Information Hiding in SOAP Messages: A Steganographic Method for Web Services

Bachar Alrouh[1], Adel Almohammad[2], Gheorghita Ghinea[3]
Brunel University, West London, UK[1, 3]
University of Aleppo, Syria[2]
bachar.alrouh@brunel.ac.uk [1], adel.almohammad@gmail.com[2],
george.ghinea@brunel.ac.uk [3]

Abstract

Digital steganography is the art and science of hiding communications; a steganographic system thus embeds secret data in public cover media so as not to arouse an eavesdropper's suspicion. Hence, it is a kind of covert communication and information security. There are still very limited methods of steganography to be used with communication protocols, which represent unconventional but promising steganography mediums. In this paper, we discuss and analyze a number of steganographic studies in text, XML as well as SOAP messages. Then, we propose a novel steganography method to be used for SOAP messages within Web services environments. The method is based on rearranging the order of specific XML elements according to a secret message. This method has a high imperceptibility; it leaves almost no trail because of using the communication protocol as a cover medium, and since it keeps the structure and size of the SOAP message intact. The method is empirically validated using a feasible scenario so as to indicate its utility and value.

1. Introduction

Secure and secret communication methods are needed for transmitting messages over the Internet. Cryptography scrambles the message so that it cannot be understood. However, it makes the message suspicious enough to attract eavesdropper's attention. Additionally, due to increasing of computers capabilities and cipher texts availability, cryptographic techniques could be vulnerable. However, this vulnerability can be reduced significantly using steganography, which is a method of covert communication and information security.

Unlike encryption, steganography hides the even existence of secret information rather than hiding its meaning only. Thus, steganography is the art of hiding secret messages within other innocuous-looking cover files (i.e. images, audio, video, and text files) so that it cannot be observed. Consequently, steganography aims to hide the very existence of communication by embedding messages within other cover objects. As a result, the purpose of steganography is to keep others from thinking that a secret message even exists within the stego files.

Using only encryption for secret communication draws the attention of others. Therefore, steganography combined with cryptography raises the security level and would be the most secure method to go.

Steganography can be considered as a solution to exchange secret information and news between people around the world over the Internet without any fear of the message being detected. However, it has been claimed that the terrorists of the September 11th attacks used steganography to plan their attacks. Therefore, steganography is called "a terrorist's tool" [1], yet there is no evidence supporting such direction [2]. Additionally, businesses and governments have interests in breaking steganography (steganalysis) to detect secret messages for competitive advantages in the market (i.e. trade secrets or new product information) and to benefit national security [3].

Watermarking is a data hiding technique that protects digital documents, files, or images against removal of copyright information. Therefore, the goal of steganography is the secret messages while the goal of watermarking is the cover object itself [4]. Watermarking is the process of embedding a specific copyright mark into digital documents in the same way. Nevertheless, in order to detect any break of licensing agreement, a serial number is embedded in every copy of this digital document. This process is known as fingerprinting.

Text steganography refers to the process of hiding secret information in text files. For security and imperceptibility reasons, it is very important for stego texts not to show any detectable artifacts. Thus, readers should not notice or discover the modifications made in the stego text files. Generally, the redundant information in text files is very limited in comparison to that in images and audio files. Therefore, using text as cover files in steganography represents the most difficult way of information hiding [5].

Basically, there are three major methods to hide data in text files. The first method, open space method, manipulates white spaces in the text. Therefore, it exploits inter-sentence spacing, end-of-

line spaces, and inter-word spacing. The second method, syntactic method utilizes punctuation. However, the third method, semantic method, manipulates the words of the text themselves [5].

It is well known that Web represents the world's premier network and Extensible Mark-up Language (XML) represents the world's premier data representation format. Though, Web services require a data exchange in the form of XML documents, Simple Object Access Protocol (SOAP) exactly provides this kind of data transport. Therefore, SOAP supports a common data transfer protocol for effective communication over the Web [6]. Thus, XML is playing an increasingly important role in the exchange of a wide variety of data on the Internet. Therefore, XML documents are considered as a language of Web pages and digital contents. Moreover, they are used for the data exchange between organizations.

Web services provide a platform neutral and programming language independent technology that supports interoperable machine-to-machine interaction over a network. Moreover, clients and other systems interact with the Web service using a standardized XML messaging system, such as SOAP [7]. Therefore, structured and typed information can be exchanged between peers of distributed environment using SOAP messages.

In Web services, the interaction between service providers and requesters occurs typically via SOAP messages. Therefore, such messages offer a kind of steganography cover files. Hence, secret information can be embedded in SOAP messages and sent over the network to an intended destination.

Basically, a SOAP message is an XML document that contains text. Therefore, steganography methods used for text files and XML documents can theoretically be used for SOAP messages. Practically, some or all of these methods might be infeasible. Therefore, we are going to design and propose a new steganography method to embed secret information in SOAP messages. This method changes the order of XML elements according to the secret message to be embedded.

