Global Information Assurance Certification Paper

How to provide security for web service (SOAP)

By Wen Xue
Submit date: June26, 2004

SANS GIAC GSEC Practical Assignment Version 1.4B
Option 1

Table of contents

Abstract

This article starts with an overview of web service (SOAP) and its security challenges. One way to address those challenges is on the network level, i.e. to run web service over SSL or IPSec. Another approach is to solve the issue on the application level. One such effort is Web Service Security: SOAP Message Security (WSS:SMS) standard, which is a security add-on to the SOAP standard.

This paper focuses on WSS:SMS. With an overview and a detailed example, this paper will give the reader a good idea of how WSS:SMS resolves the security challenges, as well as its capabilities and its processing rules. The paper also discusses each solution's advantages and disadvantages. Finally, a recommendation is given on how to combine the technologies in order to provide the best result.

1. Web Service and Security challenge

1.1 What is web service

Web service can mean a lot of different things to different people. In this paper, a definition from an article found on a Microsoft website is used:

> We use "Web service" to describe application components whose functionality and interfaces are exposed to potential users through the application of existing and emerging Web technology standards including XML, SOAP, WSDL, and HTTP. [1]

By this definition, web browsing is not web service because it requires people's direct interaction. Any services using Remote Method Invocation (RMI), Enterprise Java Bean (EJB) or Common Object Request Broker Architecture (CORBA) are not web service. On the other hand, if two servers from two businesses are exchanging information automatically using SOAP, that is web service.

1.2 Web Service technology

Web Service is a suite of three major technologies, SOAP, WSDL and HTTP. A high level overview of each is given in this section. Another good overview can be found in [2].

1.2.1 SOAP:

Simple Object Access Protocol (SOAP) is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. [3]
The SOAP standard defines three things:

- How messages should be constructed.
- A set of encoding rules that describe data types for an application to process data.
- A convention for representing remote procedure calls and responses.

A SOAP message is an XML document that is enclosed in the SOAP <Envelope>. Inside the <Envelope> there is an optional SOAP <Header>, and a mandatory SOAP <Body>.

- The <Header> is the place to add new elements to extend SOAP features. By describing new functions in the <Header>, the two communicating parties can understand each other in the distributed environment without any pre-agreement.
- The <Body> contains the actual request or response message.

Below is a simple SOAP message sample copied from the SOAP standard but slightly modified for easy understanding. The SOAP body contains an alert message. The header part contains priority and expiration information for the recipient to decide how it should be processed.

