEjFOU0s4TCTbJBCa
                        GzUQJVuoQNIjgGC2BZeEQoodKwwgKHPTvuZPoFUUdUS6VufGq167hx+z78RaRfp
                        x0jJGM8knAB5YF0SgchboI3JvYxDQmx/rIOkBDqdBBMnSC8De4X8qSQ=</Signatur
                        eValue>
                        <KeyInfo>
                                <KeyValue>
                                <RSAKeyValue>
                                    <Modulus>o2uOmzXxRRt1o0u7ZEh7XN/0we3pNGz3riszG0G1HKcZy
                                KDKfwWSBWlpTaGO5TmkSMoU0lcSHzNjPA6cjk6/QAWO9xKt3+
                                mZjGDvcsDCZZQ+lKumlPeG7EReDhmpXyFwqIEK4Fx0nfP4pxJgFu
                                /8XD4CcWOP51ZYXuqvtWVsVgk=</Modulus>
                                <Exponent>AQAB</Exponent>
                                </RSAKeyValue>
                                </KeyValue>
                        </KeyInfo>
                </Signature>
        </Header>
        <Body>
                <CreditCard id="tr">
                        <CCNumber>1234 5678 9012 3456</CCNumber>
                        <Amount>3000</Amount>
                        <FirstName>John</FirstName>
                        <LastName>Smith</LastName>
                        <City>Famagusta</City>
                        <ZipCode>1234</ZipCode>
                </CreditCard>
        </Body>
</Envelop>
```

Figure 3.4: SOAP Message Signed Using ID Method

Another method, examined in this investigation was XPath, which according to McIntosh and Austel, 2005, could counter simple context and optional context [39]. In the analysis of XPath method, the signed part of message was placed in first, middle and at the end of the file and the results of all of them were the same, because of the fact that, XPath method is searching for all passible occurrences of the specified expressions. After finding one specified element, the function does not stop and it will continue to find another occurrence of the expression, inside the file. The result is shown as follows (see Figure 3.5 and Appendix B).

```
<Envelop>
        <Header>
                <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
                <SignedInfo>
                         <CanonicalizationMethod
                        Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"  />
                         <SignatureMethod   Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-
                        sha1" />
                         <Reference URI="">
                                <Transforms>
                                <Transform    Algorithm="http://www.w3.org/TR/1999/REC-xpath-
                        19991116">
                                 <XPath>ancestor-or-self::CreditCard[@id='tr']</XPath>
                                </Transform>
                        </Transforms>
                        <DigestMethod      Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"
                />
                        <DigestValue>Nj+SlZIuNO0w5P66eUYbyvh4xqc=</DigestValue>
                        </Reference>
                </SignedInfo>
                <SignatureValue>yKka68YHgCkWopn9HJupqBcGyggWkTvOXKDcMSi8ymOl
                uPTYTIJWftMdREyWOco6Ak+Vx1t1+jRntAGS2GJLOm39xwrL1JAKtNAk+G
                fLqJIsxT3511cUqG/BLkxbuZjviOKCGAVoMtrUCtvx0Mefbdmv7XyS6yMZ7h2
                hmLYw2Ww=</SignatureValue>
                <KeyInfo>
                 <KeyValue>
                <RSAKeyValue>
                <Modulus>p93LjAYkR+T72NViS+taNb2dnSg8WCDbfj9qwuMdqCaLCC7Ndg
                        KCxNS9pL9LfQihkQzqHEnq7ljR8tObuOOeqQUhtgdlzjVcZNtOK3yddx
                TRZZbWizmk2Jdj6Pbpd+0e1W4TkA1LwN0ZKcKkIRm1sw11H/Ac/SUdnxCxE
                OfMnG0=</Modulus>
                 <Exponent>AQAB</Exponent>
                </RSAKeyValue>
                </KeyValue>
                </KeyInfo>
                </Signature>
          </Header>
        <Body>
                <CreditCard id="tr">
                <CCNumber>1234  5678 9012 3456</CCNumber>
                <Amount>3000</Amount>
                <FirstName>John</FirstName>
                <LastName>Smith</LastName>
                <City>Famagusta</City>
                <ZipCode>1234</ZipCode>
                </CreditCard>
        </Body>
</Envelop>
```

Figure 3.5: SOAP Message Signed Using Xpath Method

Although this method (XPath) is so far, the best proposed method, but this method is very time consuming. The results of determination of time for finding the element for files with size between 750 Kb to 2250 Kb were as follows, in Table 3.2.

Table 3.2: Time Durations to Detect the Signing Element Using Xpath Method

Size (Kb)	Time consumed by XPath (ms)
50	6.0004
100	7.0003
150	8.003
250	10.0002
750	14.0008
1050	17.0004
1350	20.0006
1650	23.0014
1950	26.0016
2250	29.0016
3150	37.0013

The graph, shown in Figure 3.6, is showing the trend of above measured time durations.

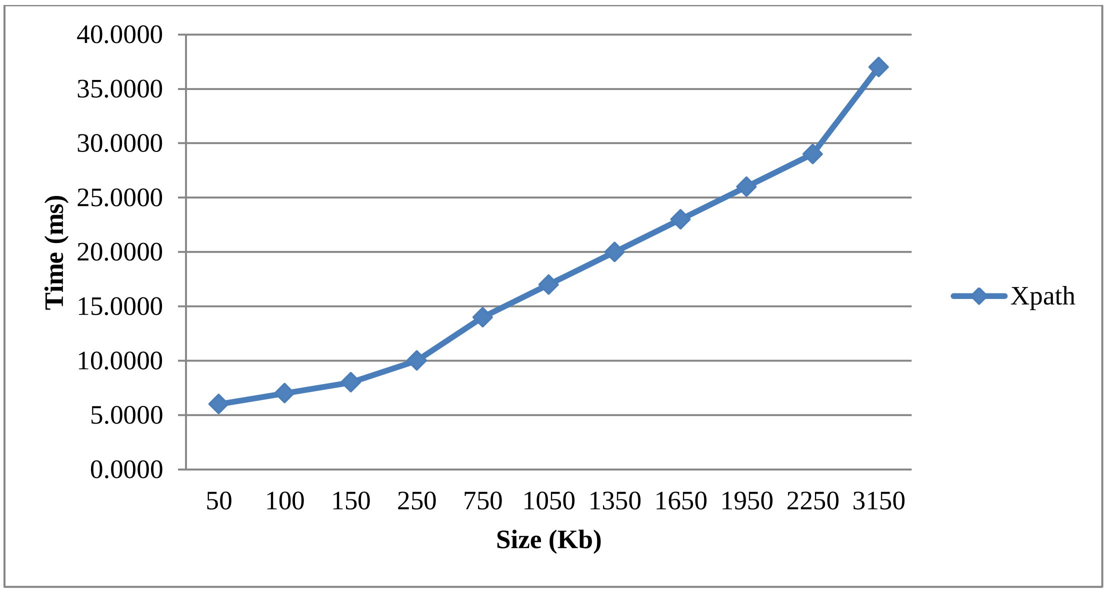


Figure 3.6: Time Consumed to Detect the Signing Element

As a conclusion to this evaluation and time determination, it can be said that although XPath is time consuming and still vulnerable to sibling optional context, so far it is the most sufficient method for countering XML Wrapping attack. On the other hand, although ID is a fast method, it is untrustworthy.

It should be added here that in the next chapter, it has been tried to explain about a new method, which is aimed to counter all the aforementioned vulnerabilities. Besides explaining, verifications have been done, to see the performance of the method.

3.4 Summary

In this chapter, firstly, after knowing about a typical available cloud system service, called Google App Engine and explaining its features, two of the available methods of referencing the signed part of SOAP Message to secure XML signature called ID and XPath referencing method, were explained and examined on a SOAP message.
It was observed that although both of the methods are being utilized to counter the XML signature wrapping attacks, they are not completely satisfying. XPath method is secure but it is also time consuming and it is not preventing Optional Sibling Value and Sibling Order Context attacks. On the other hand, ID method is fast but it is not completely safe and secure.

Chapter 4

PROPOSED SESOAP METHOD TO COUNTER XML SIGNATURE WRAPPING ATTACK IN SOAP MESSAGE

In this chapter, the new method, which is developed to counter what is so-called XML signature wrapping attack, will be explained comprehensively. In the first step, the logic behind the method and its structure is explained. This section is followed by explaining the performance of this method against different categories of wrapping attacks. In the last section, conducted experiments, to find verify the implementation of this method, are given out, which are done by using Visual Studio 2010 framework and C#.net programming language.

Since most of the XML signature wrapping attacks are done through changing the structure of the original SOAP message, sent by the genuine sender [28], it is logical to propose a protecting method, which aims to protect the structure of the sent message, from attacker. To fulfill this aim, the digital signature can be used to guarantee the integrity of message.

The method of this thesis, i.e. Signing Entire SOAP (SESoap) method, is to apply the digital signature structure over entire Soap envelop element, which results in securing the whole document. Consequently, an attacker will not be able to change the location of elements or remove or add any element to the original document. In the case of modification in any part of the document, the signature cannot be verified.

In the coming sections, the SESoap method will be examined in all types of attacks. The relevant results of each examination will be given in details, and discussions will be tried to be done about them. Skeleton of SESoap method and the XML document of SOAP message that is signed by SESoap method is shown in the following figures (Figure 4.1 and 4.2).

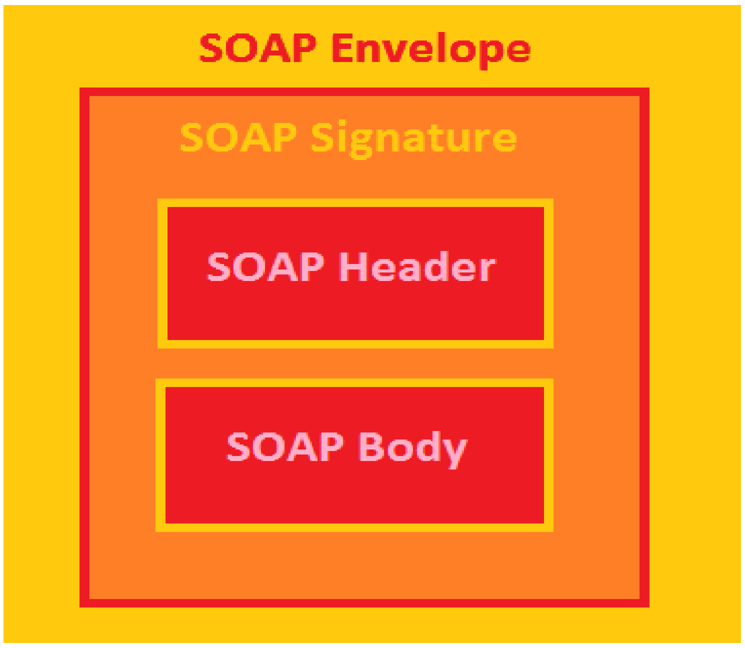
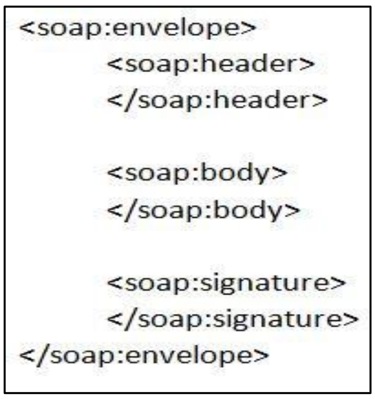


Figure 4.1: Skeleton of SESoap method



