Secure and Flexible Certificate Access in WS-Security through LDAP Component Matching

Sang Seok Lim
IBM Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598
slim@us.ibm.com

Jong Hyuk Choi
IBM Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598
jongchoi@us.ibm.com

Kurt D. Zeilenga
IBM Linux Technology Center
Redwood City, CA 94105
zeilenga@us.ibm.com

ABSTRACT

As an integral part of the Web Services Security (WS-Securi ty), directory services are used to store and access X.509 certificates. Lightweight Directory Access Protocol (LDAP) is the predominant directory access protocol for the Internet, and hence for the Web services. Values of LDAP attribute and assertion value syntaxes, though defined using ASN.1, are encoded in simple octet string formats which generally do not preserve the complete structure of the abstract values. As a result, LDAP matching rules for certificates need to be provided in a certificate-syntax specific way, while X.500 matching rules can be constructed from structured ASN.1 syntax definition. Moreover, LDAP has traditionally lacked the capability to make assertions against components of values of complex syntaxes such as X.509 certificates. The WS-Security needs to be able to locate a target X.509 certificate by matching against arbitrary certificate components in its security token references. Therefore, WS-Security requires the directory server to be prepared with all the possible matching functions for maximum flexibility. This is very cumbersome due to the lack of ASN.1 awareness in LDAP server implementations. This led to development of remedies such as the recently proposed Certificate Parsing Server (XPS). XPS extracts relevant components of the certificate and stores them in separate and searchable attributes. Due to the significant downside of these remedies, we decided to seek after an ASN.1 based Component Matching alternative in an attempt to make an LDAP directory server ASN.1 aware. With Component Matching and ASN.1 awareness, LDAP can provide WS-Security with various matching rules flexibly. In this paper, we describe our implementation of the Component Matching and ASN.1 awareness in *OpenLDAP* Software. This paper will also describe the use of the Component Matching technology in various security components of Web Services, especially in the context of WS-Security and XKMS. The experimental results show that flexible and secure certificate access can be accomplished without sacrificing performance and manageability.

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: Online Information Services-Web-based services; C.5.5 [**Computer System Implementation**]: Metrics—*Servers*

General Terms

Security, Design

Keywords

PKI, X.509 Certificate, Certificate Repository, Component Matching, LDAP

1. INTRODUCTION

Public Key Infrastructure (PKI) [6] is an integral part of the WS-Security specification [17] which provides an application-layer security framework for Web services by including security information in SOAP (Simple Object Access Protocol) [1] messages. Portions of a SOAP message can be signed in order to ensure integrity and can be encrypted to ensure confidentiality. Certificate Authority (CA) provides the most universal form of the Security Token Service to both Web service providers and requesters. The use of PKI ensures the authenticity of the security tokens themselves. The public key of a Web services entity is used to validate the integrity and authenticity of SOAP messages which were established by the corresponding private key and to encrypt the contents of the SOAP messages.

X.509 certificates [4] are often managed in the directory and are accessed through standard directory access protocols such as X.500 [11] Directory Access Protocol (DAP) [7] and Lightweight Directory Access Protocol (LDAP) [5]. In order to embed an X.509 certificate as a security token in a SOAP message, a Web service requester needs to retrieve the X.509 certificate from a CA. Alternatively, if the Web service requester includes a certificate URI as a reference to the security token, it is a Web service provider who retrieves the certificate from the CA. Even when the certificate is embedded in the SOAP message, the Web service providers may need to contact the certificate repository in order to verify whether the certificate has been revoked by retrieving a Certificate Revocation List (CRL).

LDAP is predominantly being used as the directory access protocol for the Internet. Compared to X.500 Directory

ACM Workshop on Secure Web Services, October 29,2004, Fairfax VA, USA.

Access Protocol, LDAP renders lightweight directory service by providing string-based encodings of names and attribute values (hence of assertion values), simple protocol encoding, direct mapping onto TCP/IP, and the reduced number of operations. However, these simplifications come at a price: LDAP generally does not preserve the complete structure of abstract values and the protocol and its implementations are unable to harness the full power of the underlying ASN.1 data definitions. Due to the use of specialized string-based encodings for each syntax, adding support for new syntaxes and matching rules requires new development efforts. DAP avoids these problems by the use of ASN.1 (Abstract Syntax Notation One) [9] encoding rules, in particular the Basic Encoding Rules [8].

Though these limitations were not viewed as a significant problem during LDAP's early years, it is clear that a number of directory applications, such as PKI, are significantly hampered by these limitations. For instance, in PKI, a certificate needs to be located based upon the contents of its components, such as *serialNumber*, *issuer*, *key identifiers*, and *subjectAltName* [10]. LDAP search operations do not understand the ASN.1 type of the certificate attribute and the assertion as defined in [10], because attributes and assertions in LDAP are encoded in an octet string with syntax specific encoding rules. Not only would it require exceptional effort to support matching rules such as *certificateMatch* and *certificateExactMatch* as defined in [10], that effort would have to be repeated for each matching rule introduced to match on a particular component (or set of components) of a certificate. Because of the amount of effort each server vendor must undertake to support each new rule, few new rules have been introduced to LDAP since inception. Applications had to make due with existing rules.

Foreseeing the need to be able to add new syntaxes and matching rules without requiring recoding of server implementations, the directory community engineered a number of extensions to LDAP to address these limitations. The Generic String Encoding Rules (GSER) [14] was introduced and is now used in describing and implementing new LDAP string encodings. GSER produces human readable UTF-8 [22] encoded Unicode [19] character strings and supports reuse of existing LDAP string encodings. The Component Matching [15] mechanism was also introduced to allow LDAP matching rules to be defined in terms of ASN.1. Implementations may use automated systems to (statically and dynamically) value parsing and matching functions. Additionally, Component Matching also introduces rules which allow arbitrary assertions to be matched against selected component values of complex data types such as certificates. For example, the Component Matching enables matching against the selected components of a certificate without the need to define a specific matching rule or requiring custom codes to implement that matching rules for the certificate attributes.