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
        <n:priority>1</n:priority>
       <n:expires>2001-06-22T14:00:00-05:00</n:expires>
</env:Header>
<env:Body>
        <m:alert xmlns:m="http://example.org/alert">
              <m:msg>Pick up Mary at school at 2pm</m:msg>
       </m:alert>
</env:Body>
</env:Envelope>
```
[2]

1.2.2 WSDL

Web Service Description Language (WSDL)[4] is used as meta-data for SOAP. It defines what a web service can do, where it resides, how it should be invoked, and the structure and format of the SOAP message for a particular web service.

1.2.3 HTTP

Hypertext Transfer Protocol (HTTP) is the network protocol used to transfer SOAP requests and responses between client and server. SOAP can potentially be carried by a variety of other protocols, such as SMTP. However, HTTP is the only binding transport protocol defined in the SOAP standard.

1.3 Web service security challenges

Web service faces all the security challenges that other Internet applications are facing. Because of business involvement and the financial liability, some risks are more significant than others. This paper only discusses network transport and transaction related issues.

- Confidentiality: Unintended parties should not be able to understand the message. Special care must be taken especially because of SOAP's ASCII format. Otherwise even an amateur hacker can sniff the message.
- Integrity: Web service works in a distributed environment. The message may be received and forwarded on by an intermediary, who may not be completely trusted. No one should be able to modify the message during the transfer without being detected
- Authentication: We want to verify that the response is really originated from the claimed sender.
- No-repudiation: The sender must not be able to deny that he ever sent the message. This is one step above authentication. For example, if authentication is based on a shared secret, the sender still has a chance to deny because the receiver also has the ability to create the message using the shared secret. The concept of no-repudiation is very important in the business world.

Authorization will not be discussed because it's not tightly coupled with web service, and most businesses will choose their own implementation.

2. Solutions on network level

These security challenges can be partially solved by using existing network protocols, such as SSL and IPSec to carry HTTP and SOAP. The protocol stacks are shown in figure 1.

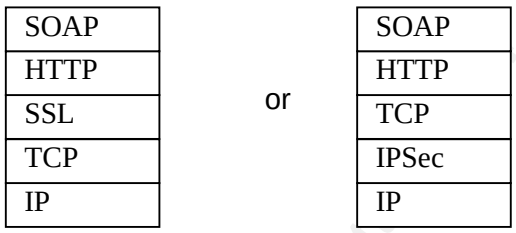

Figure 1: Network protocol solution stack

Before we talk about the solutions, let's briefly review the cryptosystem.

2.1 Crypto system

Based on the SANS Security Essential with CISSP CBK book [5], there are three types of crypto algorithms.

- **Secret key cryptosystem**: A single key is used for both encryption and decryption. It is also called symmetric key.
- **Public key cryptosystem**: Each key holder has a public key and a private key. As its name implies, everyone can know the public key, but only the key holder knows his own private key. There are three primary usages for the public key cryptosystem.

    - Encryption: If a message is encrypted by A's public key, assuming only A knows A's private key, then A and only A can decrypt the message with A's private key.
    - Authentication and non-repudiation: If a signature can be verified by A's public key upon a particular message, then it is A and it must be A that signed the message using his private key
    - Key exchange: By a series of mathematical calculations, A and B can agree on a new key without ever transferring the key.
- **Hashing**: Hashing is a one-way mathematical calculation. From the same message the hashing of the message always yields the same hashed value. Any change in the message will yield a different result. The hashing calculation is fast. But for the other direction, from the hashed value, it is extremely difficult to derive/guess the original message. Hashing is usually used to check message integrity. Sometimes the hashed value is called message digest.

2.2 SSL

The Secure Socket Layer (SSL) protocol runs above TCP/IP and under application level protocols such as HTTP or IMAP. SSL-enabled servers and clients can authenticate each other, and establish an encrypted connection to provide confidentiality and integrity. [6]

The SSL protocol uses a public-key cryptosystem to authenticate each other and to negotiate the symmetric key for later encryption. This is because symmetric key encryption is much faster than public-key (asymmetric) encryption, but public-key encryption provides better authentication. The major steps of SSL can be summarized as follows:

1. Every SSL session begins with a handshake. The client and the server send information to each other, such as SSL version, cipher setting, random data, etc. Based on the information, the two sides agree upon a common setting for later communication. Furthermore, the client and server should exchange their X.509 digital certificates.
2. The client and the server authenticate each other (or just authenticate in one direction) by validating the other's X.509 certificate.
3. By utilizing the public-key based key exchange algorithms and the results from the handshake, the client and the server can negotiate a symmetric session key. The key can be used to encrypt and to decrypt messages during later communication. Because the key is never transferred across the network, only the client and the server know this session key and can understand the data.
4. Now the client and the server can start to transfer data with confidence that they are sending and receiving data to and from the right party. Nobody else can understand the data. Nobody can tamper with the data.

2.3 IPSec

IP Security (IPSec) is actually a suite of protocols developed by the IETF. A good quick overview can be found in [7]. The three most important protocols are:

- **Authentication Header (AH):** AH is mainly used for IP source authentication. Every IP header contains a source IP address. Many firewalls use the source IP as a criterion to admit or reject a packet. An attacker may modify his source IP address to an allowed IP address in order to gain access. To prevent this, the sender is required to calculate a hash value based on every field in the IP header (except for the field that may change during the transmission, such as TTL). The hash value is put into the AH header and inserted between IP header and IP payload. The hash calculation uses a key that is negotiated at the beginning of the communications. Only the parties who passed the IKE authentication phase can negotiate the key and calculate the correct hash value. Therefore any tampering with the IP header can be detected.
- **Encapsulated Security Payload (ESP):** ESP offers authentication, confidentiality and data integrity. The idea is similar to AH: Use negotiated key to encrypt date, to calculate and verify the hash value. ESP doesn't specify the encryption algorithm, though DES3 is most commonly used. Depending on the user's security requirements, this mechanism may be used to encrypt either a transport-layer segment (e.g., TCP, UDP, ICMP, IGMP) or an entire IP datagram.[8] Because of this, AH is not often used anymore because ESP does everything AH does and more.
- **Internet Key Exchange (IKE)** is the protocol used to negotiate the session details that can be used by AH and ESP. In IKE version 1, the negotiation includes two steps. This first step is to build an authenticated and secure connection to protect further conversation. The second step is to negotiate all the details for AH and ESP to happen. IKE version 2 use new procedures. However, it is not widely implemented yet.

2.4 Deficiencies of SSL and IPSec for web service

From the above discussion we can see that SSL and IPSec can provide good authentication, confidentiality and message integrity. However, for web service, they have some deficiencies, as discussed in this section.

2.4.1 SSL and IPSec only provide point-to-point security, not end-to-end security.

In this paper, the point-to-point communication is defined in IP layer, particularly the communication from source IP to destination IP. It is not considered end-to-end even there are gateways, routes and firewalls in the communicating path.
On the other hand, when the communication is beyond the IP layer, it is end-to-end. For example, a communication between a web client and a web server via a proxy is end-to-end, because the proxy server will unpack the IP packets and exam packets in HTTP level. Or if packets arrives at the destination and printed out and put on the manger's desk, that is end-to-end.

Based on the article found on a Microsoft website:[1]

When data is received and forwarded on by an intermediary beyond the transport layer, both integrity of the data and any associated security information that is transferred with the *data may be lost. This forces any upstream message processors to rely on the security evaluations made by previous intermediaries and to completely trust their handling of the content of messages.*

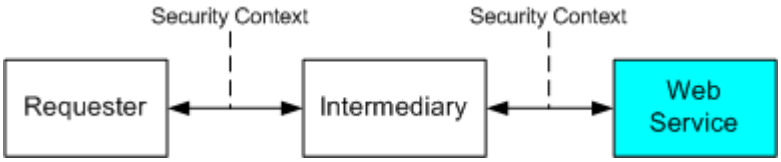

Figure 2 Point-to-point security

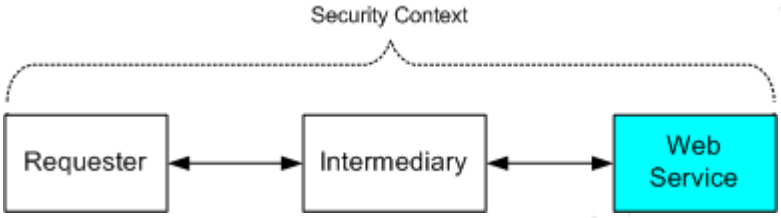

Figure 3: End-to-end security

Above two diagrams are also copied from [1].
Figure 2 shows that IPSec and SSL can provide security for the point-to-point communication. Figure 3 shows that IPSec and SSL may provide security for the end-to-end communication only when they can completely trust all the intermediary points.

2.4.2 SSL and IPSec is not no-repudiation friendly.

For SSL and IPSec, the public key is only used during the initial negotiation. The message payload is encrypted with a symmetric key, which is known to both the sender and the receiver, so no-repudiation can not be provided by just checking an individual IP packet. Furthermore, one SOAP message might be carried by multiple packets. It will be quite hard to piece together all the packets to prove the non-repudiation.

3. Solution at the Application level: Web Service Security-SOAP Message Security (WSS:SMS)

Web Service Security: SOAP Message Security (Shortened as WSS:SMS in this paper) is a specification developed by the OASIS group (http://www.oasis-open.org). This specification proposes a standard set of SOAP extensions that can be used when building secure Web services to implement message content integrity and confidentiality.

Based on the WSS:SMS standard[9], WSS:SMS itself doesn't provide new technology for security. Instead, WSS:SMS relies on existing mechanisms, such as PKI, Kerberos, etc. WSS:SMS provides rules on how to use existing technology to process the original SOAP message. WSS:SMS also provides syntax on how to embed the security

information in the message, such as signature, encryption result, etc. and how to describe the process. If the sender follows the rules to process the original message and put enough information on what he did, then without any pre-agreement, the receiver knows how to proceed to decrypt or to verify the message.

3.1 WSS:SMS defines three main functions

3.1.1 How to pass security token.

The message sender may want to pass some credential, such as user name and password, for authentication. He may also want to pass some secret key or public-key so the receiver can decrypt the message and verify the signature. WSS:SMS uses the term "security token" to represent all security-related information that can be applied to the message.

- A security token itself can be signed. That means the token issuer (not the token owner) should sign the token with his private key and attach the signature to the security token. Then everyone who knows the issuer's public-key can verify that the token is indeed issued by that issuer. X.509 digital certificate and Kerberos ticket are two examples of signed security tokens.