```
<soap:envelope>
        <soap:header>
        </soap:header>

        <soap:body>
        </soap:body>

        <soap:signature>
        </soap:signature>
</soap:envelope>
```

Figure 4.2: XML Document of SOAP Messaged Signed Using Sesoap Method

It should be noted that the element of Soap:signature, is the resulted by signing the entire content of soap:envelop, except the element of soap:signature itself. To explain better, the structure of SOAP after applying the SESoap method, have been shown in Figure 4.2.

4.1 Simple Element Context

In simple Context attack, a wrapper alters the location of the Soap body and adds a new Soap body to threaten the SOAP document [39]. It is quite clear that by using digital signature over entire document, any alteration or adding any element to the signed document will be totally prevented.

4.2 Optional Element Context

In optional Element context, a wrapper adds some information to optional element to application logic of a program could not pars that element [39]. Again, the same as the previous attack, when a wrapper tends to add something to the document, the attack is prevented by SESoap.

4.3 Sibling Value Context

The two previous types of attacks are possible to be prevented by means of XPath method [39]; however XPath is susceptible against this attack [29]. As it has been explained in the previous chapter, Time stamp element, which is an optional sibling element of signature element, can be threatened by wrapper. But for wrapping on this element, the wrapper again must modify some parts of document [39]. Consequently, as modifications are prevented in SESoap method, Sibling value Context will not be allowed to occur.

4.4 Sibling Order attack

This attack relies on changing the order of individual sibling elements [39]. Therefore, since reordering is also not possible in SESoap, again no wrapper can be successful in implementation of this attack.

4.5 Conducted Experiments

SESoap method has been implemented by using C#.net, in order to determine how fast it is, comparing to the previous methods of ID and XPath. These examinations have also been performed by means of Laptop, having 2.00 GHz Core2Duo CPU, and 1.00 GB memory in Windows 7. The time duration for finding specified element inside the SOAP body element of the document, in three methods as it is shown in Table 4.1.

Table 4.1: Time Durations for Finding Specified Element

Size of files (Kb)	Time of finding an element (ms)		
	ID	Xpath	SESoap
50	0	6.0004	0
100	0	7.0003	0
150	1.0003	8.003	0
250	1.0006	10.0002	0
750	3.0002	14.0008	0
1050	4.0003	17.0004	0
1350	5.0003	20.0006	0
1650	6.0004	23.0014	0
1950	7.0003	26.0016	0
2250	8.0004	29.0016	0
3150	11.0034	37.0013	0

The time durations for finding the element in SESoap are 0, because this technique does not search for any specified element inside the SOAP document. The graph for comparing these time durations, have been shown in Figure 4.3.

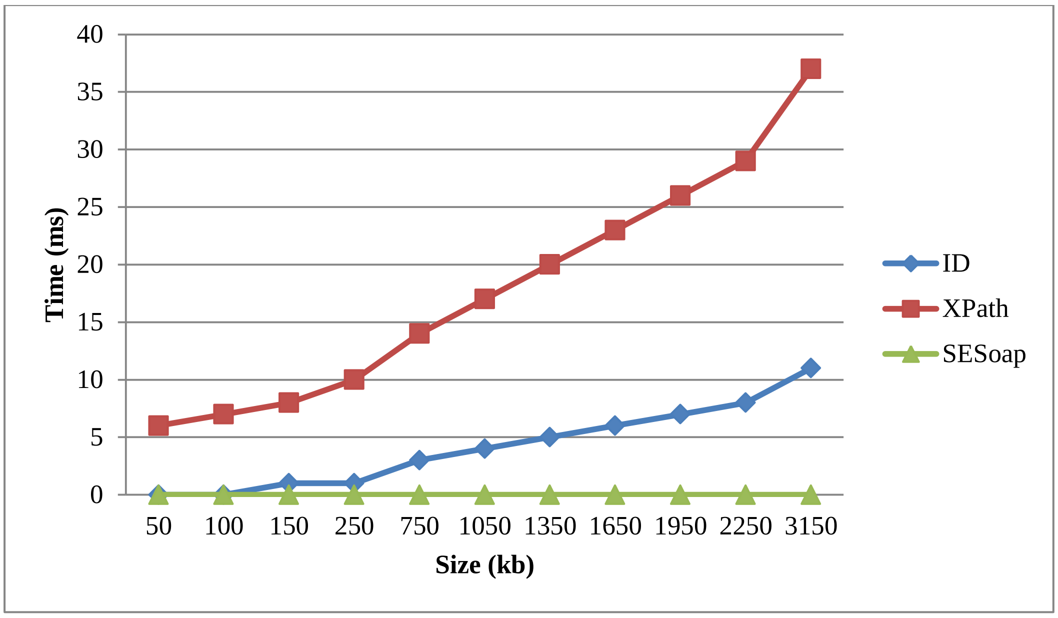


Figure 4.3: Time Durations of the Three Considered Methods

In the next step, the time durations for hashing the specified element inside the SOAP document, have been estimated. The results are shown in Table 4.2.

Table 4.2: Time Durations for Hashing the Specified Element

Size of files (Kb)	Time of Hash function (ms)		
	ID	Xpath	SESoap
50	0	0	0
100	0	0	0
150	0	0	1.0001
250	0	0	1.0012
750	1.0001	1.0001	3.0001
1050	1.0001	1.0001	4.0002
1350	1.0001	1.0001	5.0003
1650	1.0001	1.0001	6.0004
1950	1.0001	1.0001	7.0005
2250	1.0001	1.0001	8.0005
3150	1.0001	1.0001	11.0087

The graph below (Figure 4.4) is showing the trend of the evaluated numbers.

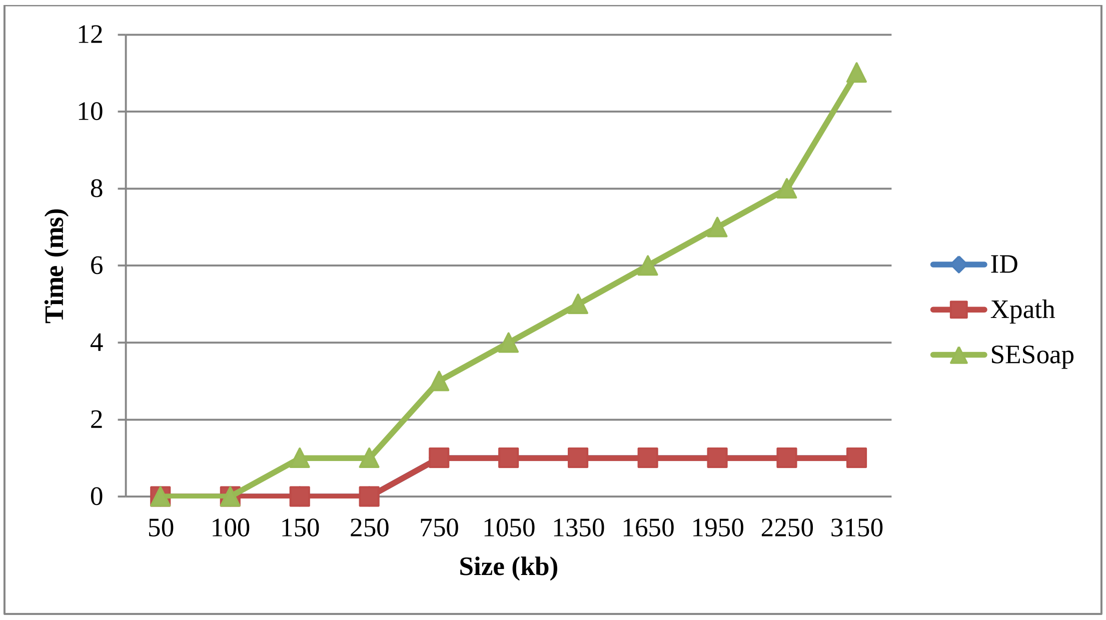


Figure 4.4: Time Durations for Hashing the Specified Element

In addition, the consumed time for encrypting data, in all the three methods are equal, because in the digital signature, encryption function applies on the signed info element of signature. The sizes of the signed info element in all the methods are equal. As the result, the consumed times for encrypting the signed info elements are the same. In this study RSA key is used to encrypt the signed info element of the signature and the time consumed for all three methods was 3.0004 in milliseconds. In this study two types of codes are used to estimate the total time of signing the message. These codes are named as Code1 and Code2. In Code1 the whole operations are done as one component and in Code2 each function (finding element, hashing and encryption) is done separately [46]. The total consumed times to sign the SOAP message, in each of the three methods by Code 1 are as Table 4.3 (See also appendix D).