Though the directory community saw GSER and Component Matching as an eloquent solution to LDAP syntax and matching rule limitations, there was some concerns, as most LDAP server implementations were not ASN.1 aware, that its adoption would be slow. To fulfill immediate needs of PKI applications, another solution based upon component extraction (or "data de-aggregation") was proposed and implemented in Certificate Parsing Server (XPS) [2]. While some viewed this as being more pragmatic, as it shifted significant complexity from the server to the client and introduced a number of security and management issues, it is considered by many to be not a workable solution for PKI applications (and certainly not a workable general solution to the component matching problem). In the spring of 2004, IBM undertook an engineering effort to provide ASN.1 awareness, GSER, and component matching support in the OpenLDAP Project's Standalone LDAP Daemon (*slapd*) (the directory server component of OpenLDAP Software). We started from implementing ASN.1 awareness in the server by providing an automatic path to generate encoders and decoders for BER, DER [8], and GSER from given ASN.1 data type definitions. In addition, this ASN.1 infrastructure automatically generates the component extraction and matching routines for the individual components of an attribute type. The search filter functions were modified to support *ComponentFilter*, *ComponentAssertion*, and *ComponentReference*. As a result, it becomes possible to support complex matching such as X.509 *certificateMatch* with a *CertificateAssertion* syntax without manually defining syntax specific encodings and matching routines.

We show an example of the flexible reference method with an extension to <ds:X509Data> named, *GenericCertificateReference* which is designed after the *certificateMatch* of the X.509 recommendation. Defining *GenericCertificateReference* was easy because the Component Matching infrastructure could automatically generate the corresponding certificateMatch matching rules from the ASN.1 type specification. Web services can also use Component Matching in a URI reference of Security Token Reference by specifying the *componentFilterMatch* matching rule as the extended matching rule in the LDAP URI. Component Matching provides XKMS with the interfaces to PKI, allowing it to locate and validate a certificate flexibly. With the Component Matching enabled LDAP server as the security token service, the specification of token references will become very flexible. The Component Matching also fills the security gap in the certificate access which required special Directory Information Tree (DIT) structuring to avoid. The Component Matching and GSER facilitate the flexible and extensive use of security token references in SOAP messages.

This paper is organized as follows. Section 2 introduces WS-Security and its use of X.509 certificates as security tokens. Section3 presents an example X.509Data extension element and how Component Matching technology is used for XKMS. Section 4 introduces GSER and the Component Matching. In Section 5, we present the design of the GSER and Component Matching support in the OpenLDAP directory server. Section 6 shows experimental results of our prototype implementation of LDAP component matching in *slapd*. Section 7 concludes the paper.

2. WEB SERVICES SECURITY

2.1 SOAP and WS-Security

SOAP (Simple Object Access Protocol) is a protocol for invoking methods on servers, services, components, and objects [1]. It is a way to create widely distributed, complex computing environments using an existing Internet infras-

```
01 <?xml version=”1.0” encoding=”utf-8”?>
02 <S:Envelope xmlns:S=http://www.w3.org/2001/12/soap-envelope>
03 <S:Header>
04    <wsse:Security>
05       <wsse:SecurityTokenReference wsu: Id=“#X509Token
06          <wsse:Reference
07                         URI=“ldap://ldap.example.com/dc=example, dc=ibm???(userCertificate:componentFilterMatch:=
08                                    and:{ item:{ component “tbsCertificate.serialNumber”, rule allComponentsMatch, value 9453771 },
09                                          item:{ component “tbsCertificate.extensions.1.KeyUsage.value”, rule bitStringMatch, value ‘01000000’B }}”)>
10       </wsse:SecurityTokenReference>
11       <ds:Signature>
12          <ds:SignedInfo>
13             <ds:CanonicalizationMethod
14                          Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
15             <ds:SignatureMethod
16                          Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
17          </ds:SignedInfo>
18          <ds:SignatureValue>
19                         Hp1ZkmFZ/2kQLXDJbchm5gK..
20          </ds:SignatureValue>
21          <ds:KeyInfo>
22             <wsse:SecurityTokenReference>
23                <wsse:Reference URI="#X509Token"/>
24             </wsse:SecurityTokenReference>
25          </ds:KeyInfo>
26       </ds:Signature>
27    </wsse:Security>
28 </S:Header>
29 <S:Body>
30    <m:GetLastTradePriceResponse xmlns:m=”http://example.com”>
31        <Price>34.5</Price>
32    </m:GetLastTradePriceResponse>
33 </S:Body>
34 </S:Envelope>
```

Figure 1: Web Service Request and Response with Security Token Reference.

tructure, enabling Web service developers to build Web services by linking heterogeneous components over the Internet. For interoperability over heterogeneous platforms, it is built on top of XML and HTTP which are universally supported in most services.

WS-Security [17] is recently published as the standard for secure Web Services. It provides a set of mechanisms to help Web Services exchange secure SOAP messages. WS-Security provides a general purpose mechanism for signing and encrypting parts of SOAP messages for authenticity and confidentiality. It also provides a mechanism to associate security tokens with the SOAP messages to be secured. The security token can be cryptographically endorsed by a security authority. It can be either embedded in the SOAP message or acquired externally.

2.2 LDAP-PKI in WS-Security

In WS-Security, a security token can be either embodied in a SOAP message or acquired from an external service by following *SecurityTokenReference* [17]. Signed security tokens such as X.509 certificates are accessed through different forms of references defined in the WS-Security standard and in various security token profiles [18]. The security tokens can be retrieved and registered with the existing access methods. Either standard access protocols such as LDAP and HTTP or an ad-hoc access mechanism can be used to access security tokens. XKMS (XML Key Management Specification) [20] also provides a Web Service client which lacks direct access to the security tokens with an offloading mechanism. X.509 certificates are the most universal form of the security tokens which is well proven in the heterogeneous and distributed network, the Internet. X.509 certificates were originally designed to be stored in directories. Hence, directory access protocols such as X.500 and LDAP are the most appropriate way to access them in terms of the naming and information models and its powerful search capabilities.