- A security token can also be unsigned including things such as username and password. However, username/password is weak by nature, and a password can not be used to sign or to encrypt messages. So in general, username and password are not recommended for web service without special care. In this article we will only discuss signed security tokens.

WSS:SMS provides the methods and syntax to carry security token within a SOAP message.

3.1.2 How to provide authentication and message integrity

WSS:SMS provides the format and syntax for a sender to specify which part of the message he wants to sign, which security token and what signature algorithm he will use, etc. So when the receiver sees the message, he will know how to verify the message by just looking into the message.

3.1.3 How to encrypt the message

The WSS:SMS specification allows encryption of any combination of body blocks, header blocks, any of these sub-structures, and attachments. The encryption is done by using either a symmetric key that is shared by the sender and the receiver, or by using a new symmetric key carried in the message, which itself is encrypted with the receiver's public key.

These three mechanisms can be used independently, e.g., only to pass username and password for authentication. Or in a combined manner, e.g., to pass signed security tokens, and use the tokens to sign and to encrypt the messages.

3.2 Example walk-through

As we discussed before, new elements can be added to a SOAP <Header> block in order to provide new features. <Security> and many of its sub elements are the elements introduced by WSS:SMS to add security features. With special rules and syntax, these elements can be used to pass security tokens, to describe algorithms, to attach a signature, or to encrypt a message.

Elaborating on the details of the WSS:SMS format and syntax can be tedious. Instead, this paper will introduce the idea by walking the readers through a scenario:

Two companies, A and B, are doing business on the Internet. A provides the web service. B uses the service. Every request and response should be authenticated. Every request and response will be archived in case of future disputes. The communicating companies don't want any other party to be able to understand the messages being passed.

The approach can be like this:

- Every message should be signed with the sender's private key. Sender should pass his public key to the receiver for signature verification.
- Every message should be encrypted with a symmetric key. To be more secure, the two parties should not use a pre-shared symmetric key. Instead, the sender should pass the symmetric key within the message in a secure manner.

Step1. A and B get their public-key based security token, such as X.509 digital certificate.

The certificate should be issued by a Certificate Authority (CA) that both A and B can trust. The CA can be a third party, such as VeriSign, Entrust, etc. Or in this case it can be A itself, because B must trust A, and A of course trusts itself. The distribution of certificate is a complicated issue and is beyond the scope of this paper.

The X.509 digital certificate[10] contains the owner's name, owner's pubic key and validation date, etc. It also contains the issuer (CA)'s name and a signature. The signature was calculated based on the certificate contents and CA's private-key. The CA's public key should be well accessible so people can easily verify that the certificate is indeed issued by that CA.

The public key inside the certificate is meant to be seen by everyone. But the private key, on the other hand, must be kept secret and secure.

Once A and B get the digital certificates, they can use them many times until the certificates expire or are revoked by the CA.

On the B side

Step 2: B creates a SOAP request as usual.

One simple SOAP example was provided in section 1.2.1.

Step 3: B presents the security token, i.e. X.509 in the <Security> element inside the SOAP header.

This can be done in several ways [11]:

- The X.509 can be split up and each of its attributes can be put into different elements.
- It can be carried as binary data. This is the recommended way to present all signed security keys, including X.509 and Kerberos keys.
- It can be referred to as a URL. A should be able to access that URL to fetch the certificate.
- If the X.509 certificate is issued by A, then B can refer to it with A's name and the certificate serial number. A should know how to fetch the X.509 from its own network.

Step 4. B chooses the target element to sign.

WSS:SMS provides the flexibility for the sender to only sign the elements that he considers important. The target can be the whole message, a single element, or multiple elements. If there is any ambiguity, the element(s) must be tagged and referred to with a unique ID.

The specification allows multiple signatures and certificates to be attached to a single message. The targets to be signed can be from different parts of the message, or can be overlapping. This is very important for distributed applications that have messages flow through multiple processing stages. For example, a company's purchasing department may create the purchasing request, sign the OrderID and attach the signature. Next, the accounting department may add a BillingID in the request and sign the OrderID and BillingID together again, and attach the second signature to the message. In this way both departments can be held accountable.

Step 5. B canonicalizes and transforms the SOAP message.

To canonicalize and transform SOAP message is due to the fact that a SOAP message is ASCII based. Two SOAP messages many have the same business logic, but have some textual difference, such as an extra space between the first name and the last name. Even when the two messages are exactly the same, different XML parsers may handle line delimiters in different ways when trying to serialize and de-serialize an XML data structure. If the algorithm that is used to verify the digital signature runs against a slightly different serialized version of the data, the result will fail, although logically the verification should pass. Therefore the SOAP message should be canonicalized and transformed first to reach a consistent binary representation before the message can be digested and signed.[12]