Table 4.3: Total Times Consumed to Sign the Soap Message, in Each Method Using Code2

Size of files (Kb)	Total time (ms)		
	ID	XPath	SESoap
50	3.0004	9.0008	3.0004
100	3.0004	10.0007	3.0004
150	4.0007	11.0034	4.0005
250	4.0012	14.0006	4.0016
750	7.0027	18.0031	6.0024
1050	8.0028	21.0027	7.0025
1350	9.0028	24.0029	8.0026
1650	10.0029	27.0037	9.0027
1950	11.0028	30.0039	10.0030
2250	12.0029	33.0039	12.0030
3150	15.0003	40.0037	15.0009

According to these results, as the numbers show, the total consumed time to sign a SOAP document by SESoap method is almost three times as much as XPath method and approximately equal to ID. Consequently, it can be claimed that, the SESoap method is operating more sufficiently, than the other two methods, considering both aspects of security and time.

The results of running code 1 to sign the specified element [46] are as the Table 4.4 (See also Appendix D).

Table 4.4: Total Times Consumed to Sign the Soap Message, Using Code1

Size of files (Kb)	Total time (ms)		
	ID	XPath	SESoap
50	3.0012	10.0011	4.0023
100	4.0005	13.0012	7.0003
150	6.0006	33.0003	11.0012
250	8.0009	48.0004	15.0023
750	15.0007	148.0014	28.0001
1050	21.0007	207.0234	40.0012
1350	36.0014	269.0002	53.0013
1650	42.0003	329.0019	64.0015
1950	53.0012	341.0015	76.0245
2250	68.0023	401.0023	98.0001
3150	113.0003	590.0145	134.0231

These results are more complying with the previous research [28], but as it can be obviously noticed, the results of that research are less efficient than what is done in this study.

4.6 Summary

In this chapter, a new method for securing the SOAP message is proposed, in which the concept of digital signature to fix the location of elements and avoid of entering new malicious elements inside the SOAP message is used. Analyzing of the method and comparing it to the other available methods are also conducted in this chapter. The results illustrated that the new method is faster and more secure than the other available methods.

Chapter 5

CONCLUSIONS

The primary goal of this study was to secure SOAP message, which is employed to exchange information between web applications of cloud computing systems. Having this aim, a new method –SESoap– has been proposed. The concept of this method is using Digital Signature technique to immune the information inside a SOAP message from modification by an adversary.

Google App engine has been investigated to get familiar with cloud computing systems. It is an example of cloud computing systems that provides infrastructure, to build web applications.

ID and XPath methods for signing SOAP message have been analyzed and compared from two viewpoints, i.e. security and the consumed time to find the signing element. The result of these investigations illustrated that, the ID method is a fast method with significant security concerns and in contrast, XPath is a safer method, but slow.

SESoap method, which is proposed in this investigation, signs the SOAP message in a different way. In the current countermeasures, such as XPath and ID, just one part of SOAP document, which specified by these methods, is signed, while in SESoap method, the entire information inside the SOAP envelope is used as an input of signing function. In this case, modification of the SOAP message will be impossible.

The results obtained from implementation of SESoap method indicate that, this method is slower than the other examined methods, for hashing the information. The reason of this observation is that, comparing to the other examined methods, in this method, the hash function is applied over a greater size of data. On the other hand, for finding element in SOAP message, SESoap method does not consume any time. In this study a more efficient way is utilized to sign the SOAP message, and the total time duration to sign the message is approximately one third of the total time consumed in XPath method.

REFERENCES

[1] P. Mell and T. Grance, "COMPUTER SECURITY," September 2011. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf. [Accessed 30 April 2013].

[2] S. Bhayal, "A study of security in cloud computing," ProQuest Dissertation Publishing, 2011.