In the following example scenario which illustrates an operational flow of accessing security tokens for a SOAP message shown in Figure 1, it is assumed that the Web service provider needs to contact the Security Token Service in order to fetch and verify its security token. The flow is as follows:

1. The service requester constructs a SOAP message. It contains a signature for the key elements of the message. <ds:signature> is placed in the Security Header block (lines 03 - 28 in Figure 1). In this scenario, a X.509 certificate is located in the external Security Token Service. The reference to the external token or the certificate is included in the message within <SecurityTokenReference> elements (line 05 - 10). The constructed message is sent to the Web Service.

2. The Web Service fetches KeyInfo in line 21-25. It references X.509Token, or ID which is defined in line 05-10 as SecurityTokenReference. In this case, the reference is an LDAP URI in line 07 - 09. In the URI, there is a component filter in which “*tbsCertificate.issuer*” and “*tbsCertificate.extensions.1.keyUsage.value*” refer to the corresponding components of a certificate and 9453771 and “’01000000’B” are assertion values against the components. More detailed explanation of the component filter will be described in Section 5.2.1.

```
CertificateAssertion ::= SEQUENCE {
    serialNumber                [0] CertificateSerialNumber OPTIONAL,
    issuer                      [1] Name OPTIONAL,
    subjectKeyIdentifier        [2] SubjectKeyIdentifier OPTIONAL,
    authorityKeyIdentifier      [3] AuthorityKeyIdentifier OPTIONAL,
    certificateValid            [4] Time OPTIONAL,
    privateKeyValid             [5] GeneralizedTime OPTIONAL,
    subjectPublicKeyAlgID       [6] OBJECT IDENTIFIER OPTIONAL,
    keyUsage                    [7] KeyUsage OPTIONAL,
    subjectAltName              [8] AltNameType OPTIONAL,
    policy                      [9] CertPolicySet OPTIONAL,
    pathToName                  [10] Name OPTIONAL,
    subject                     [11] Name OPTIONAL,
    nameConstraints             [12] NameConstraintsSyntax OPTIONAL
}
```

(a) CertificateAssertion XML Schema

```
<wsse:SecurityTokenReference>
   <ext:KeyIdentifier EncodingType="…#XER"
      ValueType="…#GenericCertificateFields"
         <CertificateAssertion>
             <serialNumber>9453771</serialNumber>
              <issuer>cn=ray</issuer>
              <keyUsage>Signature</keyUsage>
          </CertificateAssertion>
   <wsse:KeyIdentifier>
</wsse:SecurityTokenReference>
```

(b) XER

```
<wsse:SecurityTokenReference>
   <wsse:KeyIdentifier EncodingType="…#GSER"
      ValueType="…#GenericCertificateFields"
      { serialNumber 9453771, issuer {type cn, value ray} ,
                                          keyUsage '100000000'B }
   <wsse:KeyIdentifier>
</wsse:SecurityTokenReference>
```

(c) GSER

Figure 2: Certificate Assertion Schema Definition and GenericCertificateReference XER and GSER.

3. The Security Token Service searches for the certificate according to the LDAP URI from the Web service provider and returns the resulting certificate as a search result.

4. The Web service provider validates the <ds:Signature> in the <wsse:Security> header and processes the SOAP message to return the service result to the Web service requester.

In the above scenario, the Web service retrieves the security token stored in a directory through LDAP as a security token access method. In order to retrieve the certificate, the Web service will send an LDAP request corresponding to the reference URI. The Security Token References can have various forms of references in addition to LDAP URI. In case of X.509 certificates, they can be referenced by *X509IssuerSerial, X509SubjectName, and X509SKI*. The *<ds:X509Data>* element of the *<ds:KeyInfo>* can also be extended to represent elements from external namespaces.

2.3 LDAP Deficiencies for Certificate Access

Although WS-Security needs to be able to locate a certificate by matching against arbitrary components of a certificate, LDAP does not easily provide an universal matching mechanism to the X.509 certificate. In LDAP, attribute and assertion values are represented in octet string. No structural information is stored along with them. A brute force way to provide matching for complex attributes is to provide syntax specific matching rules. For a *X509IssuerSerial* reference for example, it is possible to write a special matching function which matches the certificate attribute against the assertion value consisting only of issuer name and serial number. Obviously, it will be too costly to define syntax specific matching rules for all possible references. In order for an LDAP server to support *X509SubjectName* and *X509SKI*, matching rules specific to the references need to be implemented. For a very flexible reference such as *CertificateMatch* as defined in X.509 recommendation [10], it would be very difficult to provide the corresponding matching rule in LDAP. Moreover, it would be difficult to cope with changes such as certificate extensions.

To address this mismatch between X.509 certificate and LDAP, the attribute extraction mechanism was recently proposed in the Certificate Parsing Server (XPS) designed by the University of Salford [3]. In XPS, all the certificate attributes are extracted and stored as simple and searchable LDAP attributes and matching is performed on the extracted attributes. Although it facilitates matching against components of a complex attribute, it can be considered as a suboptimal approach due to the following respects. First, matching is performed on the extracted attributes but not on the certificate itself. Because the contents of the extracted attributes are mutable, there is a chance of returning wrong certificates to Web services. It is strongly recommended for Web services to verify the returned certificate again. In order to minimize the security hole, the server administrator must ensure the integrity of a certificate and extracted attributes. Second, when there is more than one certificate in a directory entry, one per key usage for example, the Web services may be returned with multiple certificates even though they searched for certificates having a specific key usage. The matched values control [10] does not solve the problem, because matching is not performed on the certificate itself. It is inevitable to restrict the Directory Information Tree (DIT) structure in designing a certificate DIT to avoid an additional searching step in the Web services [2]. Third, the XPS does not facilitate matching against a complex assertion value as in X.500 directory. It is not possible to perform a flexible matching as in X.509 *certificateMatch* without making the LDAP directory server ASN.1 aware.