B should specify what canonicalization and transformation algorithm he will use. Two examples of recommended algorithms are Exclusive XML Canonicalization (URL: http://www.w3.org/2001/10/xml-exc-c14n#) and SOAP Message Normalization (URL:http://www.w3.org/TR/2003/NOTE-soap12-n11n-20030328/).

Step 6. B calculates the message digest value based on canonicalization and transformation result.

Digesting is one-way hashing calculation. The digest result is usually short, e.g. MD5's result is 128 bits, and SHA's result is 160 bits. The reason for doing digesting will be explained in next paragraph.

Step 7. B calculates the signature based on the digest result.
B can use his private key to sign the digest result. The reason for signing on the digest value, not on the original SOAP message, is that the SOAP message can be quite long. The process of applying the sender's private key and cryptography algorithm to sign the full message could significantly impact the performance of Web service. On the other hand, since the digest value is short and unique for that message, signing on the digest result has the same effect as signing the original message, but significantly improves the performance.

B must specify what signing algorithm and which security token were used.
The signature process and syntax must follow W3C XML signature standard. [13]

After the signing process, B can continue with the encryption process.
Step 8. B chooses data items to encrypt.
WSS:SMS provides the flexibility for the sender to encrypt only the content that he wants to encrypt. It can be elements, the contents of elements or any arbitrary data.

Step 9. B passes the encryption key.
WSS:SMS specifies that the encryption key must be symmetric. The sender and receiver may already share the key. In this case, only the key ID needs to be transferred. In another option, the sender can generate a new secret symmetric key and send the key along with the message. This new key must be encrypted with the receiver's public key in case a hacker sniffs the message.

Please don't confuse the symmetric encryption key with the asymmetric public key: the asymmetric public key is only used to encrypt the content of symmetric encryption key.

Step 10. B encrypts the target data and replaces the original data with the encrypted result.
B must describe what encryption algorithm and encryption key were used. The encryption process and syntax must follow the W3C encryption standard. [14]

Step 11. B sends the SOAP message to A over HTTP protocol.

Step 12. A processes the message.
When A receives the message, A should first try to decrypt the message and verify the signature based on the descriptions inside the message. The sequence of decrypting and verifying may vary. After this seemingly simple but indeed very complicated process, A can process the message as normal.

Below is a real example of a signed and encrypted SOAP message. It is copied from the WSS:SWS standard[9] but slightly modified for easier understanding.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
```

```
3       xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
4       xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext"
5       xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
6   <S:Header>
7   <wsse:Security>
8       <wsse:BinarySecurityToken
9               ValueType="wsse:X509v3"
10              EncodingType="wsse:Base64Binary"
11              Id="X509Token">
12              MIIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
13      </wsse:BinarySecurityToken>
14      <xenc:EncryptedKey>
15      <xenc:EncryptionMethod Algorithm= "http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
16              <ds:KeyInfo>
17              <ds:KeyName>CN=Hiroshi Maruyama, C=JP</ds:KeyName>
18              </ds:KeyInfo>
19              <xenc:CipherData>
20                      <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
21                      </xenc:CipherValue>
22              </xenc:CipherData>
23              <xenc:ReferenceList>
24              <xenc:DataReference URI="#MsgBody"/>
25              </xenc:ReferenceList>
26      </xenc:EncryptedKey>
27      <ds:Signature>
28              <ds:SignedInfo>
29                      <ds:CanonicalizationMethod Algorithm=
30                      "http://www.w3.org/2001/10/xml-exc-c14n#"/>
31                      <ds:SignatureMethod Algorithm=
32                      "http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
33              <ds:Reference URL="#MsgBody">
34                      <ds:Transform          Algorithm=
35                         "http://www.w3.org/2001/10/xml-exc-c14n#"/>
36                      </ds:Transforms>
37                     <ds:DigestMethod lgorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
38                     </ds:DigestMethod>
39                      <ds:DigestValue>EULddytSo1...</ds:DigestValue>
40              </ds:Reference>
41              </ds:SignedInfo>
42              <ds:SignatureValue>
43              BL8jdfToEb1l/vXcMZNNjPOV...
44              </ds:SignatureValue>
45              <ds:KeyInfo>
46              <wsse:SecurityTokenReference>
47                      <wsse:Reference URI="#X509Token"/>
48              </wsse:SecurityTokenReference>
49              </ds:KeyInfo>
50      </ds:Signature>
51  </wsse:Security>
52  </S:Header>
53  <S:Body>
```

```
54      <xenc:EncryptedData
55         Type=http://www.w3.org/2001/04/xmlenc#Element id="#MsgBody">
56              <xenc:EncryptionMethod
57              Algorithm="http://www.w3.org/2001/04/xmlenc#3des-cbc"/>
58              <xenc:CipherData>
69                      <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
60                      </xenc:CipherValue>
61              </xenc:CipherData>
62      </xenc:EncryptedData>
63  </S:Body>
```

