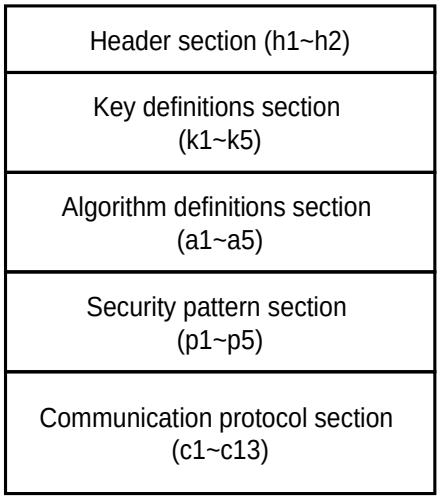
3 Syntax of the WSSL

The syntax of the WSSL document is designed according to the above-mentioned security requirements as well as the operational model shown in Figure 3. Figure 13 shows the general syntactic form of a WSSL document, which consists of the following five sections:

1. *Header*: Since a WSSL document is also an XML document, a WSSL document should begin with an XML declaration that specifies the version of XML being used (e.g., <?xml version="1.0"?>; see line h1 in Figure 13). The root element of the WSSL document in Figure 13 has the start tag <wssl:security_document xmlns:wssl= "http://www.xml-wssl.com/ 2006/wssl" version="1.0">.
2. *Key definition*: This section defines keys that are referred to in the security pattern section. Since the WSSL supports explicit key definition, the key definition scheme proposed in the DSL, XML encryption, and an XML digital signature is insufficient [7,8,9,10,11].
3. *Algorithm definition*: This section defines the algorithms referred to in the security pattern section.
4. *Security pattern*: According to the requirements of element-wise encryption and temporal-based element-wise digital signatures, each encrypted or signed section may contain different security patterns. A security pattern contains the keys used and the associated cryptographic algorithms.
5. *Communication protocol*: This section consists of two subsections (request and response) that describe how to encrypt and embed digital signatures in secured SOAP messages X_s and R_s, respectively.




```
h1      <?xml version="1.0" ?>
h2      <wssl:security_document xmlns:wssl="http://www.xml-wssl.com/2006/wssl" version="1.0">
k1      <!- -Key definitions section >
k2          <!- -key definition >
k3          <!- -key definition >
k4                  :
k5      <!- -End of key definitions section >

a1      <!- -Algorithm definitions section >
a2          <!- -algorithm definition >
a3          <!- -algorithm definition >
a4                  :
a5      <!- -End of algorithm definitions section >

p1      <!- -Security patterns section >
p2          <!- -security pattern >
p3          <!- -security pattern >
p4                  :
p5      <!- -End of security patterns section >

c1      <!- -Communication protocol section >
c2          <!- -encrypt and sing for request >
c3              <!- -security template >
c4                      :
c5              <!- -end of security template>
c6              <!- -embed digital signature >
c7                      :
c8              <!- - end of embed digital signature>
c9          <!- - end of encrypt and sing for request >
c10         <!- - encrypt and sing for response >
c11                     :
c12         <!- - end of encrypt and sing for response >
c13     <!- -End of communication protocol section >
        </wssl:security_document>
```

Figure 13 : Architecture of a WSSL document

3.1. Key Definition

The WSSL supports explicit key definition. This scheme is called “explicit” because it does not require any prior negotiation between the service requester and service provider. According to the proposed operational model shown in Figure 3, the service

requester obtains the WSSL document from the service provider. The location or identity of some of the keys can be present at the WSSL document, and usually these are the keys of the service provider. Each such key is referred to as a *static key*. However, because of the many-to-one operational model shown in Figure 4, it is impossible to identify or locate the keys of the service requesters in the WSSL document. In this case, the WSSL document should instruct the service requester to select one of its own keys. It is obvious that the selected key should satisfy the relevant specification. This key is usually used by the service requester to embed digital signatures in X_s, and is referred to as a *dynamically selected key*. Another problem is that the service provider may have to encrypt some of the data stored in R_s. Note that the service provider does not know the identity of the service requester until it receives X_s, which points to the need for a third key type called *keys applied to signatures*. The identity or location of such keys is obtained from the digital signatures embedded by the service requester in X_s. Generally speaking, the identities and locations of dynamically selected keys and keys applied to signatures are not stored in the WSSL document, but rather can only be obtained when a Web services transaction is executed. However, explicit key definition removes the need for extra negotiation between the service provider and service requester.