[3] Small Business Technology, "Infrastructure as a service.," 2012. [Online]. Available: http://www.biztechmagazine.com/sites/default/files/cloud-iaas_2.pdf. [Accessed 28 April 2013].

[4] K. L. Jackson and C. Landis, "Platform as a Service (PaaS)," January 2012. [Online]. Available: http://appdeployer.com/home/wp-content/uploads/2012/12/NJVC-Virtual-Global-PaaS-White-Paper.pdf. [Accessed 20 April 2013].

[5] J. Brown, "What is SaaS?," [Online]. Available: http://www.powerdms.com/downloads/ebook/Moving-Toward-the-Cloud-What-is-SaaS.pdf. [Accessed 20 April 2013].

[6] D. Jamil and H. Zaki, "Security Issues in Cloud Computing and Countermeasures," *International Journal of Engineering Science and*

Technology, vol. 3, no. 4, pp. 2672-2676, 2011.

[7] W3C, "SOAP Specifications," W3C, 2007. [Online]. Available: http://www.w3.org/TR/soap/. [Accessed 28 April 2013].

[8] W3C, "Extensible Markup Language (XML)," W3C, 2012. [Online]. Available: http://www.w3.org/XML/. [Accessed 25 April 2013].

[9] N. Gruschka and L. Lo Iacono, "Vulnerable Cloud: SOAP message security validation revisited," in *2009 IEEE International Conference on Web Services*, Los Angeles, 2009.

[10] A. Milroy, "16 Key Attributes of Cloud Computing," January 2011. [Online]. Available: http://www.andymilroy.com/2011/01/16-key-attributes-of-cloud-computing.html. [Accessed 30 April 2013].

[11] Perspectives on Cloud Computing & Training from Learning Tree International, "Cloud Service Models: Comparing SaaS PaaS and IaaS | Perspectives on Cloud Computing & Training from Learning Tree International," November 2011. [Online]. Available: http://cloud-computing.learningtree.com/2011/11/09/cloud-service-models-comparing-saas-paas-and-iaas/. [Accessed 27 April 2013].

[12] M. Rouse, "What is public cloud? - Definition from WhatIs.com," Cloud computing information, news and tips, May 2009. [Online]. Available: http://searchcloudcomputing.techtarget.com/definition/public-cloud.

[Accessed 29 April 2013].

[13] M. Rouse, "What is hybrid cloud?," June 2010. [Online]. Available: http://searchcloudcomputing.techtarget.com/definition/hybrid-cloud. [Accessed 30 April 2013].

[14] Amazon Web Services, "Amazon Virtual Private Cloud," 2013. [Online]. Available: http://aws.amazon.com/vpc/. [Accessed 14 May 2013].

[15] "SOAP – Simple Object Access Protocol for Message Negotiating and Transmission," The Two Way Web, [Online]. Available: http://www.thetwowayweb.com/soapmeetsrss. [Accessed 15 May 2013].

[16] "SOAP (Simple Object Access Protocol) Definition," 2013. [Online]. Available: http://www.techterms.com/definition/soap. [Accessed 17 May 2013].

[17] W3Schools, "SOAP Introduction," W3Schools, [Online]. Available: http://www.w3schools.com/soap/soap_intro.asp. [Accessed 17 May 2013].

[18] w3schools, "SOAP Syntax- SOAP Building Blocks," 2013. [Online]. Available: http://www.w3schools.com/soap/soap_syntax.asp. [Accessed 6 May 2013].

[19] World Wide Web Consortium, "Web Services Glossary," 2004. [Online]. Available: http://www.w3.org/TR/2004/NOTE-ws-gloss-

20040211/#webservice. [Accessed 2 May 2013].

[20] J. Hwang, "Web services SOAP message validation," IBM, 2010. [Online]. Available: http://www.ibm.com/developerworks/webservices/library/ws-soapvalid/#N10078. [Accessed 15 May 2013].

[21] R. Salz, "Understanding XML Digital Signature," July 2003. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms996502.aspx. [Accessed 19 May 2013].

[22] M. Bartel, J. Boyer, B. Fox, B. LaMacchia and E. Simon, "XML Signature Syntax and Processing Version 1.1," World Wide Web Consortium (W3C), April 2013. [Online]. Available: http://www.w3.org/TR/2013/REC-xmldsig-core1-20130411/. [Accessed 29 April 2013].

[23] C. O. h. Eigeartaigh, "Open Source Security: XML Signature Wrapping attacks on Web Services," October 2012. [Online]. Available: http://coheigea.blogspot.com/2012/10/xml-signature-wrapping-attacks-on-web.html. [Accessed 20 May 2013].

[24] M. McIntosh and P. Austel, "XML Signature Element Wrapping Attacks and Countermeasures," IBM Research Report, NewYork, 2005.

[25] "XML Signature Wrapping - Simple Context -," WS-Attacks.org, 2011. [Online]. Available: http://clawslab.nds.rub.de/wiki/index.php/XML_Signature_Wrapping_-

_Simple_Context. [Accessed 4 April 2013].

[26] "XML Signature Wrapping - Optional Element," WS-Attacks.org, April 2011. [Online]. Available: http://clawslab.nds.rub.de/wiki/index.php/XML_Signature_Wrapping_-_Optional_Element. [Accessed 4 May 2013].