3. COMPONENT MATCHING

In order to overcome the mismatch between LDAP and the requirement of flexible certificate matching imposed by advanced applications like Web Services, we decided to provide 1) ASN.1-awareness and 2) Component Matching capabilities to an LDAP directory server. With the ASN.1-awareness support for the LDAP server, ASN.1 values are manifested within the server. It will be possible to use an ASN.1 value in an assertion. Hence, matching can be performed in an abstract level according to the corresponding ASN.1 type definition. With Component Matching, it becomes possible to match an assertion value against specific

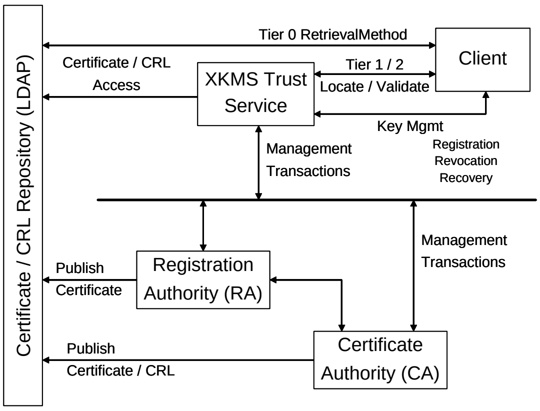


Figure 3: XKMS in PKI.

components of a complex attribute. For example, an infrastructure is provided to perform matching on an arbitrary component of an X.509 certificate, such as *serialNumber, issuer, keyUsage*, and *subjectAltName*.

Compared to the XPS approach, our approach of providing ASN.1 awareness and Component Matching has the following advantages:

1. It does not store the X.509 attributes separate from the certificate itself. Therefore, it does not increase storage requirements and does not open a potential to the compromised integrity between a certificate and its extracted attributes.
2. Matching is performed directly on the contents of the certificate but not on the associated attribute's contents. Even if there is more than one certificate in a user's entry, it can return only the matched certificate when it is used with the matched values control [10].
3. Flexible matching becomes possible because matching between an attribute value and an assertion value, both represented in ASN.1, will be provided.

3.1 Component Matching and Security Token Reference

In the X.509 token profile [18] of WS-Security, it is defined that the following three types of token references can be used:

1. Reference to a Subject Key Identifier: the value of a certificate's X.509SubjectKeyIdentifier.
2. Reference to a Security Token: either an internal or an external URI reference.
3. Reference to an Issuer and Serial Number: the certificate issuer and serial number.

Because it is defined to be extensible, any security token can also be used with appropriate schemas. It is shown in Figure 1 that the <ds:X509Data> element of <ds:KeyInfo> is used as the security token. <ds:X509Data> defined in [21] contains various references such as *X509IssuerSerial, X509SubjectName, X509SKI*, and so on. With the ASN.1 awareness and the Component Matching support in an Open LDAP directory server, these references can be used without the need of implementing syntax specific matching rules for various types of the references. It is also possible to use elements from external namespace in <ds:X509Data> for further flexibility. Figure 2 shows one such an example. Here, the *GenericCertificateReference* element from *dsext* namespace is used to provide a generic reference mechanism which implements *CertificateMatch* in the X.509 recommendation [10]. The reference consists of a sequence of certificate attributes, *serialNumber, issuer, subjectKeyIdentifier, authorityKeyIdentifier, certificateValid, privateKeyValid, subjectPublicKeyAlgID, keyUsage, subjectAltName, policy, pathToName* shown in Figure 2 (a), each of which is defined optional. By using the example reference, it would be possible to resolve a security key reference in a very flexible way. For instance, searching for a certificate having a *subjectAltName* with a specific *keyUsage* becomes possible. Figure 2 (b) shows that the reference is encoded in XML while Figure 2 (c) shows that the reference is encoded in GSER.

With the ASN.1 aware and Component Matching enabled LDAP server, the flexible reference format for an X.509 certificate can now be defined in ASN.1 with configuring the LDAP server to understand the reference. The required matching rules, encoders, and decoders for the reference type will be automatically generated and integrated to the LDAP server. This improvement in flexibility will foster the flexible use of security token references in the Web Services by making it easy to create and update references.

3.2 Component Matching and XKMS

Figure 3 illustrates a general PKI architecture which is comprised of CA, RA, and two types of end-entities. When a PKI is used for Web Services, there are two types of PKI clients: one directly accesses PKI; the other indirectly accesses it by using service proxies such as XML Key Management Specification (XKMS) [20] services which provide clients with a well defined interface to a PKI so as to hide the complexities of the underlying PKI. The XML Key Information Service Specification (X-KISS) is one of the services provided by XKMS [20]. It defines two key services: locate and validate. In the following sections, it will be presented how the Component Matching can be used in the X-KISS services with respect to certificates and CRLs access.