In this example message:
Line 7~51 is the <Security> element added in the SOAP header.
Line 8~13 is a X.509 digital certificate in binary format.
Line 29~30 specifies the canonicalization method.
Line 31~32 specifies the signing method
Line 33 specifies the digesting and signing target. In this example it is the whole SOAP body.
Line 34~36 specifies the transform method.
Line 37~39 specifies the digest method and the digest values.
Line 42~44 is the signature value.
Line 45~49 specifies what security token is used. In this example, it is the x.509 specified in line 8~13.

Line 15~26 are the encryption related information.
Line 16~18 specifies that the symmetric encryption key is encrypted by public key with the ID of “CN=Hiroshi Maruyama, C=JP”’.
Line 19~22 is the encrypted value of symmetric encryption key.
Line 23~25 specifies which part of the message will be encrypted. In this example it is the whole SOAP body.
Line 56~57 specifies what symmetric encryption method is used.
Line 58~61 is the encryption result of the message body.

3.4 Drawback of WSS:SMS

WSS:SMS does solve the security challenges we discussed in the first section. However, it still has some issues.

3.4.1. Lose readability when message is encrypted.

One big selling point of Web Service is the ASCII format and human readability. These characteristics provide better compatibility and are very important for business-to-business communication. But when the message is encrypted, the message becomes a string of characters that can no longer be understood by humans. Did we just lose the major benefit of web service?

3.4.2. Encryption can be the performance bottleneck.

As we discussed before, a cryptography algorithm is very computing expensive, even with a symmetric key algorithm. When we sign the message, we sign the shorter digest

result, which is usually 128 bits or 160 bits and this saves a lot of calculation. But when we deal with encryption, we have no choice but to do the calculation on the entire message, which can be quite long. And by now the calculation can only be done in software level. The encryption process may become a possible bottleneck for some web services.

3.4.3 The standard and implementation are relatively untested.

WSS:SMS is a relatively new standard. Actually it just reached the "standard" level in January 2004. WSS:SMS also relies on many XML standards, such as XML encryption and XML signature, which are also relatively new. The inter-operability is a big issue. The security of the standard and implementation need to be tested and proved in real life usage.

4. Practical recommendations

So far we discussed two types of solutions: network protocol based solutions such as using IPSec and SSL, and the application level solution: WSS:SMS. They all have their own advantages and disadvantages. Some disadvantage comes with technology itself, e.g. IPSec and SSL don't support end-to-end security. Others can be improved given more time: e.g. WSS:SMS's implementation will be improved by fixing bugs. WSS:SMS encryption may no longer be the bottleneck when CPU speeds increase.

So in the current situation, what should businesses do to provide web service in a cheap, fast and secure manner? In next section, I will give some practical recommendations.

4.1. Always use WSS:SMS to sign the message. Only sign once, and sign the whole message.

IPSec and SSL can provide authentication, confidentiality and integrity, but not repudiation. WSS:SMS can do a good job in all the areas, and can do it pretty efficiently because it signs on the digest result.

Theoretically, you can sign multiple parts of the message. The business reason was discussed in section 3.2, step 4. But when you put that theory into practice, it causes lots of unnecessary complexity for development and testing. Do you want to spend a lot of time on trouble-shooting? Can you trust the security of the implementation when dealing with such a high level of complexity? Unless your business really needs it, try to avoid it and use simpler mechanism. The deployment will be much faster and the service will be more reliable.

4.2. Always use X.509 digital certificate. Always attach the certificate in the binary form.

Though WSS:SMS allows any security token, X.509 is the best choice. X.509 and its related technology are mature, well understood, widely deployed, and proven to be

secure. Because of this, OASIS provides a separate specification [11] just to describe the inter working between X5.09 and WSS:SMS.

X.509 can be represented in the message in several ways. It is recommended to embed X.509 as a binary string. This is a straightforward and the most secure method for the sender and the receiver. Other methods require more steps to access the certificate, therefore are exposed to more risks. e.g., if X.509 is referenced by a URL, then what if the server hosting the URL is down? Even worse, what if that server is hacked?

4.3. If only point-to-point confidentiality is needed, don't use WSS:SMS encryption feature. Use SSL or IPSec.

Many web services are between two businesses across the Internet. These businesses are happy as long as the messages are secure between two business' networks. In this point-to-point situation, SSL or IPSec is a much better choice than WSS:SMS encryption.