[27] "XML Signature Wrapping - Optional Element in Security Header -," WS-Attacks.org, 2010. [Online]. Available: http://clawslab.nds.rub.de/wiki/index.php/XML_Signature_Wrapping_-_Optional_Element_in_Security_Header. [Accessed 4 May 2013].

[28] C. Mainka, M. Jensen, L. I. Luigi and J. Schwenk, "XSPRES: ROBUST AND EFFECTIVE XML SIGNATURES FOR WEB SERVICES," in *2nd International Conference on Cloud Computing and Services Science*, Porto, 2012.

[29] N. Gruschka and L. Lo Iacono, "Vulnerable Cloud: SOAP Message Security Validation Revisited," in *2009 IEEE International Conference on Web Services*, Maimi, 2009.

[30] M. A. Rahaman, A. Schaad and M. Rits, "Towards secure SOAP message exchange in a SOA," in *3rd ACM workshop on Secure web services*, New York, 2006 .

[31] S. Gajek, L. Liao and J. Schwenk, "Breaking and fixing the inline approach,"

in *ACM workshop on Secure web services*, New York, 2007 .

[32] K. Bhargavan, C. Fournet, A. D. Gordon and G. O'Shea, "An advisor for web services security policies," in *2005 workshop on Secure web services*, New York, 2005 .

[33] K. Lawrence and C. Kaler, "WS-Trust 1.3," Advancing open standards for the information society, March 2007. [Online]. Available: http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.html. [Accessed 29 May 2013].

[34] S. Gajek, M. Jensen, L. Liao and J. Schwenk, "Analysis of Signature Wrapping Attacks and Countermeasures," in *ICWS 2009. IEEE International Conference on Web Services*, Bochum, 2009 .

[35] Google Developers, "What Is Google App Engine?," Google, April 2013. [Online]. Available: https://developers.google.com/appengine/docs/whatisgoogleappengine. [Accessed 8 May 2013].

[36] D. Sanderson, "Introducing Google App Engine," in *Programming Google App Engine*, Sebastopol, O’Reilly Media, 2009, pp. 1-15.

[37] Google Developers, "Quotas," Google, April 2013. [Online]. Available: https://developers.google.com/appengine/docs/quotas. [Accessed 28 April

2013].

[38] M. Rudominer, "HOW TO: Build a SOAP Server and a SOAP Client on Google App Engine," Google, February 2011. [Online]. Available: https://developers.google.com/appengine/articles/soap. [Accessed 12 May 2013].

[39] M. McIntosh and P. Austel, "XML signature element wrapping attacks and countermeasures," IBM research division, 2005.

[40] W3Schools, "SOAP Tutorial," 2013. [Online]. Available: http://www.w3schools.com/soap/default.asp. [Accessed 30 April 2013].

[41] W3C, "SOAP Security Extensions: Digital Signature," W3C, 2013. [Online]. Available: http://www.w3.org/TR/SOAP-dsig/. [Accessed 28 April 2013].

[42] W3C, "XML Signature Syntax and Processing (Second Edition)," W3C, 2013. [Online]. Available: http://www.w3.org/TR/xmldsig-core/. [Accessed 29 April 2013].

[43] T. Berners-Lee, R. Fielding and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," August 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2396.txt. [Accessed 24 April 2013].

[44] S. Gajek, M. Jensen, L. Liao and J. Schwenk, "Analysis of Signature Wrapping Attacks and Countermeasures," [Online]. Available:

http://datenschutz.web-schell.de/tl_files/web-datenschutz-schell/Website-Dateien/PDF/wrapping-attacks_and_countermeasures.pdf. [Accessed 28 April 2013].

[45] W3C, "XML Path Language (XPath)," W3C, [Online]. Available: http://www.w3.org/TR/xpath/. [Accessed 30 April 2013].

[46] Microsoft, "XmlDsigXPathTransform Class (System.Security.Cryptography.Xml)," Microsoft, 2013. [Online]. Available: http://msdn.microsoft.com/en-us/library/system.security.cryptography.xml.xmldsigxpathtransform(v=vs.85).aspx. [Accessed 30 June 2013].

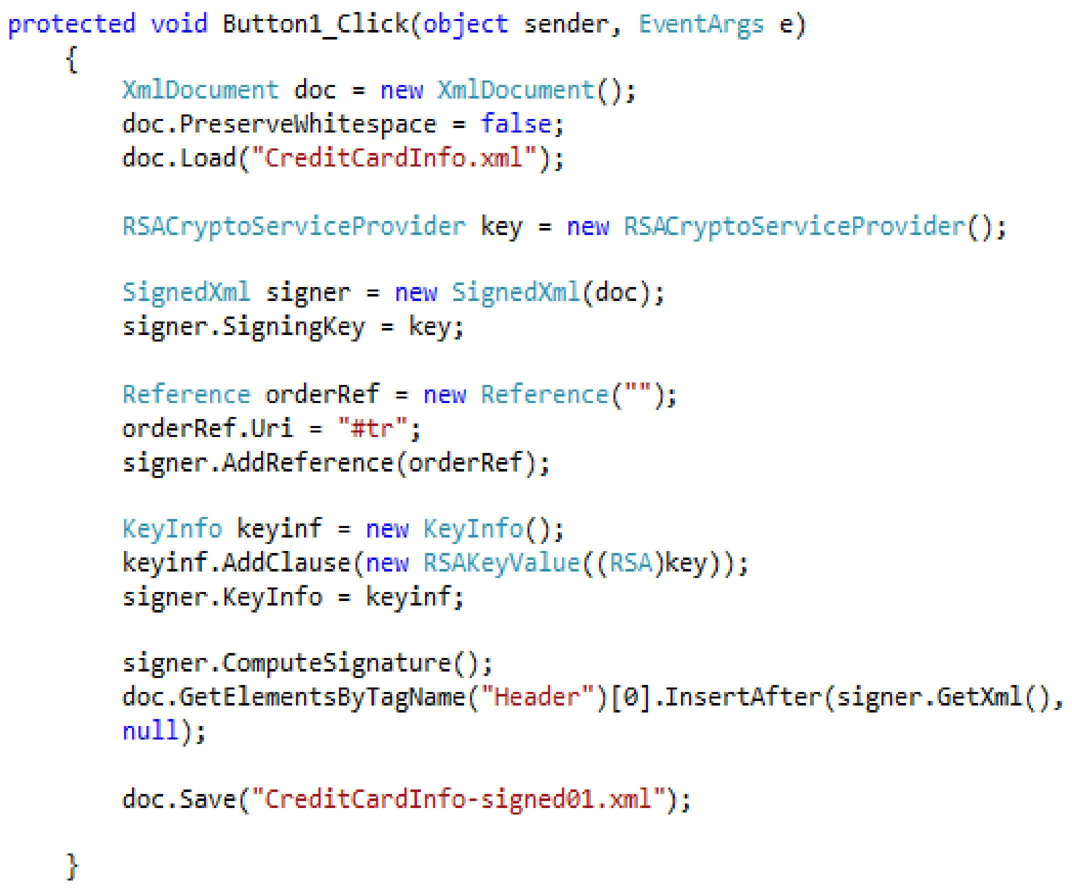
[47] M. Jensen, J. Schwenk, N. Gruschka and L. Lo Iacono, "On Technical Security Issues in Cloud Computing," in *2009 IEEE International Conference on Cloud Computing*, 2009.

[48] IBM, "WebSphere Application Server Version 6.1," 2013. [Online]. Available: http://pic.dhe.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=%2Fcom.ibm.websphere.base.doc%2Finfo%2Faes%2Fae%2Fcwbs_wssmessage.html. [Accessed 6 May 2013].

[49] B. Peng, B. Cui and X. Li, "Implementation Issues of a Cloud Computing Platform," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering,* pp. 59-67, 2009.

APPENDICES

Appendix A: Code Used to Sign SOAP Message by Means of ID Referencig Method

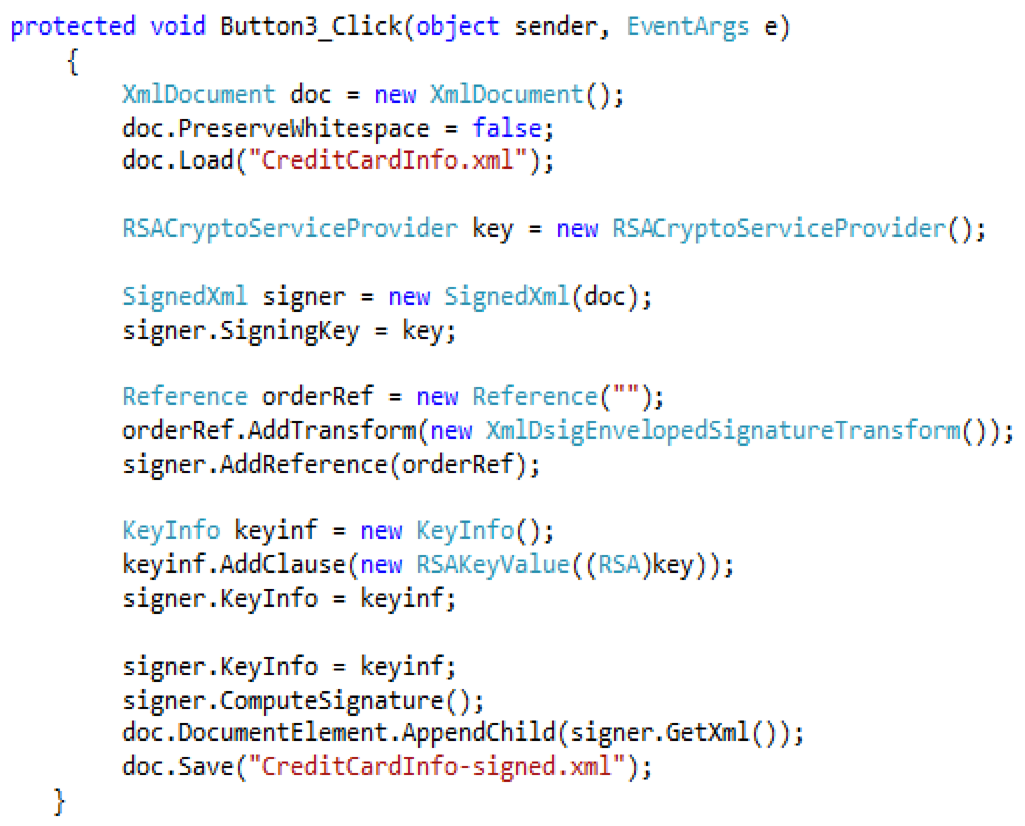