3.2.1 Certificate Access

Figure 4 shows an example X-KISS locate service request. In the request, there is <QueryKeyBinding> which describes how to bind this request to a desired public key. In the example, <KeyUsage> and <ds:KeyInfo> are provided to bind the request. A client using the XKMS service sends the X-KISS Locate request shown in Figure 4. In response to the request, the XKMS service needs to resolve the request and then might contact an LDAP directory server to locate the desired certificate. In the example locate request, the serial number in line 15-17 and the key usage in line 07 are supplied by the client for <QueryKeyBinding>. With Component Matching, the component filter will be constructed

```
00 <?xml version="1.0" encoding="utf-8"?>
01 <LocateRequest xmlns:ds=http://www.w3.org/2000/09/xmldsig#
02        xmlns:xenc=http://www.w3.org/2001/04/xmlenc#
03        Id="I8fc9f97052a34073312b22a69b3843b6“
04       Service=http://test.xmltrustcenter.org/XKMS
05        xmlns="http://www.w3.org/2002/03/xkms#">
06    <QueryKeyBinding>
07       <KeyUsage>Signature</KeyUsage>
08       <ds:KeyInfo>
09          <wsse:SecurityTokenReference>
10             <ds:X509Data>
11                <ds:X509IssuerSerial>
12                   <ds:X509IssuerName>
13                      o=IBM,c=US
14                   </ds:X509IssuerName>
15                   <ds:X509SerialNumber>
16                       9453771
17                   </ds:X509SerialNumber>
18                </ds:X509IssuerSerial>
19             </ds:X509Data>
20          </wsse:SecurityTokenReference>
21       </ds:KeyInfo>
22    </QueryKeyBinding>
23 </LocateRequest>
```

Figure 4: Example XKMS Locate Service Request.

from the request by the XKMS services as shown in Figure 5. The component reference “tbsCertificate.extension.*” refers to all extensions of the certificate of which *KeyUsage* is one extension. In the value, “extndID 15” is the object identifier of a *KeyUsage* and “’100000000’B” is used to check if a certain bit(the first bit for signature) is set. The corresponding component filter of Figure 5 enables the XKMS service to find a certificate of an issuer for a signature purpose only. If the client uses *GenericCertificateReference* explained in the previous section and the Component Matching is supported in an LDAP directory server, the XKMS service can use arbitrary fields of a certificate in order to construct component filters to process the locate request.

3.2.2 CRL Access

The XKMS service also provides an X-KISS Validate Service with which a client using XKMS services can check the status of a public key by sending some of <ds:KeyInfo> elements in <QueryKeyBinding> in line 08-21 of Figure 4. For instance, if it checks the validity of a X.509 certificate using CRLs, the client may send <X509Data> containing <ds:X509SerialNumber> 9453771 </X509SerialNumber> in <QueryKeyBinding>. In response to the request, the XKMS service might choose to use either CRL based validation or Online Certificate Status Protocol (OCSP) [16].

A certificate revocation list (CRL) [6] can be generated and distributed periodically by the CA, making it available on the Internet by typically using an LDAP directory server or a Web server. Because a CRL contains the list of multiple revoked certificates, it can become quite large and burdensome to transmit it over the Internet. To alleviate this problems, OCSP was proposed to provide a timelier and more efficient status mechanism. The XKMS service can speak to OCSP responders which can check the status of a certificate. The OCSP responder returns the status (either *good*, *revoked*, or *unknown*) of a requested certificate. With OCSP, the XKMS service does not need to download and scan the CRL, which are burdensome to the service.

We conceive that Component Matching can be used as an alternative to OCSP. The CRL is a sequence of pairs of a revoked certificate’s serial number and a revoked time [6]. In order to check the status of a certificate, the XKMS service needs to construct a component assertion by using the serial number of the certificate as shown in the upper item of Figure 5 and send it to the LDAP directory server. Then the server will perform Component Matching on the CRL against the assertion to find the asserted certificate in the CRL. By using the Component Matching based CRL validation, the whole CRLs is not required to be downloaded to the XKMS service. An LDAP server already has been widely used for distributing certificates and CRLs. Hence, if the server can perform validity checking over LDAP as well, it will be a very practical and efficient alternative to the OCSP which needs additional protocol layer, or and an OCSP responder.

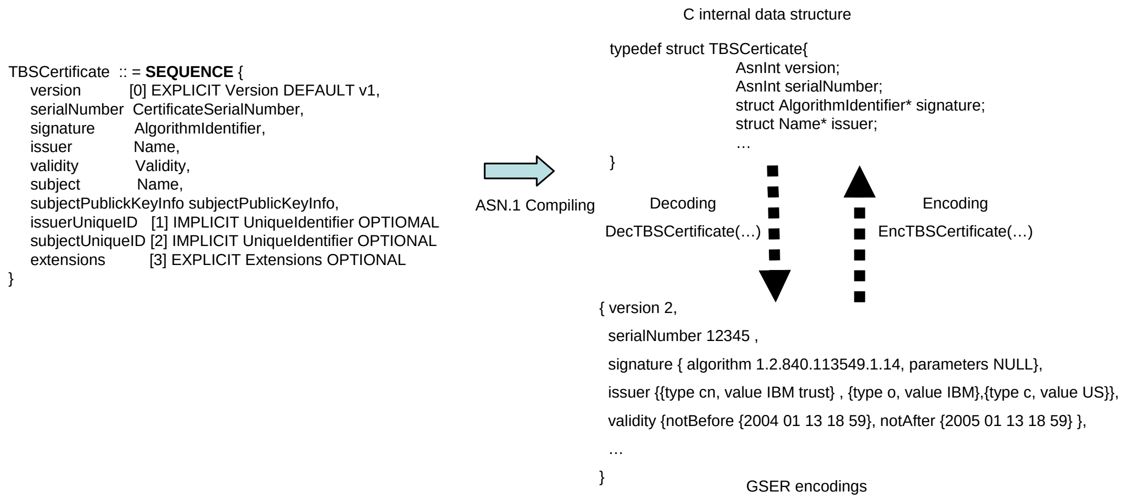
```
(userCertificate:componentFilterMatch
          := and:{
                     item:{
                              component “tbsCertificate.serialNumber”,
                              rule IntegerMatch,
                              value 9453771
                             },
                      item:{
                              component “tbsCertificate.extension.*”,
                              rule allComponentsMatch,
                              value {extnID 15, extnValue ‘100000000’B}
                             }
                  }
)
```

Figure 5: Example Component Filter.