The rest of this paper is organized as follows. Section 2 reviews the related work on text and XML steganography. Section 3 discusses and explains the concept of information hiding within SOAP messages. Furthermore, our designed and proposed steganography method is illustrated in Section 4. An example scenario is illustrated in section 5. Finally, the conclusion is presented in Section 6.

2. Related Work

There is a relatively small number of text steganography studies in comparison to that of image video, and audio based steganography. This might be due to the lack of redundancy in text files [8].

Por and Delina [9] improved the open space method proposed by [5]. Therefore, they proposed a hybrid steganography method for text by combining both inter-word spacing and inter-paragraph spacing methods. Thus, whitespaces between words and paragraphs in right-justification of text are used for data hiding in order to increase the embedding capacity. However, the cover text was dynamically generated according to the size of the secret message.

Shirali-Shahreza [10] proposed a new steganography method for texts. This method is based on the different spelling of some words in English between UK and US. For example, “centre” has different terms in UK (centre) and US (center).

The model proposed in [11] defines a text steganography method based on substituting the words which have different terms in UK and US. For example, (Gas) has different terms in UK (Petrol) and US (Gas).

Liu et al. [12] proposed a text steganography method to be used in online chat. This method is based on an Internet meme named typoglecymia, which means that changing the order of word’s middle letters has a slight to no effect on the ability of skilled readers to understand the text (e.g. Guitar and Guiatr). Therefore, it used the redundancy found in the interior letters’ order. Since this letter randomization equals to the common error made by chatters due to high speed typewriting, it is likely to be used in online chats, where the text usually contains mistakes.

However, the previous studies provide text steganography method, which are not necessarily applicable in SOAP messages context due to the fact that SOAP messages are exchanged and monitored by computer systems rather than humans. Using misspelled or alternative words in SOAP messages would result in the SOAP parsers not being able to handle the SOAP messages received because they do not comply with the expected semantic.

To the best of our knowledge, there are only a couple of studies and examples of research regarding information hiding in XML files. Inoue et al. [8] proposed five steganography methods to be used with XML files. These steganography methods are summarized as follows:

1. The empty elements are represented according to the secret bit; either a start-tag immediately followed by an end-tag (<img></img>), or an empty-element tag (<img/>). This technique can embed one bit per empty element.

2. According to the secret bit, we can either add a white space before the close bracket (<tag >), or delete (normal with no added spaces) this white space (<tag>). This technique can embed one bit per tag.

3. Two elements may or may not be exchanged according to the secret bit. Thus, one bit per an exchange of two elements can be hidden.

4. The order of attributes in an element can be exchanged to hide the secret data. Thus, one bit per an exchange of the attributes order can be hidden.

5. Elements that contain each other can be used to hide secret data by exchanging inner-tags and outer-tags. In this method, one bit per an exchange can be hidden.

If an element has no content then empty-element tag can be used whether or not it is declared using the keyword EMPTY. However, the number of such elements in an XML document is limited and then the capacity of method (1) is limited too. Additionally, using two formats to represent empty elements in the same document will arouse the attention of observers. Also, the parser may use only one representation of empty elements rather than two, which invalidate this method. Names of XML elements can't contain spaces but there can be space before the closing character ">" (<tag >). However, this process will increase the size of the XML file and the hidden data may be destroyed due to parsing which may discard these added spaces (secret data). Additionally, tags are case sensitive and therefore the tag <tag> is different from the tag <tag >. In other words, the endtag's name has to exactly match the start-tag's name. Thus, the method (2) is practically infeasible since it uses a start-tag different from the end-tag (one tag may contain a white-space). The order in which attributes are included on an element is not considered relevant. For example, if an XML parser encounters a specific order of an element attributes, it doesn't necessarily have to give us the attributes in the same order. As a result, method (4) above is infeasible in terms of validity and applicability even though its capacity is very limited. However, a certain order of information can be maintained in an XML document if we put this information into elements, rather than attributes. As a result, method (3) above is a valid and possible solution for steganography. Nevertheless, hiding only one secret bit per an exchange of two elements represents a very small capacity. Finally, an XML document must have a top-level element and all the other elements are its children. Furthermore, one and only one root element must be included in each XML document even if this element has no content. However, each of these children elements may represent a parent element and therefore have some sub-elements. Thus, exchanging a parent element with a sub-element technically looks valid (method (5) above). However, it seems impractical since the semantics will not make sense and we will get a new and different parent element by such an exchange. Also, the steganographic capacity of this method is very limited.

Since XML documents are widely used for data exchange over different networks and exposed to different threats, XML security become a key concern of organizations. Thus, Memon et al. [13] considered XML steganography as a new method and solution for secure communication. Furthermore, they proposed and designed four XML based steganography methods for the purpose of securing the cover file (XML document) rather than for the purpose of secret communication. The main aspects of these methods are as follows:

1. Random characters are inserted inside all tags and their values. So, after the first character of the first tag one random character is inserted, after the second character of the first tag two random characters are inserted and so on. Thus, it mixes up the actual XML data with random fake characters and therefore increases the size of the stego XML file significantly.

