The Problem of Handling Multiple Headers in WS-Security Implementations

Ernesto Damiani, Valerio Coviello, Fulvio Frati
Computer Science Department
Università degli Studi di Milano
Milan, Italy
{ernesto.damiani, fulvio.frati}@unimi.it
valerio.coviello@studenti.unimi.it

Claudio Santacesaria
Research&Development Department
Rototype S.p.A.
Milan, Italy
claudio.santacesaria@rototype.com

***Abstract*—This paper discusses some practical problems encountered when generating multiple WS-Security confidentiality headers to be handled by different intermediaries along the SOAP message delivery chain of a real banking application. A patch using a special-purpose encryption component is described.**

Keywords—WS-Security, SOAP, banking application, security header.

I. INTRODUCTION

SOAP is a widespread standard for encoding messages exchanged within Service Oriented Architectures (SOA). Version 1.2 was introduced as early as 2003 [1] as an XML based messaging protocol that consists of three parts: an *envelope*, which defines a framework for describing what is in a message and how to process it, a set of *encoding rules* for expressing instances of application-defined datatypes, and a convention for representing *remote procedure calls* and *responses*.

While many applications rely on a basic *point-to-point* scenario where a client uses SOAP messages over HTTP to send parameters to a remote Web service, the SOAP standard XML-based data model can be used for exchanging information between peers in any decentralized, distributed environment.

A more complex scenario is the one of a *delivery chain*, where some parts of a SOAP message (called *header blocks*) are intended for being intercepted by one or more SOAP *intermediaries* on the message path to the ultimate receiver. A SOAP intermediary is any application that is capable of both receiving and forwarding SOAP messages. According to the standard, a SOAP intermediary should not forward a SOAP header block intended for it, but consume it locally, unless the block contains a special attribute enabling relay. This feature is particularly relevant to Web service security and was first proposed in [2].

Later, the WS-Security standard introduced some special SOAP headers that contain information needed to protect message integrity, message confidentiality, and single message authentication. WS-Security headers can be used to describe a wide variety of security models and encryption technologies.

Coupling WS Security headers with intermediaries along message paths should enable powerful design patterns for sharing responsibilities when enforcing protection policies [3]. In some of these patterns [4], a SOAP message carries multiple headers for protecting integrity or confidentiality of different XML sub-trees within the message. Since each header can be handled by a different intermediary, the pattern can delegate decryption or signature control to different agents [5]. However, as we shall see, WS-Security implementations may not support generating and handling multiple security headers. This paper, after presenting a motivating scenario (Section II) describes some issues of WS Security implementations (Section III) and introduces the problem of handling multiple WS-Security headers (Section IV). Then, it outlines a practical solution (Section V). Section VI draws the conclusion and discusses perspectives for future work.

II. MOTIVATING SCENARIO: DIGITAL CHECKS

Today, most banks are operating by offering Web-based services that allow customers to access their private profile, check details relative to their account, and execute a growing set of banking operations. Such remote services need to satisfy a number of challenging security requirements to ensure that customers access them with the same confidence level they would have in a bank branch's front desk. An emerging new service is the one giving customers the opportunity of managing *digital checks* through an online interface.

Digital checks are an evolution of traditional ones, where new elements have been added to the classic check template. These elements carry machine-readable, certified information on *(i)* the identity of the *drawee*, namely the bank that supplies the empty check to the user, *(ii)* the identity of the *drawer*, namely the person authorized by the drawee to draw the check, and *(iii)* the integrity and consistency of the data filled in the check.