4. COMPONENT MATCHING AND GSER

The attribute syntaxes of X.500 are defined in ASN.1 types. Basically, the types are constructed structurally from basic types to composite types. Every field of an ASN.1 type is a component. Component Matching [15] defines how to refer to a component within an attribute value by retrieving the structural information of an ASN.1 type and how to match the component against an assertion value. Matching rules are defined for the ASN.1 basic and composite types. It also defines a new assertion and filter targeted for a component. These definitions are based on ASN.1 so that they can be applied to any syntax, as long as the syntaxes are specified in ASN.1.

A native LDAP encoding does not represent the structure of an ASN.1 type. Instead, it is represented either in octet string or in binary. With the LDAP encoding, as a result, it is difficult to contain the structural information of ASN.1 type in its representation. In order to solve this problem, Legg [14] recently proposed GSER (Generic String Encoding Rules). Component matching uses GSER as its basic encoding for the component filter. GSER generates a human readable UTF-8 character string encoding of a given ASN.1 specification. It defines UTF8 string encodings at the lowest level of primitive ASN.1 types such as INTEGER, BOOLEAN, and STRING types and then it builds up more complex ASN.1 types such as SEQUENCE and SET from the lowest level. Thus, the structural information of an ASN.1 specification is maintained in encodings so that it can be recovered in the decoding process. By using GSER to store attribute values instead of the native LDAP string encoding, LDAP server will become capable of identifying



```
TBSCertificate  :: = SEQUENCE {
    version          [0] EXPLICIT Version DEFAULT v1,
    serialNumber  CertificateSerialNumber,
    signature        AlgorithmIdentifier,
    issuer             Name,
    validity           Validity,
    subject           Name,
    subjectPublickKeyInfo subjectPublicKeyInfo,
    issuerUniqueID   [1] IMPLICIT UniqueIdentifier OPTIOMAL
    subjectUniqueID [2] IMPLICIT UniqueIdentifier OPTIONAL
    extensions         [3] EXPLICIT Extensions OPTIONAL
}
```

```
typedef struct TBSCerticate{
                AsnInt version;
                AsnInt serialNumber;
                struct AlgorithmIdentifier* signature;
                struct Name* issuer;
                …
}
```

```
{ version 2,
  serialNumber 12345 ,
  signature { algorithm 1.2.840.113549.1.14, parameters NULL},
  issuer {{type cn, value IBM trust} , {type o, value IBM},{type c, value US}},
  validity {notBefore {2004 01 13 18 59}, notAfter {2005 01 13 18 59} },
  …
}
```

Figure 6: ASN.1 TBSCertificate specification, its compiler output, and example GSER encodings.

the structure of ASN.1 specification of the attribute types. Furthermore, a component filter of an LDAP request is also encoded in GSER. Hence, GSER is an essential mechanism to ASN.1 awareness and Component Matching. GSER encoding can also be used in an assertion to facilitate a very flexible and abstract matching of two ASN.1 values.

Figure 6 shows the ASN.1 type specification of TBSCertificate and its GSER encodings. TBSCertificate is defined as SEQUENCE so that there are curly braces at the beginning and at the end of its GSER encodings. It has *version*, *serialNumber* etc. as its components inside of SEQUENCE. Within the braces in the encoding, there is *version* and *2*, or its value, followed by a comma which separates the encoding of the subsequent field. GSER defines each basic type's encoding and then combines them structurally to more complex ones by using "{", "," and "}". On the other hand, a native LDAP encoding does not have any uniform rule to construct the structural information of attribute value in it.

5. COMPONENT MATCHING IMPLEMENTATION IN OPENLDAP

An ASN.1 compiler translates ASN.1 modules into C data structures representing ASN.1 and their encoding / decoding routines as illustrated in Figure 7. We extended the eSNACC ASN.1 compiler to implement the GSER backend in addition to the originally supported BER and DER [12]. It also generates component equality matching rules and component extract functions which will be discussed in the rest of the section in detail. In order to facilitate the integration of the newly defined syntaxes without the need of rebuilding the *slapd* executable, the generated data structures and routines are built into a module which can be dynamically loaded to *slapd*.

The rest of the section will provide a detailed description of the component matching in two steps. After describing how to make the OpenLDAP directory server ASN.1 aware, the description of component filter processing, aliasing, and component indexing will be presented. Then, the overall operational flow of Component Matching will be described.

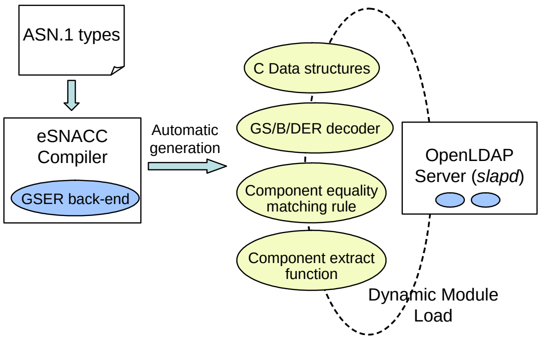


Figure 7: Input and Output of an eSNACC Compiler.

5.1 ASN.1 Awareness

5.1.1 eSNACC Compiler

We have implemented a GSER backend for the extended eSNACC compiler. GSER can be used as an LDAP encodings for newly defined attribute types. With GSER, string-based LDAP encodings can maintain the structures of their corresponding ASN.1 types. The assertion values in the component filter are also represented in GSER and the GSER decoders are used to decode them into their internal representations. Figure 6 shows the example C data structure for a TBSCertificate ASN.1 type and its corresponding GSER encodings. Generated C data structure for the ASN.1 type has data fields corresponding to components of the ASN.1 type. Once the C data structure for TBSCertificate is instantiated, it can be converted into GSER encodings by *EncTBSCertificate()* and the encodings can be decoded into the C data structure by *DecTBSCertificate()*. The eSNACC compiler also generates matching rules and extract functions automatically which will be further discussed in the next subsection.

Table 1: Attribute Aliasing Table.