```
protected void Button1_Click(object sender, EventArgs e)
    {
        XmlDocument doc = new XmlDocument();
        doc.PreserveWhitespace = false;
        doc.Load("CreditCardInfo.xml");

        RSACryptoServiceProvider key = new RSACryptoServiceProvider();

        SignedXml signer = new SignedXml(doc);
        signer.SigningKey = key;

        Reference orderRef = new Reference("");
        orderRef.Uri = "#tr";
        signer.AddReference(orderRef);

        KeyInfo keyinf = new KeyInfo();
        keyinf.AddClause(new RSAKeyValue((RSA)key));
        signer.KeyInfo = keyinf;

        signer.ComputeSignature();
        doc.GetElementsByTagName("Header")[0].InsertAfter(signer.GetXml(),
        null);

        doc.Save("CreditCardInfo-signed01.xml");

    }
```

Appendix B: Code Used to Sing SOAP Message by Means of Xpath

```
protected void Button2_Click(object sender, EventArgs e)
    {

        XmlDocument doc = new XmlDocument();
        doc.PreserveWhitespace = false;
        doc.Load("CreditCardInfo.xml");

        RSACryptoServiceProvider key = new RSACryptoServiceProvider();

        SignedXml signer = new SignedXml(doc);
        signer.SigningKey = key;

        Reference orderRef = new Reference("");
        orderRef.Uri = "";
        XmlDsigXPathTransform Xpathtransform = CreateXPathTransform("ancestor-
        or-self::CreditCard[@id='tr']");
        orderRef.AddTransform(Xpathtransform);
        signer.AddReference(orderRef);

        KeyInfo keyinf = new KeyInfo();
        keyinf.AddClause(new RSAKeyValue((RSA)key));
        signer.KeyInfo = keyinf;

        signer.ComputeSignature();
        doc.GetElementsByTagName("Header")[0].InsertAfter(signer.GetXml(),
        null);

        doc.Save("CreditCardInfo-signed.xml");

    }

private static XmlDsigXPathTransform CreateXPathTransform(string XPathString)

  {
      // Create a new XMLDocument object.
      XmlDocument doc = new XmlDocument();

      // Create a new XmlElement.
      XmlElement xPathElem = doc.CreateElement("XPath");

      // Set the element text to the value
      // of the XPath string.
      xPathElem.InnerText = XPathString;

      // Create a new XmlDsigXPathTransform object.
      XmlDsigXPathTransform xForm = new XmlDsigXPathTransform();

      // Load the XPath XML from the element.
      xForm.LoadInnerXml(xPathElem.SelectNodes("."));

      // Return the XML that represents the transform.
      return xForm;
  }
```

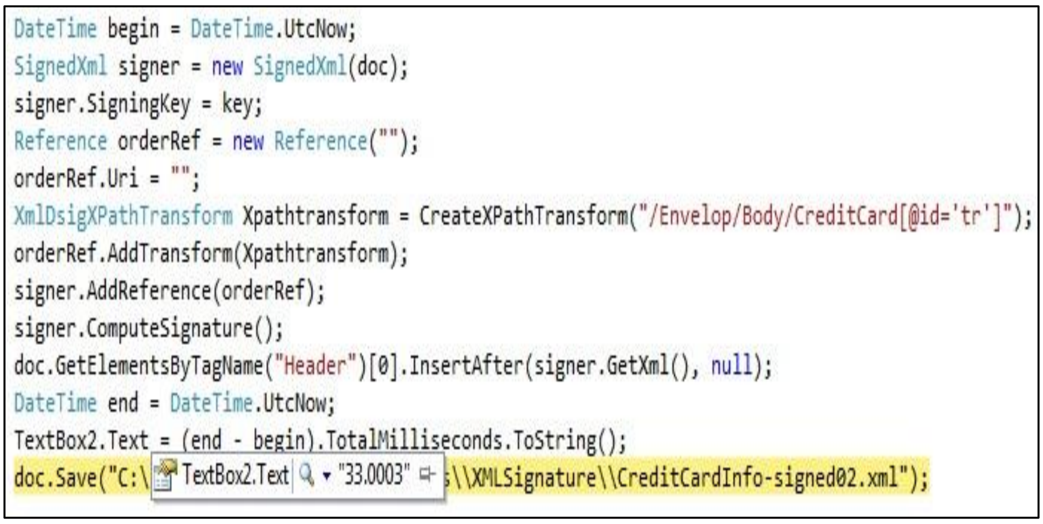
Appendix C: Code Used to Sign the SOAP Message by Means of SESoap Message



```
protected void Button3_Click(object sender, EventArgs e)
    {
        XmlDocument doc = new XmlDocument();
        doc.PreserveWhitespace = false;
        doc.Load("CreditCardInfo.xml");

        RSACryptoServiceProvider key = new RSACryptoServiceProvider();

        SignedXml signer = new SignedXml(doc);
        signer.SigningKey = key;

        Reference orderRef = new Reference("");
        orderRef.AddTransform(new XmlDsigEnvelopedSignatureTransform());
        signer.AddReference(orderRef);

        KeyInfo keyinf = new KeyInfo();
        keyinf.AddClause(new RSAKeyValue((RSA)key));
        signer.KeyInfo = keyinf;

        signer.KeyInfo = keyinf;
        signer.ComputeSignature();
        doc.DocumentElement.AppendChild(signer.GetXml());
        doc.Save("CreditCardInfo-signed.xml");
    }
```

Appendix D: Snap shot of the codes used to analyze time

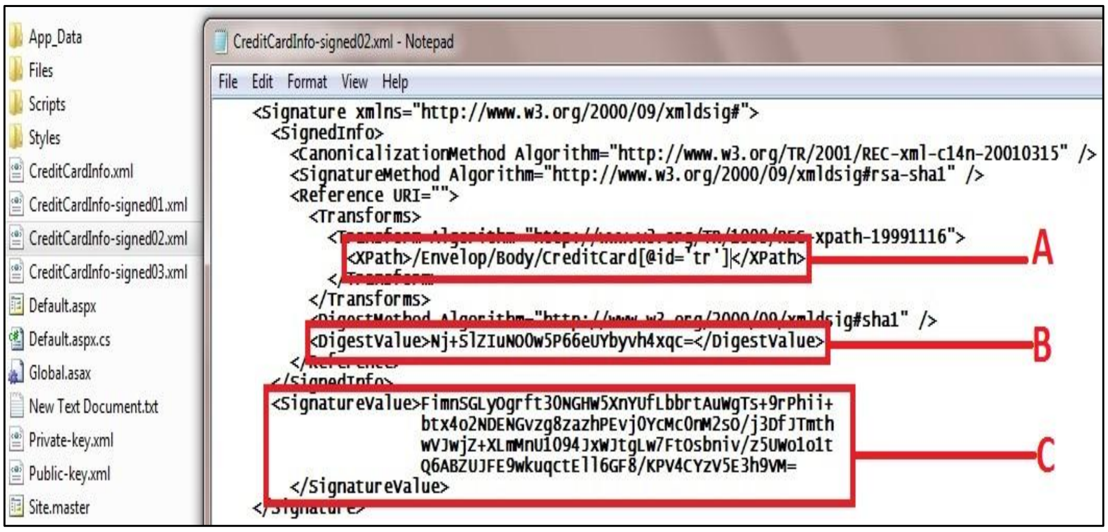
In the snap shots below of the analyzing the time, an XML file with 150 kb size is used as an input of the two codes. The name of the input file is CreditCardinfo.xml. The result shows that the time consumed for signing by Code1 is 3.0004 milliseconds. Figure D.1 shows how the code1 is working.