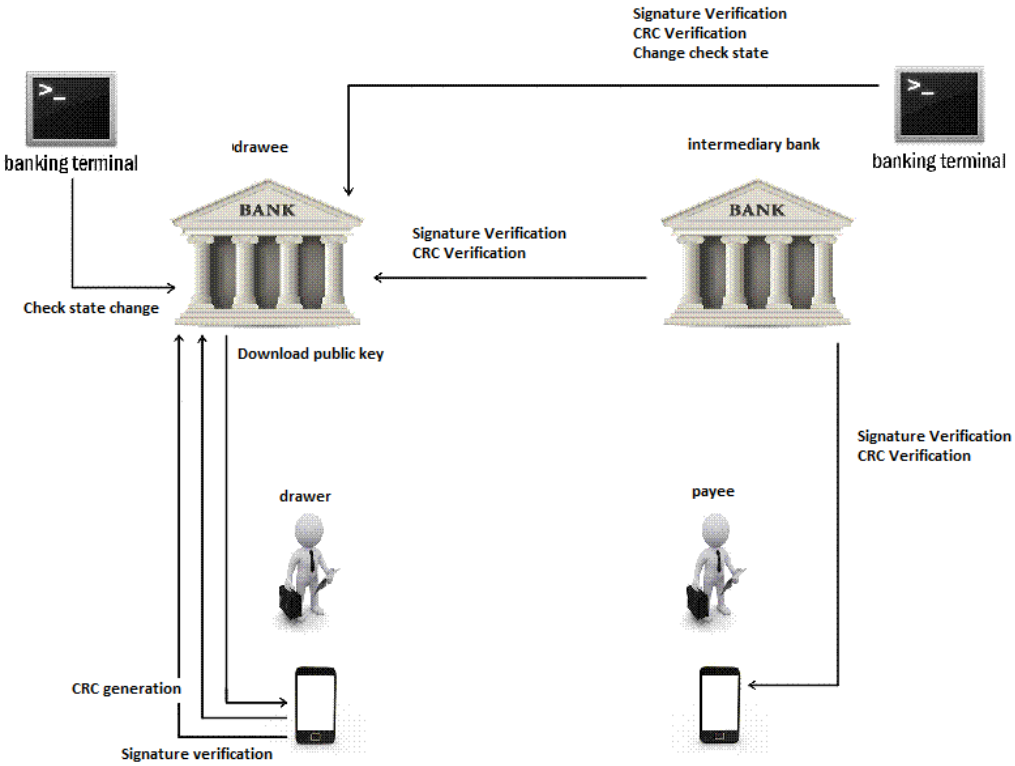

Figure 1: Digital checks process.

The goal of the introduction of the digital check is, in particular, supporting the use of checks in the context of Web-based banking.

Furthermore, there is also the need of protecting checks from new, sophisticated frauds, preventing tampering and unauthorized modifications. Finally, the digital check will allow automatic verification of check authenticity using standard digital cameras rather than costly special-purpose devices.

The exploitation of check images can pave the way to more complex and value-added operations, like the possibility for a customer to cash in checks sent by remote drawers. In that case, the drawee will prepare the digital check, with all the new graphic elements in the right place. In particular, a QR code printed on the front will include all the information related to the drawee bank. The drawee will send the check to the drawer that will fill and use it when needed.

The final receiver of the check (payee) will cash it in, scanning the check in ATMs equipped with the specific QR code reader, and, in particular, will be able to verify the data on the check using a mobile client and the provided web services. The client will scan the QR code and will ask to the payee to insert the amount, his/her name, the issuing date, and the control code printed on the check. The data will be sent to the drawee bank, through the payee's personal bank that will act as intermediary confirming the validity and integrity of the check.

Furthermore, the intermediary will verify the consistency of the digital signature included in the QR code and verify that the amount is available on the drawer account. Figure 1 depicts the digital check process described above.

When implementing this process on a SOA, all services described above should rely on security protocols to guarantee the integrity and confidentiality of the check data, supporting the high level of privacy the bank needs to assure dealing with this information.

III. Implementation of Security Standard

The WS-Security protocol defines how security headers will be represented and managed within the SOAP envelope. Furthermore, it includes a set of libraries to sign, encrypt, and decrypt the message, or parts of the message, exploiting the XML Encryption and XML Signature standards. In practice, the basic point-to-point scenario where a client using SOAP messages over HTTP to send to remote Web service whose confidentiality and integrity are protected by WS-Security, can be easily implemented through a basic configuration of AXIS and Apache Tomcat.

In the point-to-point scenario, WS-Security headers are used to define which security features (e.g., signature, encryption, or timestamp) need to be activated for inbound and outbound SOAP messages. For instance, a header can be used to protect the integrity of incoming messages with a digital signature, while another specifies how the body of the message is encrypted.

It is easy to see that the process described in Section II needs an intermediate actor acting as a proxy from the payee to

the drawee bank. The intermediary, i.e. the payee bank, will authenticate its customer, verify if the request can be satisfied directly by the bank itself (when the bank is also the drawee of the check), and then it will forward it to the actual drawee.

It is important to remark that the configuration of the WS-Security stack to manage the intermediary presents big differences with the basic point-to-point scenario. In particular, since the intermediary bank has to decrypt only those parts of the message that are needed to authenticate the user and to identify the drawer of the check, the message needs to be managed in two stages, protecting data that are directed to the final receiver.