- SSL and IPSec standards and implementation are more mature and proved to be secure. Different vendor's implementations are more compatible than WSS:SMS.
- SSL and IPSec are integrated into most firewalls and servers, thus the deployments are cheaper.
- There are many hardware level implementations of SSL and IPSec. They are much faster than the software implementation.
- Application developers don't need to worry about encryption. The development time can be drastically reduced

To use HTTP over SSL (HTTPS) is very common. Another popular solution is to set up IPSec VPN or SSL VPN. The detail of the deployment is beyond the scope of this paper.

4.4. If end-to-end confidentiality is indeed needed, use WSS:SMS to encrypt the whole message.

If you have to use the WSS:SMS encryption feature, try to encrypt the whole message. This is the easiest for processing. This may save you a lot of time, and still provides what you need.

4.5 As a general rule, the simpler, the better

You can do it, doesn't mean that you should do it. Consider your requirements carefully, and use the simplest feature that can satisfy your requirements.

5. Conclusion

Doing web service with the SOAP protocol on the Internet faces many security challenges. Using network level protocols, such as SSL and IPSec, can address some of the challenges, but with some deficiencies.
WSS:SMS is a new specification that is intended to address all the web service security issues on the application level, without depending on underlying network protocols. This

paper gives an introduction to this new specification. Also by walking through an example, I hope to give the readers a good sense of WSS:SMS's usage and capability, as well as its drawbacks.

There is no single perfect solution. So I give my practical recommendations on how to provide web service security in the current situation: Use the WSS:SMS signature feature to provide message integrity, authentication and non-repudiation. Use X.509 whenever possible. Try to use IPSec or SSL to provide confidentiality. In general, try to use the simple features of WSS:SMS.

Reference

[1] Microsoft. Security in a Web Services World: A Proposed Architecture and Roadmap. April 7, 2002 URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwssecur/html/securitywhitepaper.asp

[2] Aphrodite Tsalgatidou , Thomi Pilioura, An Overview of Standards and Related Technology in Web Services, December 2002, URL: http://www.infosys.tuwien.ac.at/Teaching/Courses/IntAppl/Papers/AnOverviewOfStandardsAndRelatedTechnologyInWebServices.pdf

[3] SOAP standard group, W3C. SOAP Version 1.2 Part 1: Messaging Framework June 2003.URL: http://www.w3.org/TR/soap12-part1/

[4] W3C, Web Services Description Language (WSDL) 1.1, March 15, 2001. URL: http://www.w3.org/TR/wsdl

[5] Eric Cole, Jason Fossen, Stephen Northcutt, Hal Pomeranz. SANS Security Essential with CISSP CBK, Version 2.1, 2003, page 912

[6] Anonymous. Introduction to SSL. URL: http://developer.netscape.com/docs/manuals/security/sslin/contents.htm

[7] Anita Karve. IP Security: Security extensions to IP bring authentication and privacy to the Internet. 02/01/1998 URL: http://www.networkmagazine.com/article/NMG20000711S0001

[8] Anonymous, Copyright Javvin Company, IPsec ESP: IP Encapsulating Security Payload. URL: http://www.javvin.com/protocolESP.html

[9] OASIS standard group. Web Services Security: SOAP Message Security 1.0 Tuesday, 17 February 2004 URL: http://www.oasis-open.org/committees/download.php/5531/oasis-200401-wss-soap-message-security-1.0.pdf

[10] R. Housley, W. Ford, W. Polk, D. Solo, RFC 2495, Internet X.509 Public Key Infrastructure Certificate and CRL Profile January 1999 URL: http://www.ietf.org/rfc/rfc2459.txt

[11] OASIS standard group. Web Service Security X509 Certificate Token Profile, May 19th, 2003, URL: http://www.oasis-open.org/committees/download.php/2131/WSS-X509-04.pdf

[12] Bilal Siddiqui, XML Canonicalization, September 18, 2002, URL: http://webservices.xml.com/pub/a/ws/2002/09/18/c14n.html

[13] W3C Recommendation, XML Signature Syntax and Processing-12, February 2002. URL: http:// http://www.w3.org/TR/xmldsig-core/

[14] W3C recommendation. XML Encryption Syntax and Processing, December 10, 2002. URL: http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/