2. XML tags are shuffled (sequentially) in such a way the position of the 1st tag and its value are swapped with that of the last tag and its value. The same process happens with the second and the second last tags, and so on. The large XML file is, the better this technique work.

3. Similar to the previous method, but the correct order of shuffled tags is identified in the attribute value of the root element. Thus, the first tag is determined by the first character of attribute value while the second character is randomly generated. Also, this method works better with large XML files.

4. The sequence of characters in all tags and values are reversed. Thus, the order of tags' characters is reversed by moving the last character to become the first one while the second last one becomes the second character and so on. As a result, the XML file will look like an encrypted file since the characters are scrambled in an unreadable form.

Then, they suggested combining all these methods together in one hybrid method to provide better XML security. In conclusion, all these four methods aim to safeguard the stego XML document against actual XML content detection rather than against hidden information detection. Additionally, their goal is the XML content not the hidden data itself. Therefore, the goal of these methods is totally different from our steganography goal which is undetectable and covert communication. Nevertheless, the first and fourth methods are definitely infeasible for steganography since the stego XML arouses the suspicion of everyone (look like encrypted). The second method may hide a few bits only, while in the third method, the secret key is included in the stego file, which is more than enough to extract the hidden message.

SOAP parsers have been developed and they only process XML that conforms to the SOAP schema and associated structural rules. Zhang et al. [7] proposed a steganography method depending on the text characteristics of SOAP technology in order to hide information in SOAP messages. Therefore, the physical properties of SOAP keywords and namespaces (self-defined) are used as cover

message. A character string is initialized by converting every letter in these keywords and namespaces into lowercase. Coordinating every secret bit with every letter of the character string, a specific letter is converted into capital letter only when the secret bit is "1". However, the amount of SOAP keywords is limited for short SOAP message. Furthermore, the stego SOAP looks suspicious since some characters of this message are in lowercase shape while others are in uppercase shape. Therefore, the overall shape of the stego SOAP may attract attention. Additionally, this method does not comply with the case-sensitivity nature of XML documents.

3. Information Hiding in SOAP Messages

The SOAP protocol is designed to enable the exchange of structured information (i.e. SOAP messages) over a variety of underlying protocols in decentralized and distributed environments. This lightweight protocol uses XML technologies to define a messaging framework that is independent of any specific programming languages or implementation semantics [6].

A SOAP message is an XML document, which consists mainly of "envelope, header, body and fault elements, as shown in (Figure 1). The "*Envelope"* is the root element that defines the XML document as a SOAP message. Also, it indicates the start and the end of the message. Application specific information (like security, reliability, etc) is usually defined within the optional "*Header"* element. Additionally, headers may contain commands to SOAP processors either to understand these headers or to reject the SOAP message. However, the actual data is defined within the required "*Body"* element. Thus, mandatory information that must be delivered to the intended recipient should be included within the body part of SOAP message. The optional "*Fault"* element is used to identify error messages. If an error occurs during SOAP processing, a SOAP fault element will be emerge in the body of the message. Then, the sender of the SOAP message will get the fault response returned.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Header>…</S:Header>
   <S:Body>…
            <S:Fault>… </S: Fault >
   </S:Body>
</S:Envelope>
```

Figure 1. SOAP Message Construct

When two parties communicate through SOAP messages, the actual data (i.e. fields and properties of objects or parameters and return values of methods) in the sender endpoint are converted (serialized) into an XML stream that conforms to the SOAP specifications. This serialized XML document is the SOAP message that needs to be de-serialized at the receiver endpoint to reconstruct the actual data. Figure 2 illustrates an example Java class *Book* and its XML serialized class instance.