Specifically, the client needs to encrypt with the intermediary's public key those information needed to authenticate the user and pay the check, namely a web banking *username* and *password*, *payee name*, *amount*, *check code line*, and *drawee bank* information. Then, the client needs to encrypt with the public key of the drawee bank all the fields that are requested by the system to validate and verify the signature in order to cash in the check, namely the fields *signature algorithm* and *crcCode*, as written by the customer on the check. This way, the intermediary node will not be able to read the protected data that can be decrypted only by the actual drawee bank.

IV. Double Security Header Problem

In principle, the scenario depicted in Section III can be straightforwardly implemented following the classical WS-Security guidelines for protecting web services, claiming to guarantee confidentiality and integrity of data also in delivery chain scenarios [3] with more than one intermediary.

However, in our first Axis-based implementation we noticed that even if the actors were configured following all the best practices indicated in the standard guidelines [8], the client failed in creating all the security headers needed for the message. As we have seen, in case of delivery chain communication, a different security header must be generated for each node that participates in the process. In our case, each SOAP envelope must contain a security header for the intermediary bank, and one for the drawee bank. However, the envelopes produced by the development environment contained a single security header. In this way, it was not possible to apply the double encryption scheme with two distinct public keys, and the interception of the SOAP message by the intermediary generated a "*Security processing failed*" exception. In fact, the single security header produced by the client was processed by the intermediary, which could read also the data directed to the final receiver. The intermediary tried to manage data that were encrypted with the drawee public key, which is not in its possess, thus generating the exception.

Figure 2 presents the security header as produced by the WS-Security stack in the client application. It should be noted that the header contains two *EncryptedKey* and *ReferenceList* elements, respectively for the intermediary and the final receiver nodes, but inside the same header.

```
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004
/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
 soapenv:mustUnderstand="1">
    <xenc:EncryptedKey
    Id="EncKeyId-F47A4FB44EF83F9
    8FD13948041684982">
        <xenc:EncryptionMethod
        Algorithm="http://www.w3.org/2001/
        04/xmlenc#rsa-1_5" />
        <ds:KeyInfo xmlns:ds=
            "http://www.w3.org/2000/09/xmldsig#">
            <wsse:SecurityTokenReference>
                <ds:X509Data>
                    <ds:X509IssuerSerial>
                        <ds:X509IssuerName>...</ds:X509IssuerName>
                        <ds:X509SerialNumber>...</ds:X509SerialNumber>
                    </ds:X509IssuerSerial>
                </ds:X509Data>
            </wsse:SecurityTokenReference>
        </ds:KeyInfo>
        <xenc:CipherData>
            <xenc:CipherValue>...</xenc:CipherValue>
        </xenc:CipherData>
        <xenc:ReferenceList>...</xenc:ReferenceList>
    </xenc:EncryptedKey>
    <xenc:EncryptedKey Id="EncKeyId-F47A4FB44EF83F9
       8FD13948041685804">
        <xenc:EncryptionMethod Algorithm=
        "http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
        <ds:KeyInfo xmlns:ds="http://www.w3.org/2000
            /09/xmldsig#">
            <wsse:SecurityTokenReference>
                <ds:X509Data>
                    <ds:X509IssuerSerial>
                        <ds:X509IssuerName>...</ds:X509IssuerName>
                        <ds:X509SerialNumber>...</ds:X509SerialNumber>
                    </ds:X509IssuerSerial>
                </ds:X509Data>
            </wsse:SecurityTokenReference>
        </ds:KeyInfo>
        <xenc:CipherData>
            <xenc:CipherValue>...</xenc:CipherValue>
        </xenc:CipherData>
        <xenc:ReferenceList>...</xenc:ReferenceList>
    </xenc:EncryptedKey>
</wsse:Security>
```

Figure 2: security header as-is

To further analyze the problem, we modified the SOAP message in Figure 2 using the SoapUI[1] tool to add the missing security header. Figure 3 showcases the envelope we crafted using the tool and containing the two headers.

The first header is intended to be managed by the intermediary node. The value "1" in the *mustUnderstand* attribute indicates the security header will be processed by the intermediary, which will decrypt the elements included in *ReferenceList*. Instead, the second header contains the value 0 in the *MustUnderstand* attribute, indicating that the intermediary should skip the element indicated in the reference list.

The intermediary will then process the first header only, and forward the message, removing the first header, to the drawee bank service. The second header remains included in the message and is used by the drawee to decrypt the data; following this procedure, the process is concluded correctly preserving the privacy and integrity of the data.