Alias Attribute	Aliased Attribute	Component Reference	Matching Rule
x509certificateSerialNumber	*userCertificate*	*tbsCertificate.serialNumber*	*integerMatch*
x509certificateIssuer	*userCertificate*	*tbsCertificate.issuer*	*distinguishedNameMatch*

5.1.2 Component Representation of ASN.1 Types

A new data structure of *slapd* is needed to represent an attribute value at its component level because the original data structure for attributes does not contain the structure information of an ASN.1 type in its representation. Every field of an ASN.1 type is a component which is addressable by a component reference. In our implementation, the component data structure consists of two parts: one to store the value of the component; the other to store a component descriptor which contains information on how to encode, decode, and match the value of the component.

The data structure of a component appears as a tree which keeps the structural information of the original ASN.1 specification using nodes and arcs. Each component node of the tree not only has data values but also represents the structural information of the given ASN.1 specification by having links to subordinate nodes. In the tree, any node can be referenced by a component reference in order to perform matching on the corresponding component. Hence, we need a function to traverse the tree and locate the referenced node. The ASN.1 compiler also generates component extractor routines for this purpose.

5.1.3 Syntax and Matching Rules

An attribute is described by an attribute type in LDAP. An attribute type contains two key fields which help to define the attribute as well as rules that it must follow. The first field is a syntax which defines the data format used by the attribute type. The second field is a matching rule which is used by an LDAP server to compare an attribute value with an assertion value supplied by the LDAP search or compare operations. Attributes must include the matching rules in their definition. At least, an equality matching rule should be supported for each attribute type. From the viewpoint of an LDAP server, an ASN.1 specification defining a new attribute type requires a new syntax and its matching rule to be defined in it. To fully automate the component matching in which the composite attribute types are defined in ASN.1, we extended the *eSNACC* compiler to generate the basic equality matching rule of a given ASN.1 type, or *allComponentMatch* matching rule specified in RFC 3687 [15]. The *allComponentMatch* matching rule evaluates to true only when the value of the referenced component of the attribute and that of *ComponentAssertion* are the same. It can be implemented by performing matching from the top-most component which is identified by the component reference recursively down to the subordinate components. The generated matching function of each component can be overridden by other matching functions through a matching rule refinement table. Therefore, it is possible that a syntax developer can replace the compiler-generated matching functions with existing matching functions of *slapd* which might be more desirable. In order to support this refining mechanism, *slapd* checks if a matching function is overridden or not by looking up the refinement table, whenever it is executed.

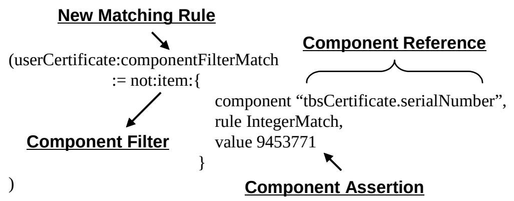


Figure 8: Example Component Filter.

5.2 Component Matching

5.2.1 Component Assertion and Filter

RFC 3687 [14] defines a new component filter as the means of referencing a component of a composite attribute and as the means of representing an assertion value for a composite attribute types. Component assertion is an assertion about the presence or the values of components within an ASN.1 value. In the component assertion, there are three key fields:

- Component Reference: specifies which component of an attribute value will be matched against an assertion value.
- Matching Rule: specifies which matching rule will be used to perform matching on the values.
- Value: An assertion value in GSER.

Component filter is an expression of component assertions, which evaluates to either *TRUE*, *FALSE*, or *Undefined* after performing matching. Figure 8 illustrates an example component filter. The component reference *“Certificate.serialNumber”* identifies one component in the Certificate attribute value. In the component reference, “.” means identifying one of components subordinate to the preceding component. In the component assertion, *rule* is followed by an *integerMatch* matching rule [11] which will be used to compare the following assertion value with the referenced component of the attribute value. The routines required to support the component filter and the component assertion were hand-coded while the routines for the component assertion values are automatically generated from a given ASN.1 type.

5.2.2 Attribute / Matching Rule Aliasing

To enable component matching, clients as well as servers need to support GSER and new component matching rules. However, the client side changes will be minimal, if at all, because the component filter can be specified by using the existing extensible matching rule mechanism of LDAPv3 and the component assertion value is represented as the text centric GSER encoding rules. In particular, the clients that

Table 2: Decoder Performance Comparison.

	d2i_X509()	ASN.1 Decoder	ASN.1 Decoder for CM
Time (usec)	32.74	40.20	72.00

accept search filters as strings require no changes to utilize component matching other than filling in the necessary component filter as the search filter. However, for those clients who have search filters hard coded in them, we propose an attribute aliasing mechanism which maps a virtual attribute type to an attribute component and a component matching rule, and a matching rule aliasing mechanism which maps a virtual matching rule to a component assertion.

Attribute aliasing registers a set of virtual attributes to an LDAP server. The virtual attributes themselves find corresponding matching rules and component references by looking up an attribute alias table. The example attribute alias table is shown in Table 1. *x509certificateSerialNumber* attribute is aliased to *userCertificate.tbsCertificate.serialNumber* with the *integerMatch* matching rule. Hence, the filter (*x509certificateSerialNumber=12345*) is considered equivalent to (*userCertificate:ComponentFilter:=item:component tbsCertificate.serialNumber, rule integerMatch, value 12345*). With an attribute aliasing, clients only have to form simple assertions to utilize component matching. A matching rule alias works in a similar way. An alias matching rule is mapped into the corresponding a component reference and a matching rule.