It is obvious that the syntax for the key definition defined in [6,7,10,11] becomes insufficient. Both XML encryption and the DSL only support definition of a static key (see Figure 14). The key definition in the WSSL is rooted with the key_definition element, which contains two attributes. The mode attribute specifies the type of key, and has three possible values (static_key, dynamically_selected_key, and key_applied_to_signature), and the key_link attribute specifies the link name of the key definition that is referred to in the security pattern section. Since the key definitions are

referred to in the security pattern or digital signature section, their link names should be specified for later reference.

The key_specification subelement specifies the key. If the defined keys are static or dynamically selected, they have several subelements: key_type, PKI, key_issuer, and key_subject. The key_type element specifies the type of key, which can be a private key or a certificate (see Figure 14A and B). The PKI element identifies the PKI (public key infrastructure) used. The key_issuer and key_subject elements specify the issuer and subject of the key, respectively. If the key is applied to a digital signature, only one subelement, signature_name, is used to specify the name of the digital signature (see Figure 14C). Beside, in order to be more flexible, the key_specification can be added the extra subelement to specific more information about the defined key. For example, to see Figure 14D , there is a extra download_protocol element in the key_specification. It can specific the location of the certificate to download the needed key used to encrypt the message.

The message_to_user subelement is used to identify the actual location of the key. The client-side and server-side proxy programs can use this message to interact with the user if the information in the key_specification element is insufficient for locating the key.

(A)	<key_definition mode="dynamic_selected_key" key_link="clientKeyCert"> <key_specification> <key_type>certificate</key_type> <PKI>X.509v3</PKI> <key_issuer>

	CN=tomcat,OU=ICLAB,O=NTNU,L=Taipei,S=Taiwan,C=TW </key_issuer> </key_specification> <message_to_user> This key should be your certificate of the service request. The corresponding private key is the key whose key_link is clientKeyPrivate. </message_to_user> </key_definition>
(B)	<key_definition mode=“static_key " key_link=“staticKeyPrivate"> <key_specification> <key_type>private_key</key_type> <PKI>X.509v3</PKI> <key_issuer> CN=tomca,OU=ICLAB,O=NTNU,L=Taipei,S=Taiwan,C=TW </key_issuer> <key_subject> CN=taco,OU=ICLAB,O=NTNU,L=Taipei,S=Taiwan,C=TW </key_subject> </key_specification> <message_to_user> This key should be the private key of the service provider. </message_to_user> </key_definition>
(C)	<key_definition mode=“key_applied_to_signature" key_link=“applied_to_signature"> <key_specification> <signature_name>request_sign</signature_name> </key_specification> </key_definition>
(D)	<key_definition mode="static_key" key_link="staticKeyCert"> <key_specification>

```
        <key_type>certificate</key_type>
        <PKI>X.509v3</PKI>
        <key_issuer>
          CN=tomca,OU=ICLAB,O=NTNU,L=Taipei,S=Taiwan,C=TW
        </key_issuer>
        <key_subject>
          CN=taco,OU=ICLAB,O=NTNU,L=Taipei,S=Taiwan,C=TW
        </key_subject>
        <download_protocol
              location="http://www.csie.ntnu.edu.tw/ICLAB/taco.cer"/>
    </key_specification>
    <message_to_user>
        1. It is used to encrypt the SOAP message of the service requester.
        2. It can verify the service provider signature.
    </message_to_user>
</key_definition>
```

Figure 14: Examples of key definition elements

3.2. Algorithm Definition

The algorithm definition section is used to define algorithms that will be executed for securing and unsecuring SOAP messages. The body of this section contains multiple algorithm definitions. The defined algorithms can be used to encrypt and decrypt XML elements, used as digest functions in generating hash codes, or to generate and verify digital signatures. In the WSSL, we adopt the algorithm definition section that appears in the DSL [10,11] (see Figure 15 for examples).

```
<algorithm_definition algorithm_link="aglrsa-v15 " use="SECURITY">
    <algorithm_id>http://www.w3.org/2001/04/xmlenc#rsa-1_5</algorithm_id>
```

```
    <cipher_format XML_text="YES"/>
</algorithm_definition>

<algorithm_definition algorithm_link="aglrsamd5" use="SIGNATURE">
    <algorithm_name>signaturemd5-rsa.ser</algorithm_name>
    <type>RSA</type>
    <version>2.0</version>
    <property mode="ECB" encode="PKCS#1" />
    <cipher_format XML_text="YES" />
    <download_protocol linking_method="DDL"
        jar_file_location="http://iclab.ntnu.edu.tw/signaturemd5-rsa.jar"
        serialization_file_location="http://iclab.ntnu.edu.tw/signaturemd5-rsa.ser" />
</algorithm_definition>
```