```
public class BookOrder{
   private String isbn;
   private String author;
   private String bookName;
   private int numOfPages;
   private String publisher;
   private int year;
   private double price;
  public getters and setters
}
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:BookOrder
xmlns:ns2="http://service.bookorder.com/">
         <book>
            <isbn>1-11-111111-1</isbn>
            <author>Author_1</author>
            <bookName>Book_1</ bookName >
            <numOfPages>372</numOfPages>
            <publisher>Publisher_1</publisher>
            < year >2009</year>
            <price>29.99</price>
         </book>
      </ns2:BookOrder>
   </S:Body>
</S:Envelope>
```

Figure 2. Example Java Class and Its XML Serialized Instance

An endpoint application normally employs a SOAP package to perform the serialization and de-serialization processes, as Web applications and clients care mainly about the actual data transmitted and not the structure of the SOAP message. Hence, secret information can be smuggled into SOAP messages, which provide a perfect cover if the hidden secret message does not damage the SOAP messages or spoil the actual data.

The main concern of hiding secret information within SOAP message is how we can hide this information and not to be detected. Basically, end users care about the actual data transmitted but they do not care about other issues like SOAP namespace, keywords, or the order of elements' attributes. However, the transmitted message must generate no errors and therefore not to discard the message.

Hiding secret information in a SOAP message means that the mule that is used to convey the secret message is the communication protocol that governs the actual data path over a network, instead of using the actual data itself as a cover. This idea can overcome many of the limitations that faced the

conventional steganography techniques. Traditional techniques hide secret messages inside digital files, which impose the threat of detecting the secret as these files are usually saved. Alternatively, a SOAP message leaves almost no trail as they are normally deleted after receiving the message and de-serializing the actual data. In addition, a secret piece of information can be divided into multiple smaller messages and transmitted over several SOAP messages to overcome the size limitation as well.

This paper provides a novel steganography method that manipulates the SOAP protocol by rearranging the order of the contents and attributes of specific elements in a SOAP message, where every permutation represents a specific status according to a secret key shared between the sender and the receiver. For example, there are 7 sub-elements within the element *book* in Figure 2. These sub-elements are arranged in a particular order (*isbn, author, bookName, numOfPages, publisher, year, price*). This order does not have any importance for the endpoint application, however. If the order of these sub-elements is rearranged, the message will still have the same meaning for the endpoint.

For a set of *n* sub-elements, there are a maximum of *n!* (factorial of *n*) permutations . This means that n! different sequences of order can be presented.

4. Steganography Framework for SOAP Messages

Considering the previous concept, we have designed and implemented a data hiding method that monitors a SOAP message just after its serialization in the sender endpoint and before it is sent, analyzes its elements and embeds a secret message accordingly. Figure 3 illustrates the general model of data hiding in SOAP messages.

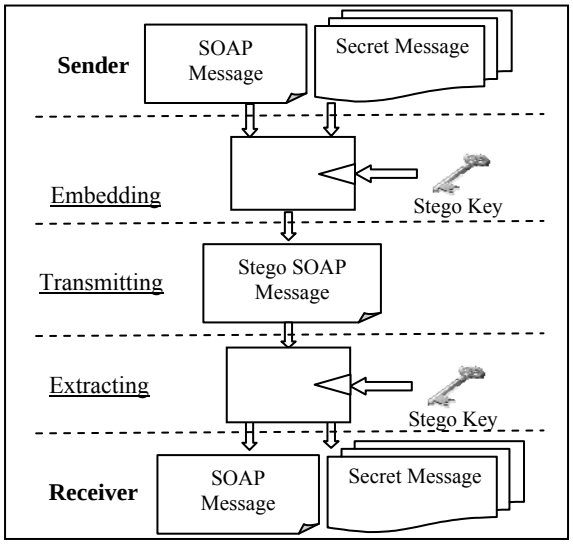

Figure 3. SOAP Steganography Model

When the stego SOAP message arrives at the receiver endpoint, the secret message is extracted using a stego key that is shared between the sender and receiver.

4.1 Embedding Procedures

In our proposed method, the procedure of hiding a secret message within SOAP consists of the following six steps.