```
DateTime begin = DateTime.UtcNow;
SignedXml signer = new SignedXml(doc);
signer.SigningKey = key;
Reference orderRef = new Reference("");
orderRef.Uri = "";
XmlDsigXPathTransform Xpathtransform = CreateXPathTransform("/Envelop/Body/CreditCard[@id='tr']");
orderRef.AddTransform(Xpathtransform);
signer.AddReference(orderRef);
signer.ComputeSignature();
doc.GetElementsByTagName("Header")[0].InsertAfter(signer.GetXml(), null);
DateTime end = DateTime.UtcNow;
TextBox2.Text = (end - begin).TotalMilliseconds.ToString();
doc.Save("C:\ TextBox2.Text "33.0003" s\\XMLSignature\\CreditCardInfo-signed02.xml");
```

Figure D.1. Code1

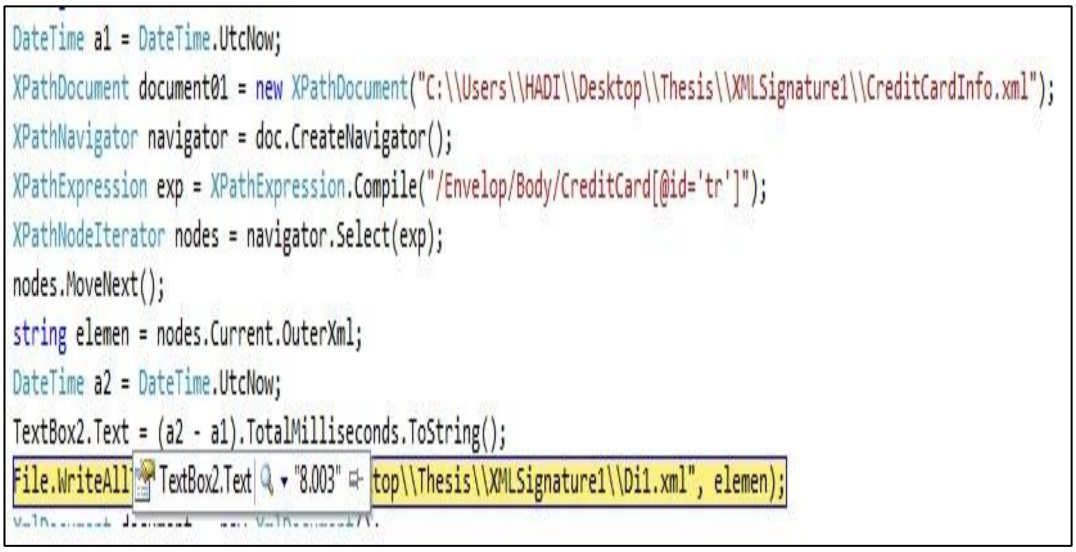
The result of Code1 is as the Figure D.2. The File is consisting of A. Xpath expression, B. Digest value and C. Signature value.



```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
    <Reference URI="">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
          <XPath>/Envelop/Body/CreditCard[@id='tr']</XPath>
        </Transform>
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
      <DigestValue>Nj+SlZIuNO0w5P66eUYbyvh4xqc=</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>FimnSGLyOgrft30NGHW5XnYUfLbbrtAuWgTs+9rPhii+
                  btx4o2NDENGvzg8zazhPEvjOYcMcOnM2sO/j3DfJTmth
                  wVJwjZ+XLmMnU1094JxWJtgLw7FtOsbniv/z5UWo1o1t
                  Q6ABZUJFE9wkuqctEll6GF8/KPV4CYzV5E3h9VM=
    </SignatureValue>
</Signature>
```

Figure D.2. Result of Code1

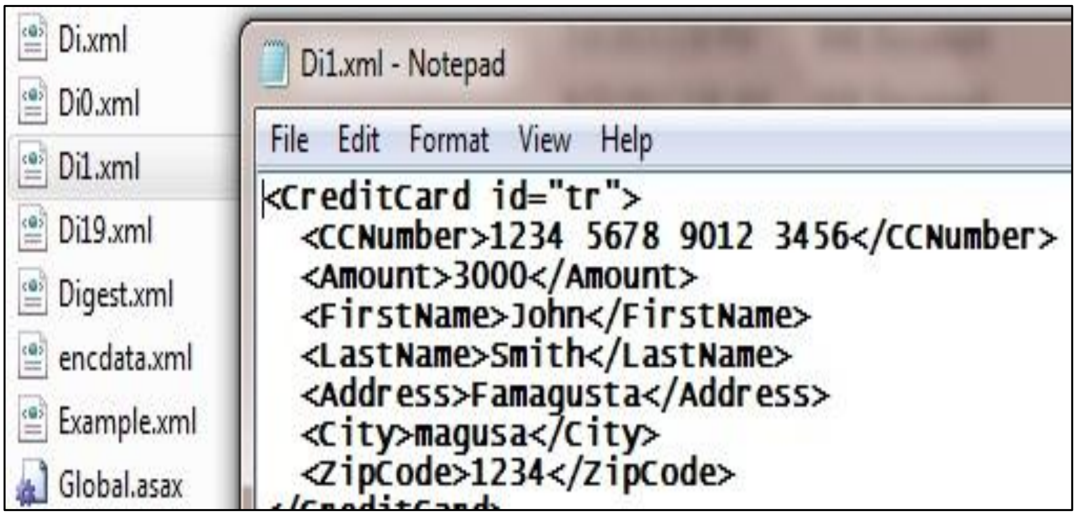
In code 2, three functions which are finding element, Hash function and encryption function are operating separately. First one is Xpath function which is shown in Figure D.3. The time consumed for this function is 8.003 milliseconds.



```
DateTime a1 = DateTime.UtcNow;
XPathDocument document01 = new XPathDocument("C:\\Users\\HADI\\Desktop\\Thesis\\XMLSignature1\\CreditCardInfo.xml");
XPathNavigator navigator = doc.CreateNavigator();
XPathExpression exp = XPathExpression.Compile("/Envelop/Body/CreditCard[@id='tr']");
XPathNodeIterator nodes = navigator.Select(exp);
nodes.MoveNext();
string elemen = nodes.Current.OuterXml;
DateTime a2 = DateTime.UtcNow;
TextBox2.Text = (a2 - a1).TotalMilliseconds.ToString();
File.WriteAll  TextBox2.Text  "8.003"  top\\Thesis\\XMLSignature1\\Di1.xml", elemen);
```

Figure D.3. XPath function of Code2

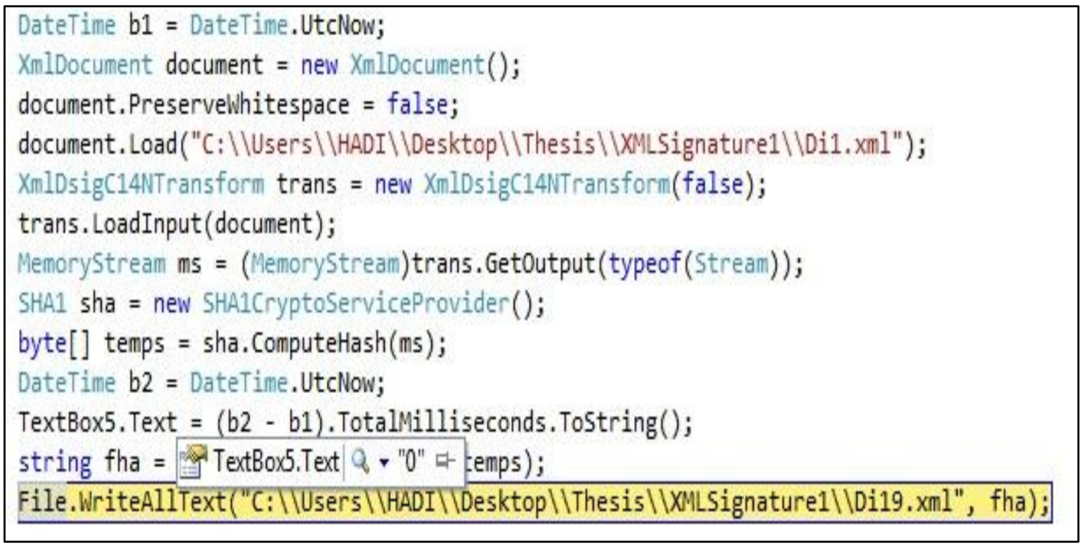
The Xpath expression in the code is equal to the section A of the Figure D.2. The result of Xpath function is as shown in Figure D.4.