Figure 15: Examples of algorithm definition

3.3. Security Pattern

The security pattern definition specifies the combination of security algorithms and encryption and decryption keys. It refers to keys and to algorithm definitions in the key and algorithm definition sections (see Figure 16).

```
<wssl:security_pattern name="request_pattern2">
    <key_information>
        <encryption_key>
            <key_definition key_link="staticKeyCert" />
        </encryption_key>
        <decryption_key>
            <key_definition key_link="staticKeyPrivate" />
        </decryption_key>
    </key_information>
```

```
    <security_algorithm>
        <algorithm_definition algorithm_link="aglrsa-v20" />
    </security_algorithm>
</wssl:security_pattern>
```

Figure 16: An example security pattern definition

3.4. Communication Protocol

The security policy settings for the request and response SOAPs, X_S and R_S, are specified in this section, which includes the encryption scope and the scope of the signed data. Note that the settings are based on the security model of element-wise encryption and temporal-based digital signatures. The settings for X_s and R_s are specified in the request and response elements, respectively.

3.4.1. Request elements

A request element contains two subelements: security_template and digital_signatures. The security_template element contains the template subelement that specifies the element-wise encryption in X_S. Its syntax is identical to the transformation template definition in [10,11], and combines the XPath [18] and scope specifier (i.e., scope attribute). Figure 17 provides an example of the template element, which specifies that the element located by the XPath "/Envelope/Body/transaction/order_info/person_info" should be encrypted according to the security pattern "request_pattern2", where the encryption scope is the “element”.

The digital_signatures element specifies how to embed digital signatures in X_s. Its syntax is identical to the digital signature definition in [11] (an example is provided in Figure 17). The scope of the signed data is located by the XPath to be the element of "/Envelope/Body". This also should be signed after the encryption.

```
<request>
 <security_template>
  <template match="/Envelope/Body/transaction/order_info/person_info">
     <wssl:value-of-encrypted-node scope="element" pattern="request_pattern2"/>
  </template>
 </security_template>

 <digital_signatures>
    <digital-signature name="request_sign" time="AFTER">
        <signature_algorithm>
            <algorithm_definition algorithm_link="aglrsamd5" />
        </signature_algorithm>
        <digest_function>
            <digest_definition digest_link="digest-md5" />
        </digest_function>
        <key_information>
            <sign_key>
                <key_definition key_link=" clientKeyPrivate " />
            </sign_key>
            <verify_key>
                 <key_definition key_link=" clientKeyCert " />
             </verify_key>
          </key_information>
          <digest-element>
              <digest-item select="/Envelope/Body" scope="element" />
          </digest-element>
      <digital-signature>
 </digital_signatures>
</request>
```

Figure 17: Examples of security_template and digital_signatures elements

3.4.2. Response elements

The basic syntax of response element is the same with the request element. Thus, here we present only an example of the response element (see Figure 18).

```
<response>
    <security_template>
        <template match="/Envelope/Body/confirmation">
            <value-of-encrypted-node scope="element" pattern="response_pattern"/>
        </template>
    </security_template>

    <digital_signatures>
        <digital-signature name="response_sign" time="BEFORE">
            <signature_algorithm>
                <algorithm_definition algorithm_link="aglrsamd5"/>
            </signature_algorithm>
            <digest_function>
                <digest_definition digest_link="digest-md5"/>
            </digest_function>
            <key_information>
                <sign_key>
                    <key_definition key_link="staticKeyPrivate"/>
                </sign_key>
                <verify_key>
                    <key_definition key_link="staticKeyCert"/>
                </verify_key>
            </key_information>
            <digest-element>
                <digest-item    select="Envelope/Body" scope="element"/>
            </digest-element>
```

```
            </digital-signature>
        </digital_signatures>
</response>
```

Figure 18 : An Examples of response elements