Acronyms

AH:	Authentication Header
CA:	Certificate Authority
ESP:	Encapsulated Security Payload
HTTP:	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol - Secure
IKE:	Internet Key Exchange
IMAP:	Internet Message Access Protocol
IPSec:	IP Security protocol
MD5:	Message Digest Algorithm #5
OASIS:	Organization for the Advancement of Structured Information Standards
SHA:	Secure Hash Algorithm
SOAP:	Simple Object Access Protocol
SSL:	Secure Sockets Layer
TTL:	Time to live
URL:	Uniform Resource Locator

W3C:	World Wide Web Consortium
WSDL:	Web Service Description Language
WSS:SMS:	Web Service Security: SOAP Message Security
XML:	eXtensible Markup Language

Last Updated: May 28th, 2017

Upcoming Training

SANS Stockholm 2017	Stockholm, Sweden	May 29, 2017 - Jun 03, 2017	Live Event
Security Operations Center Summit & Training	Washington, DC	Jun 05, 2017 - Jun 12, 2017	Live Event
SANS Houston 2017	Houston, TX	Jun 05, 2017 - Jun 10, 2017	Live Event
Community SANS Ottawa SEC401	Ottawa, ON	Jun 05, 2017 - Jun 10, 2017	Community SANS
SANS San Francisco Summer 2017	San Francisco, CA	Jun 05, 2017 - Jun 10, 2017	Live Event
SANS Charlotte 2017	Charlotte, NC	Jun 12, 2017 - Jun 17, 2017	Live Event
SANS Rocky Mountain 2017 - SEC401: Security Essentials Bootcamp Style	Denver, CO	Jun 12, 2017 - Jun 17, 2017	vLive
SANS Secure Europe 2017	Amsterdam, Netherlands	Jun 12, 2017 - Jun 20, 2017	Live Event
Community SANS Portland SEC401	Portland, OR	Jun 12, 2017 - Jun 17, 2017	Community SANS
SANS Rocky Mountain 2017	Denver, CO	Jun 12, 2017 - Jun 17, 2017	Live Event
SANS Minneapolis 2017	Minneapolis, MN	Jun 19, 2017 - Jun 24, 2017	Live Event
SANS Columbia, MD 2017	Columbia, MD	Jun 26, 2017 - Jul 01, 2017	Live Event
SANS Cyber Defence Canberra 2017	Canberra, Australia	Jun 26, 2017 - Jul 08, 2017	Live Event
SANS Paris 2017	Paris, France	Jun 26, 2017 - Jul 01, 2017	Live Event
SANS London July 2017	London, United Kingdom	Jul 03, 2017 - Jul 08, 2017	Live Event
Cyber Defence Japan 2017	Tokyo, Japan	Jul 05, 2017 - Jul 15, 2017	Live Event
Community SANS Phoenix SEC401	Phoenix, AZ	Jul 10, 2017 - Jul 15, 2017	Community SANS
SANS Munich Summer 2017	Munich, Germany	Jul 10, 2017 - Jul 15, 2017	Live Event
SANS Cyber Defence Singapore 2017	Singapore, Singapore	Jul 10, 2017 - Jul 15, 2017	Live Event
Community SANS Minneapolis SEC401	Minneapolis, MN	Jul 10, 2017 - Jul 15, 2017	Community SANS
SANS Los Angeles - Long Beach 2017	Long Beach, CA	Jul 10, 2017 - Jul 15, 2017	Live Event
Mentor Session - SEC401	Macon, GA	Jul 12, 2017 - Aug 23, 2017	Mentor
Mentor Session - SEC401	Ventura, CA	Jul 12, 2017 - Sep 13, 2017	Mentor
Community SANS Atlanta SEC401	Atlanta, GA	Jul 17, 2017 - Jul 22, 2017	Community SANS
Community SANS Colorado Springs SEC401	Colorado Springs, CO	Jul 17, 2017 - Jul 22, 2017	Community SANS
SANSFIRE 2017	Washington, DC	Jul 22, 2017 - Jul 29, 2017	Live Event
Community SANS Charleston SEC401	Charleston, SC	Jul 24, 2017 - Jul 29, 2017	Community SANS
SANSFIRE 2017 - SEC401: Security Essentials Bootcamp Style	Washington, DC	Jul 24, 2017 - Jul 29, 2017	vLive
Community SANS Fort Lauderdale SEC401	Fort Lauderdale, FL	Jul 31, 2017 - Aug 05, 2017	Community SANS
SANS San Antonio 2017	San Antonio, TX	Aug 06, 2017 - Aug 11, 2017	Live Event
SANS Prague 2017	Prague, Czech Republic	Aug 07, 2017 - Aug 12, 2017	Live Event