1. Capturing the SOAP message after its serialization.
2. Analyzing its contents to identify all the elements with contents that can be rearranged to determine if the SOAP message is suitable for embedding (i.e. has elements with contents that can be rearranged).
3. Calculating the number of elements that can be used to hide data (N).
4. Permuting every set of sub-elements to reflect a status of a symbol from the secret message.
5. If all the symbols of the secret message can be hidden in one SOAP message (the number of available sets N is greater than the length of the secret message M), then the sub-elements of the set M+1 will be rearranged to indicate the end of secret message.
6. Otherwise, if M>N, only a part of the secret message is sent in this SOAP message and the last set of sub-elements is rearranged to indicate that more hidden data are to arrive within the next received SOAP message.

Figure 4 illustrate the algorithm used for secret message embedding.

4.2 Extracting Procedure

The receiver, on the other hand, extracts hidden data by analyzing the contents of each eligible element using the secret key to reveal the hidden symbol, as described in the following section and illustrated in the Extracting Algorithm (Figure 5):

1. Capturing the SOAP message and checking its validity and capability to be a stego SOAP message.
2. Calculating the number of elements that might be used for data hiding (N).
3. Extracting the hidden symbols by analyzing the sub-elements order of each element in the stego SOAP message.
4. Stop the process either if the extracted symbol indicates that the message is not a stego SOAP or if the extracted symbol means “End of Message”.
5. If the extracted symbol means “To Continue”, new SOAP message to be captured and analyzed as in 1.
6. Otherwise, the next symbol will be extracted and so on until we get the entire secret message embedded.

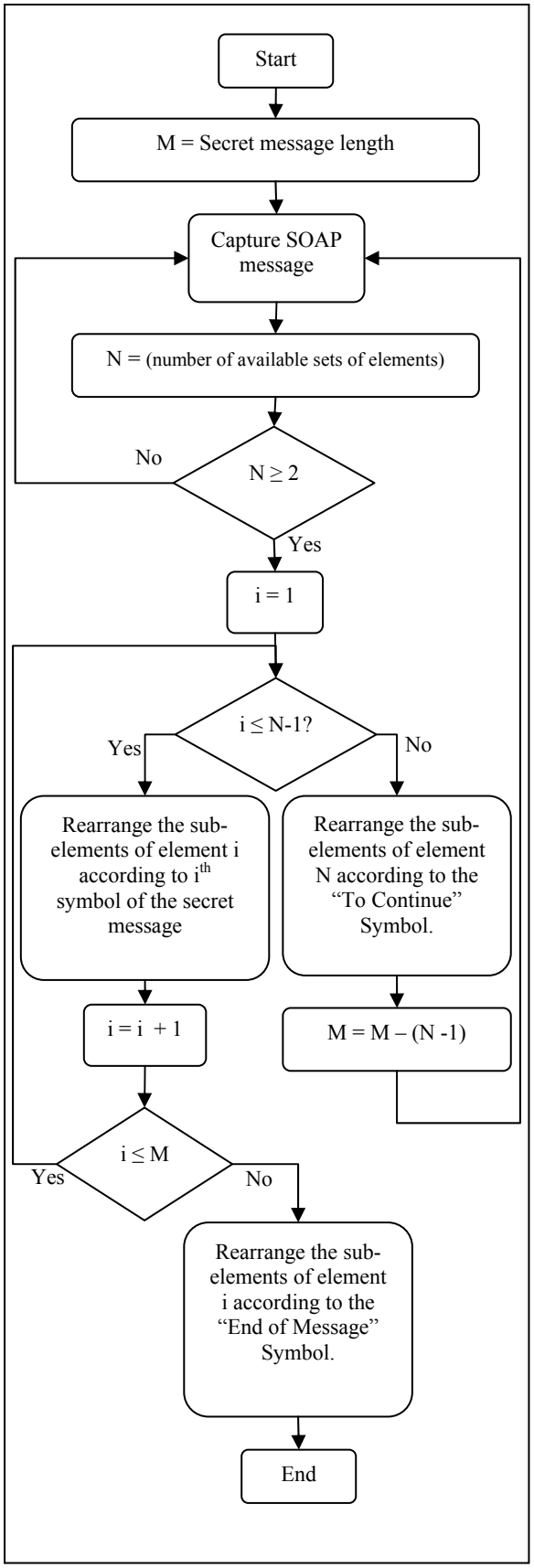

Figure 4. Secret Message Embedding

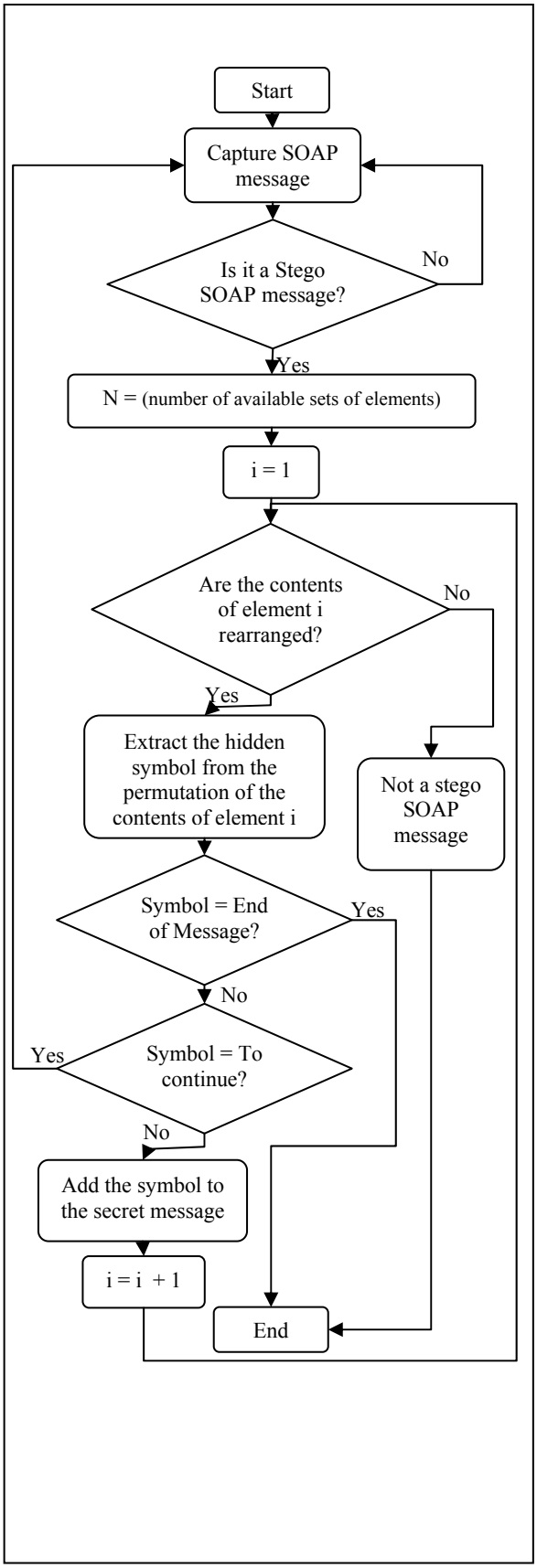

Figure 5. Secret Message Extracting

5. Example Scenario

Our proposed method for SOAP message-based steganography is empirically tested and validated. Thus, we demonstrate the data embedding and extracting algorithms using an example yet realistic web service scenario (Book Order): In this scenario, we assume that the person who wants to send secret data is the "Book Buyer" while the intended recipient of secret message is the "Book Seller". However, the opposite scenario is true since the "Book Seller" can send a secret message to the "Book Buyer" using the same procedure. The example scenario is:

Step 1: The Book Buyer (Service Requester) selects the books to be ordered from the Book Seller website (service Provider).

Step2: The Book Order will be formatted as XML document and then an XML-based SOAP message will be generated in order to be sent to the Service Provider.

Step 3: An application is used at the sender (Book Buyer) endpoint in order to capture each SOAP message before it has been sent (prevents the sending process of SOAP).

Step 4: The "Embedding Procedure" of our SOAP steganography method is applied on each captured SOAP message.

Step 5: The output of the "Embedding procedure" (a stego SOAP message) will be sent to the Book Seller.

Step 6: The Book Seller receives the SOAP message (a stego SOAP) and a similar application to that used at the Book Buyer endpoint will be used at the Book Seller endpoint to capture each received SOAP message.

Step 7: The "Extracting Procedure" of our SOAP steganography method is applied on each captured SOAP message in order to extract the secret message from the stego SOAP messages.