[1] http://www.soapui.org/

```
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss
 /2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
 soapenv:mustUnderstand="1">
    <xenc:EncryptedKey Id="EncKeyId-F47A4FB44EF83F
          98FD13948041684982">
       <xenc:EncryptionMethod Algorithm=
      "http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
       <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
          <wsse:SecurityTokenReference>
             <ds:X509Data>
                <ds:X509IssuerSerial>
                   <ds:X509IssuerName>...</ds:X509IssuerName>
                   <ds:X509SerialNumber>...</ds:X509SerialNumber>
                </ds:X509IssuerSerial>
             </ds:X509Data>
          </wsse:SecurityTokenReference>
       </ds:KeyInfo>
       <xenc:CipherData>
          <xenc:CipherValue>...</xenc:CipherValue>
       </xenc:CipherData>
       <xenc:ReferenceList>...</xenc:ReferenceList>
    </xenc:EncryptedKey>
 </wsse:Security>
 <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss
/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
soapenv:mustUnderstand="0">
    <xenc:EncryptedKey Id="EncKeyId-F47A4FB44EF83F98FD13948041685804">
       <xenc:EncryptionMethod Algorithm=
       "http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
       <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
          <wsse:SecurityTokenReference>
             <ds:X509Data>
                <ds:X509IssuerSerial>
                   <ds:X509IssuerName>...</ds:X509IssuerName>
                   <ds:X509SerialNumber>...</ds:X509SerialNumber>
                </ds:X509IssuerSerial>
             </ds:X509Data>
          </wsse:SecurityTokenReference>
       </ds:KeyInfo>
       <xenc:CipherData>
          <xenc:CipherValue>...</xenc:CipherValue>
       </xenc:CipherData>
       <xenc:ReferenceList>...</xenc:ReferenceList>
    </xenc:EncryptedKey>
</wsse:Security>
```

Figure 3: Security Header to be

The manual procedure can be considered as an evidence of a problem in the development environment's support of WS-Security stack to manage the delivery chain scenario, since the application works correctly using the double header.

In order to provide a temporary patch to this problem, we then focused on developing a component that will allow the recovering and preserving the data inside the security header which the intermediary discards after processing the header.

V. Proposed Solution

Our proprietary solution consists in *a-priori* encryption of the data directed to the drawee before creating the SOAP envelope, and including that data inside the SOAP envelope that will be created and sent to the intermediate service.

This way, when the intermediary node receives the message, it will decrypt it, since the mustUnderstand flag is set to 1. After reading the message's body, the intermediary has all the information needed to manage the check and to determine if it can be managed internally or should be forwarded to the drawee bank. If the intermediary is also the final receiver, then it will decrypt the additional data with its own private key and will execute the full verification of the check. Otherwise, the intermediary service will create a new SOAP envelope starting from the one received by the user, but modifying the security header in order to be managed by the drawee service. We take as pre-requisite that the banks that participate in the digital check program owns all the public keys of the banks in the circuit.

When the drawee service receives the message and verifies the validity and integrity of the data, it will send an acknowledgement to the intermediary indicating if the process has concluded successfully (or not). Then, the intermediary will forward the acknowledgement to the customer. The message will be again encrypted with the public key of the intermediary, first, and then sent in plain text to the customer. The latter is justified by the fact that the digital check framework will not request each customer to own a public key.

It is important to remark that all the communication between the payee and the intermediary bank will be executed exploiting a HTTPS channel, thus giving a good level of security during the delivery of the final confirmation message. Finally, the encryption methods implemented in our solution apply the RSA algorithm on all the selected elements, and the values have been further encoded using Base64 to avoid problems during the transmission of the strings.

A. *System Configuration*

The system has been developed in Java using Eclipse and deployed under Apache Tomcat 7.0. Axis2[2] has been chosen as web services engine. In the following, we give an overview of the most important configuration files used to set up the proprietary solution.

The WS-Security stack functionalities are provided within the engine through the open source module Rampart[3]. Figure 4 shows the configuration needed to select the encryption method used by the client.

The intermediary node configuration inside the Axis2 *service.xml* file indicates that the inbound messages need to be encrypted (see Figure 5), while to the outbound communication should only be added the timestamp. Please note that the security between the intermediary and the drawee services is managed by the proprietary tool and not by WS-Security directly.

```
public static byte[] encrypt(String text,
        java.security.cert.Certificate cert) {

    byte[] cipherText = null;
    try {
        final Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.ENCRYPT_MODE, cert);
        cipherText = cipher.doFinal(text.getBytes());
    } catch (Exception e) {
        e.printStackTrace();
    }
    return cipherText;
}
```

Figure 4: Client-side encryption.