```
<CreditCard id="tr">
  <CCNumber>1234 5678 9012 3456</CCNumber>
  <Amount>3000</Amount>
  <FirstName>John</FirstName>
  <LastName>Smith</LastName>
  <Address>Famagusta</Address>
  <City>magusa</City>
  <ZipCode>1234</ZipCode>
```

Figure D.4 Result of Xpath Function

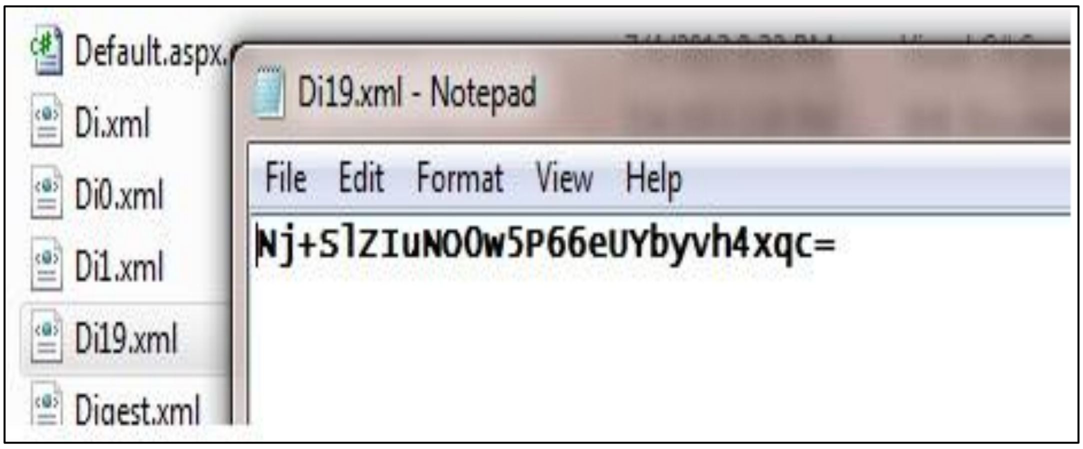
Figure D.5 is shown the code used for hash function. The time consumed for this function is 0 milliseconds.



```
DateTime b1 = DateTime.UtcNow;
XmlDocument document = new XmlDocument();
document.PreserveWhitespace = false;
document.Load("C:\\Users\\HADI\\Desktop\\Thesis\\XMLSignature1\\Di1.xml");
XmlDsigC14NTransform trans = new XmlDsigC14NTransform(false);
trans.LoadInput(document);
MemoryStream ms = (MemoryStream)trans.GetOutput(typeof(Stream));
SHA1 sha = new SHA1CryptoServiceProvider();
byte[] temps = sha.ComputeHash(ms);
DateTime b2 = DateTime.UtcNow;
TextBox5.Text = (b2 - b1).TotalMilliseconds.ToString();
string fha = TextBox5.Text "0" temps);
File.WriteAllText("C:\\Users\\HADI\\Desktop\\Thesis\\XMLSignature1\\Di19.xml", fha);
```

Figure D.5 Hash Function of Code2

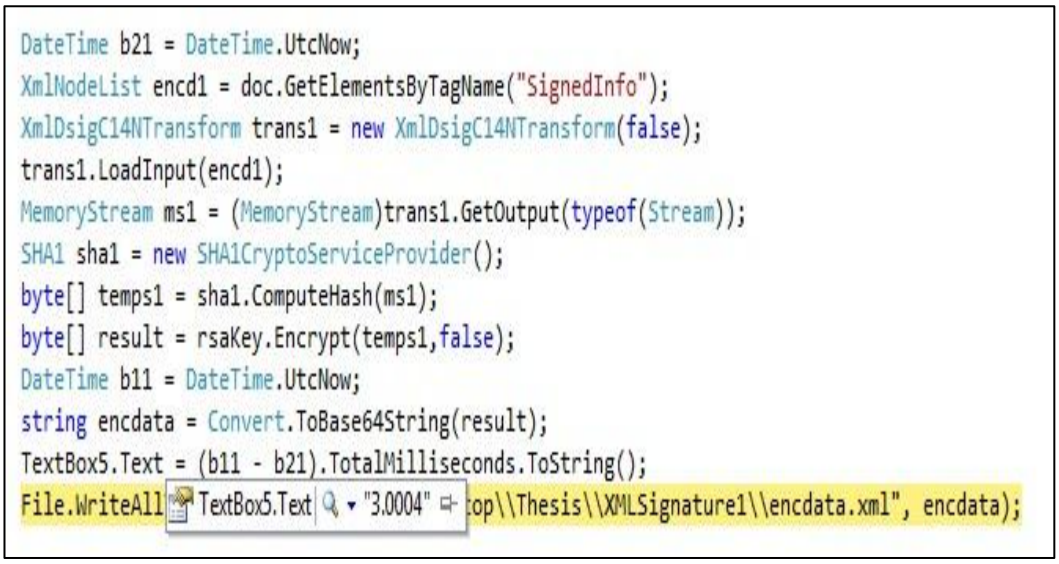
The result of the hash function is equal to section B of the Figure D.2 which is shown in Figure D.6.



```
Nj+SlZIuNOOw5P66eUYbyvh4xqc=
```

Figure D.6. Reslut of Hash Function

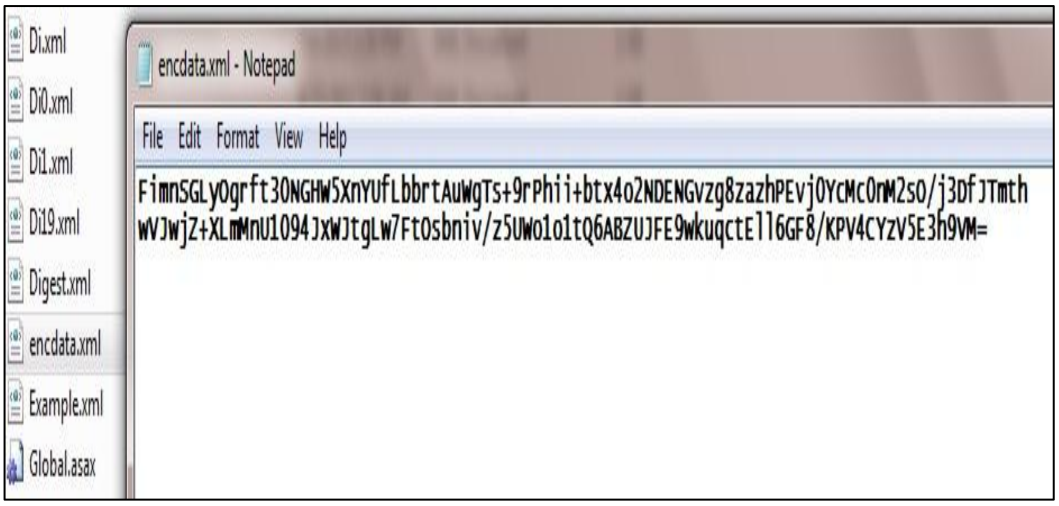
And the last one is Encryption function. The time consumed for this function is 3.0004 milliseconds. The code of this function can be seen in Figure D.7.



```
DateTime b21 = DateTime.UtcNow;
XmlNodeList encd1 = doc.GetElementsByTagName("SignedInfo");
XmlDsigC14NTransform trans1 = new XmlDsigC14NTransform(false);
trans1.LoadInput(encd1);
MemoryStream ms1 = (MemoryStream)trans1.GetOutput(typeof(Stream));
SHA1 sha1 = new SHA1CryptoServiceProvider();
byte[] temps1 = sha1.ComputeHash(ms1);
byte[] result = rsaKey.Encrypt(temps1,false);
DateTime b11 = DateTime.UtcNow;
string encdata = Convert.ToBase64String(result);
TextBox5.Text = (b11 - b21).TotalMilliseconds.ToString();
File.WriteAll TextBox5.Text "3.0004" top\\Thesis\\XMLSignature1\\encdata.xml", encdata);
```

Figure D.7 Encryption Function of the Code2

Figure D.8 contains the result of encryption function. This result is as the same as section C of Figure D.2.



```
FimnSGLyOgrft3ONGHW5XnYUfLbbrtAuWgTs+9rPhii+btx4o2NDENGvzg8zazhPEvj0YcMcOnM2sO/j3DfJTmth
wVJwjZ+XLmMnU1094JxWJtgLw7FtOsbniv/z5UWo1o1tQ6ABZUJFE9wkuqctEl16GF8/KPV4CYzV5E3h9VM=
```

Figure D.8 Result of Encryption Function

The comparison of the results produced by Code1 and Code2 shows that the outputs of code1 and code2 are identical.