As illustrated in Figures 6, 7 and 8, a book buyer is sending two messages to the book seller. The first SOAP message contains an order for 4 books, while the second is an order for 3 books. A secret message "*Hello"* is smuggled by shuffling the sub-elements of each "*book"* element in these SOAP messages. The first message contains only part of the hidden message "*Hel"* and "*to continue"* symbol, while the second message contains the rest of the message *"lo"* and the "*end of message"* symbol. Because each element has 5 sub-elements, 5! (120) different cases can be represented. That covers all the alphabetical characters (in small and capital caps), numbers and most of the printing characters. For the purpose of demonstration, we used a shifted version of the ASCII table as a secret key for data hiding. More complex secret keys can be used in real implementations.

For this experiment, NetBeans IDE 6.9 is used to develop the web service (book seller service) and the client (book buyer application). The web service is built as a web application and deployed on a Glass Fish 2.2 application server. All the SOAP messages are intercepted in the sender endpoint just after being serialized into XML messages and before the SOAP messages are sent to the receiver endpoint. Similarly, all the coming SOAP messages are intercepted before they are de-serialized. The SOAP messages are also monitored and recorded using soapUI in the standard HTTP proxy mode.

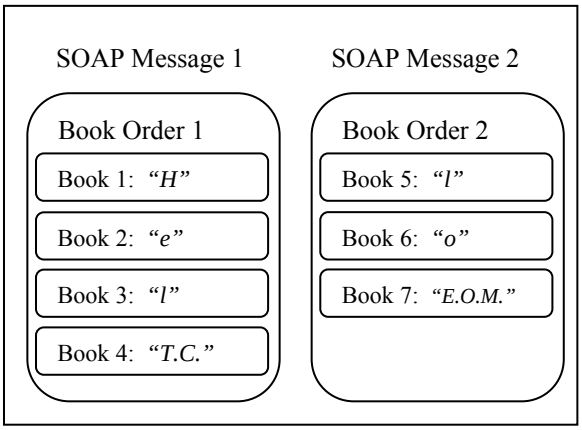

Figure 6. Example Secret Message Hidden in 2 SOAP Messages

6. Conclusion

In this paper, we have provided a communication protocol-based steganography method that manipulates the SOAP protocol. This method monitors a SOAP message just after its serialization in the sender endpoint and before it is sent. It analyzes the SOAP elements and embeds a secret message accordingly by rearranging the order of the contents and attributes of specific elements in a SOAP message, where every permutation represents a specific symbol according to a secret key shared between the sender and the receiver. As a result, the provided method has a high resistance against detection since it uses the communication protocol as a cover medium rather than the traditional digital files. Furthermore, the stego SOAP message has the same size of the original message. The method is tested and validated using a feasible scenario so as to demonstrate its utility and applicability.

Security is an ongoing process and as soon as developers fix one set of problems crackers will find yet another way to break these systems. Essentially, the applications must be flexible in order to add new security features as needed.

Furthermore, anyone on the Internet can intercept the data transmitted between different sites. Thus, distributed applications require higher security levels than internal applications.

Book Order (Step 1):
Order 1:
1-Book 1: isbn=1-11-111111-1, author=Author_1, name=Book_1, pages=111, publisher=Publisher_1
2-Book 2: isbn=2-22-222222-2, author=Author_2, name=Book_2, pages=222, publisher=Publisher_2
3-Book 3: isbn=3-33-333333-3, author=Author_3, name=Book_3, pages=333, publisher=Publisher_3
4-Book 4: isbn=4-44-444444-4, author=Author_4, name=Book_4, pages =444, publisher=Publisher_4

Cover SOAP (Stes 2+3):

SOAP Message 1:

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
 <S:Body>
  <ns2:bookOrder xmlns:ns2="http://service.testproject/">
   <book>
    <isbn>1-11-111111-1</isbn>
    <author>Author_1</author>
    <name>Book_1</name>
    <pages>111</pages>
    <publisher>Publisher_1</publisher>
   </book>
   <book>
    <isbn>2-22-222222-2</isbn>
    <author>Author_2</author>
    <name>Book_2</name>
    <pages>222</pages>
    <publisher>Publisher_2</publisher>
   </book>
   <book>
    <isbn>3-33-333333-3</isbn>
    <author>Author_3</author>
    <name>Book_3</name>
    <pages>333</pages>
    <publisher>Publisher_3</publisher>
   </book>
   <book>
    <isbn>4-44-444444-4</isbn>
    <author>Author_4</author>
    <name>Book_4</name>
    <pages>444</pages>
    <publisher>Publisher_4</publisher>
   </book>
  </ns2:bookOrder>
 </S:Body>
</S:Envelope>
```