[2] http://axis.apache.org/
[3] http://axis.apache.org/axis2/java/rampart/

```
<parameter name="InflowSecurity">
    <action>
    <items>Encrypt</items>
    <passwordCallbackClass>
        classes.PWCallback
    </passwordCallbackClass>
    <decryptionPropFile>
        classes/serviceSecurity.properties
    </decryptionPropFile>
    </action>
</parameter>
<parameter name="OutflowSecurity">
    <action>
    <items>Timestamp</items>
    </action>
</parameter>
```

Figure 5: Intermediary service configuration.

The drawee service has been configured to understand the data received by the intermediary through the *decrypt* function shown in Figure 6. The function will decrypt the nodes included in the reference list.

Finally, the listing in Figure 7 configures the drawee service in order to manage encrypted communication with the intermediary nodes, since all the banks in the circuit own public/private key pairs, and the communication should be protected in both directions.

However, differently for the client, the encryption will be applied on the full body of the message, and the envelope will include only one security header.

VI. Conclusions

In this paper we described some open issues shown by WS Security implementations when handling multiple WS-Security headers that may prevent effective applications of complex security patterns. We plan to extend our analysis to other development environments. In parallel, we are working toward improving the performance of security enforcement on delivery chains by allowing intermediaries to handle multiple similar messages [7].

```
private static String decrypt(byte[] text, PrivateKey key) {
    byte[] decryptedText = null;
    try {
        final Cipher cipher =
            Cipher.getInstance("RSA/ECB/PKCS1Padding");
        cipher.init(Cipher.DECRYPT_MODE, key);
        decryptedText = cipher.doFinal(text,0,text.length);
    } catch (Exception ex) {
        ex.printStackTrace();
    }
    return new String(decryptedText);
}
```

Figure 6: Final Receiver decryption function.

```
<parameter name="InflowSecurity">
    <action>
    <items>Encrypt</items>
    <passwordCallbackClass>
        classes.PWCallback
    </passwordCallbackClass>
    <decryptionPropFile>
        classes/serviceSecurity.properties
    </decryptionPropFile>
    </action>
</parameter>
<parameter name="OutflowSecurity">
    <action>
    <items>Encrypt</items>
    <encryptionUser>
        intermediaryBank
    </encryptionUser>
    <encryptionPropFile>
        classes/serviceSecurity.properties
    </encryptionPropFile>
    <passwordCallbackClass>
        classes.PWCallback
    </passwordCallbackClass>
    <encryptionParts>
        Body
    </encryptionParts>
    </action>
</parameter>
```

Figure 7: Drawee service configuration.

Acknowledgment

This work was partly supported by the Italian MIUR project SecurityHorizons (c.n. 2010XSEMLC) and by the EU-funded project CUMULUS (contract n. FP7-318580).

References

[1] M Gudgin, M Hadley, N Mendelsohn, JJ Moreau et al.: SOAP Version 1.2, W3C RFC 2003. Available at: http://www.w3.org/TR/2001/WD-soap12-20010709/, 2014.

[2] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, P. Samarati: Securing SOAP e-services. Int. J. Inf. Sec. 1(2): 100-115, 2002.

[3] M. A. Rahaman, A. Schaad, and M. Rits, "Towards secure SOAP message exchange in a SOA, " in SWS '06: Proceedings of the 3rd ACM workshop on Secure Web Services. ACM Press, 2006, pp. 77-84, 2006.

[4] N. Gruschka, M. Jensen, L. Lo Iacono: A Design Pattern for Event-Based Processing of Security-Enriched SOAP Messages. ARES 2010: 410-415, 2010.

[5] M. Jensen, N. Gruschka: Privacy Against the Business Partner: Issues for Realizing End-to-End Confidentiality in Web Service Compositions. DEXA Workshops 2009: 117-121, 2009.

[6] J. Tekli, E. Damiani, R. Chbeir, G. Gianini: SOAP Processing Performance and Enhancement. IEEE T. Services Computing 5(3): 387-403, 2012.

[7] E. Damiani, S. Marrara: Using XML Similarity to Enhance SOAP Messages Security. International Conference on Internet Computing 2008: 260-265, 2008.

[8] Apache Software Foundation: Apache Rampart-Axis2 Security Module. Available at: http://axis.apache.org/axis2/java/rampart/index.html, 2014.