Stego SOAP (Steps 4+5+6+7):

SOAP Message 1:

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
 <S:Body>
  <ns2:bookOrder xmlns:ns2="http://service.testproject/">
   <book>
    <author>Author_1</author>
    <pages>111</pages>
    <publisher>Publisher_1</publisher>
    <name>Book_1</name>
    <isbn>1-11-111111-1</isbn>
   </book>
   <book>
    <name>Book_2</name>
    <publisher>Publisher_2</publisher>
    <pages>222</pages>
    <isbn>2-22-222222-2</isbn>
    <author>Author_2</author>
   </book>
   <book>
    <pages>333</pages>
    <isbn>3-33-333333-3</isbn>
    <publisher>Publisher_3</publisher>
    <name>Book_3</name>
    <author>Author_3</author>
   </book>
   <book>
    <publisher>Publisher_4</publisher>
    <isbn>4-44-444444-4</isbn>
    <author>Author_4</author>
    <name>Book_4</name>
    <pages>444</pages>
   </book>
  </ns2:bookOrder>
 </S:Body>
</S:Envelope>
```

Secret Message:
"Hel"
Stego Key:
NO EMBEDDING = isbn, author, name, pages, publisher.
"H" Character = author, pages, publisher, name, isbn
"e" Character = name, publisher, pages, isbn, author
"l" Character = pages, isbn, publisher, name, author
"To Continue" = publisher, isbn, author, name, pages

Figure 7. Hiding the First Part of the Secret Message in The First SOAP Message

Book Order (Step 1):
Order 2:
1-Book 5: isbn=5-55-555555-5, author=Author_5, name=Book_5, pages =555, publisher=Publisher_5
2-Book 6: isbn=6-66-666666-6, author=Author_6, name=Book_6, pages =666, publisher=Publisher_6
3-Book 7: isbn=7-77-777777-7, author=Author_7, name=Book_7, pages =777, publisher=Publisher_7

Cover SOAP (Stes 2+3):

SOAP Message 2:

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
<ns2:bookOrder xmlns:ns2="http://service.testproject/">
        <book>
           <isbn>5-55-555555-5</isbn>
           <author>Author_5</author>
           <name>Book_5</name>
           <pages>555</pages>
           <publisher>Publisger_5</publisher>
        </book>
        <book>
           <isbn>6-66-666666-6</isbn>
           <author>Author_6</author>
           <name>Book_6</name>
           <pages>666</pages>
           <publisher>Publisger_6</publisher>
        </book>
        <book>
           <isbn>7-77-777777-7</isbn>
           <author>Author_7</author>
           <name>Book_7</name >
           <pages>777</pages>
           <publisher>Publisher_7</publisher>
        </book>
     </ns2:bookOrder>
   </S:Body>
</S:Envelope>
```

Stego SOAP (Steps 4+5+6+7):

SOAP Message 2:

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
<ns2:bookOrder xmlns:ns2="http://service.testproject/">
        <book>
           <pages>555</pages>
           <isbn>5-55-555555-5</isbn>
           <publisher>Publisger_5</publisher>
           <name>Book_5</name>
           <author>Author_5</author>
        </book>
        <book>
           <pages>666</pages>
           <author>Author_6</author>
           <name>Book_6</name>
           <isbn>6-66-666666-6</isbn>
           <publisher>Publisger_6</publisher>
        </book>
        <book>
           <publisher>Publisher_7</publisher>
           <pages>777</pages>
           <name>Book_7</name >
           <author>Author_7</author>
           <isbn>7-77-777777-7</isbn>
        </book>
     </ns2:bookOrder>
   </S:Body>
</S:Envelope>
```

Secret Message:
"lo"
Stego Key:
NO EMBEDDING = isbn, author, name, pages, publisher.
"l" Character = pages, isbn, publisher, name, author
"o" Character = pages, author, name, isbn, publisher
"End of Message" = publisher, pages, anme, author, isbn

Figure 8. Hiding the Second Part of the Secret Message in the Following SOAP Message

Encryption can be used to preserve data security but the technologies required for encryption cause problems with firewalls and they don't work very well on the Internet. Encryption has another problem; if both communication parties don't have the same platform then the receiver can't decrypt the sender's message. Thus, even a common encryption scheme usually can only work on a limited number of platforms [14]. As a result, our SOAP based steganography method could be a reasonable solution for transmitted data security. It can be used as a secret communication channel over different kinds of networks regardless of the applications used at the distributed endpoints. As a kind of communication security, the process of surely knowing the identity of the other communicating party (on the other end of a channel) is known as Authentication. Additionally, associated HTTP Authentication Framework with HTTP 1.1 provides better authentication means between communicating parties. Thus, the HTTP Authentication Framework secures only the authentication portion of the communication. Furthermore, Secure/Multipurpose Internet Mail Extensions (S/MIME) and Secure Socket Layer (SSL) use digital certificates to provide security which relies on the use of public key cryptography. Usually, using static keys provides the crackers more chance to break the system than using dynamic keys [14].

As a result, we can use the proposed SOAP steganography method to convey information of authentication which necessary to authenticate the communicating parties. Additionally, encryption keys can be embedded and transmitted in order to get dynamic keys instead of static keys, and therefore add another layer of system security.

Basically, encryption algorithms represent a conventional solution of information security but the encrypted data is still there and everyone can observe it over the network. Thus, our SOAP steganography algorithm provides a way of secret communications over the Internet. It can overcome the limitations and challenges of encryption as well as it can be used with encryption to provide a double layer of security.

In conclusion, the proposed SOAP steganography method can be used for a variety of applications such as; authentication, proof of identity, watermarking, digital signature and message hash.

6. References

[1] K. Bailey, K. Curran, J. Condell, "Evaluation of Pixel-Based Steganography and Stegodetecction Methods", *The Imaging Science Journal*, Vol. 52(3), pp. 131-150, 2004.

[2] S. Engle, "Current State of Steganography: Uses, Limits, & Implications", 2003, University of California, Davis Website. Available from: http://wwwcsif.cs.ucdavis.edu/~engle/stego.pdf. Access date: 1 May 2010.

[3] D. Artz, "Digital Steganography: Hiding Data within Data", *IEEE Internet Computing*, vol. 5(3), pp. 75-80, 2001.

[4] S. Venkatraman, A. Abraham, and M. Paprzycki, "Significance of Steganography on Data Security", In *Proceedings of the International Conference on Information Technology: Coding and Computing*, 2004.

[5] W. Bender, D. Gruhl, N. Morimoto, and A. Lu, "Techniques for Data Hiding", *IBM Systems Journal*, vol. 35(3-4), pp. 313-336, 1996.

[6] E. Newcomer, *Understanding Web Services: XML, WSDL, SOAP and UDDI*, Addison Wesley, 2002.

[7] X. Zhang, H. Wang, and J. Sun, "An Information Hiding Method based on SOAP", In *Proceedings of the 3rd International Conference on International Information Hiding and Multimedia Signal Processing (IIH-MSP 2007)*, Kaohsiung, Taiwan, 2007.

[8] S. Inoue, K. Makino, I. Murase, O. Takizawa, T. Matsumoto, and H. Nakagawa, "A Proposal on Information Hiding Methods using XML", In *Proceedings of the 1st Workshop of NLP and XML*, Nov, 2001.

[9] L. Y. Por and B. Delina, "Information Hiding: A New Approach in Text Steganography", In *Proceedings of the 7th International Conference on Applied Computer & Applied Computational Science (ACACOS'08)*, Hangzhou, China, 2008.

[10] M. Shirali-Shahreza, "Text Steganography by Changing Words Spelling", In *Proceedings of the 10th International Conference on Advanced Communication Technology (ICACT 2008)*, Phoenix Park, Korea, 2008.

[11] M. H. Shirali-Shahreza and M. Shirali-Shahreza, "A New Synonym Text Steganography", In *Proceedings of the 4th International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP 2008)*, Harbin, China, 2008.

[12] M. Liu, Y. Guo, and L. Zhou, "Text Steganography Based on Online Chat", In *Proceedings of the 5th International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP 2009)*, 12-14 Sep, 2009, Kyoto, Japan, 2009.

[13] A. G. Memon, S. Khawaja, and A. Shah, "Steganography: A New Horizon for Safe Communication Through XML", *Journal of Theoretical Information Technology*, vol. 4(3), pp. 187-202, 2008.

[14] J. P. Mueller, *Special Edition Using SOAP* , QUE, USA, 2001