5.2.3 Putting It All Together

The overall flow of acquiring security tokens in Web services with Component Matching is as follows. First, the ASN.1 specification of a X.509 certificate is compiled and the generated routines are loaded into the LDAP server. Once they are loaded successfully, Component Matching on the certificate can be performed in the server. Then, the Web Service requester sends the Web Service provider a SOAP message which contains *SecurityTokenReference*. URI reference was taken as an example in the following discussion. The Web service provider fetches the *SecurityTokenReference* from the message and performs an LDAP search based on the LDAP URI in it. The search filter can contain the component assertion against the corresponding component of the certificate. Accepting a component search request, certificates in the LDAP server are decoded by the corresponding DER decoder to construct a component tree of the certificate. It has all certificate attributes as defined in the X.509 recommendation. The incoming component filter is parsed to obtain the component assertion value and the component reference by the GSER decoder for the filter. The component extractor extracts the referenced component from the component tree which is specified by the component reference in the component filter. The component assertion value is decoded by the corresponding GSER decoder for the extracted component. At this point, we have obtained the ASN.1 internal representation of both the component assertion value and the returned certificate component. Matching is performed on these two internal data structures and the matched certificate is returned to the Web Service provider.

6. PERFORMANCE CONSIDERATIONS

The benchmarking was performed with a 40,000 entry DIT. Using an LDAP benchmark in ruby scripts [13], only read operation was performed first on the entries with varying base DNs in the LDAP search requests. The read benchmark was made to test how fast data could be read from *slapd*. The machine running *slapd* has two 2.40GHz Hyperthreading Intel Xeon CPUs and 1.5Gbytes memory. The *slapd* was configured with a BDB backend. First, we measured the performance of eSNACC compiler generated DER decoders as compared to that of the openssl decoder, `d2i_X509()` function (manually written certificate parsing function). Table 2 shows how much time both decoders took to decode an example X.509 certificate generated by openssl.

`d2i_X509` took 32.74 usec to decode a certificate. The compiler generated DER decoder took 40.20 usec in case of not allocating component descriptors which is not out side of the pure decoding process. The compiler generated decoder took 7.46 usec longer as compared to `d2i_X509()`. It is because the decoder allocated memory not only for ASN.1 values but also for the component data structures of each fields of the certificate.

We also integrated the compiler generated decoders into *slapd* and performed searching by Component Matching. We measured the average time to search for an entry containing the target certificate, varying the number of concurrent clients. The search scope is base and a component filter, "(userCertificate:componentFilterMatch:=item:component "tbsCertificate.serialNumber", rule integerMatch, value 945 3771)" is used in the search. As the number of concurrent clients is increased from 1 to 8, the overall searching time of both cases were also increased from 1 msec to 6 msec approximately. When the number of concurrent clients is 1, Component Matching (CM) takes 1.2 msec and *d2i_X509* takes 1.18 msec. When the number of concurrent client become 8, CM takes 5.91 msec and *d2i_X509* takes 6 msec. The time difference between the Component Matching and `d2i_X509` is very small and can be considered to be within the experimental error bound. It is because the decoding itself is only small portion of a search operation. It only accounts for 4-7% of overall search time. We also performed subtree search and it appeared that the average searching time of CM was only slightly higher than that of *d2i_X509*. The overheads of parsing component filters can be removed by using attribute and matching rules aliases.

Overall searching time is heavily dependent on other factors such as filter parsing, indexing, and caching rather than decoding. By the experiment, It was proven that a flexible and secure way of accessing a certificate and CRL was achieved without performance degradation.

The maintenance of proper indices is critical to the search performance in the Component Matching as much as in the conventional attribute matching. In *slapd*, the attribute indexing is performed by generating a hash key value of the attribute syntax, matching rule, and the attribute value and maintains the list of IDs of those entries having the matching value in the set of attribute values of the indexed attribute. The component indexing can be specified in the same way as the attribute indexing, except that the component reference is used to specify which component of a composite attribute to be indexed. If the referenced component is a basic ASN.1 type, the indexing procedure will be the same as the attribute indexing. The indices for the referenced component are accessed through a hashed value of the corresponding syntax, matching rule, and value in the index file for the referenced component of the composite attribute. In OpenLDAP, the indexing of the composite component is centered on the GSER encodings of the component value. The hash key of a component value is generated from its GSER encodings together with its syntax and matching rule. For the SET and SET OF constructed types, it is required to canonicalize the order of the elements in the GSER encodings before generating the hashed key value. For <all> component reference of SET OF and SEQUENCE OF. it is needed to union the indices for each value element of SET OF and SEQUENCE OF.

7. CONCLUSION

WS-Security is a key protocol for securing SOAP message exchanges. It defines token references in order to acquire security tokens internally and externally. An X.509 certificate is an universally used security token and is usually stored and retrieved via LDAP. The X.509 certificate profile of WS-Security and XML Signature standards provides a rich set of reference methods for the X.509 certificate in a flexible and extensible way. Component Matching and ASN.1 awareness in LDAP are very effective mechanisms to cope with this flexibility and extensibility.

This paper presented the design of the Component Matching in the OpenLDAP directory server in the context of WS-Security. It is described the first implementation of the component matching and ASN.1 awareness in a pure LDAP based directory server. In order to provide a fully automated infrastructure, we extended an ASN.1 compiler to automatically generate the component extraction / matching routines and encoders / decoders of BER, DER, and GSER for a given ASN.1 type. To make the server understand an incoming *ComponentFilter* in GSER format, the search filter processing was modified to support *ComponentFilter*, *ComponentAssertion*, and *ComponentReference* as well. For backward compatibility in the case where the search filters are hard coded to client applications, we also provide an alias mechanism for the attribute names and the matching rules. According to the experiments, the performance of the prototype system which integrated compiler-generated decoders was proven to be competitive to that of the hand-optimized ones. We conclude from the experiment that flexible and secure certificate access can be achieved without losing performance and manageability.

8. REFERENCES

[1] D. Box and D. Ehne. Simple object access protocol (SOAP). W3C Note, May 2000.

[2] D. W. Chadwick. Deficiencies in LDAP when used to support PKI. *Comm. of the ACM*, 46(3), March 2003.

[3] D. W. Chadwick, E. Ball, and M. Sahalayev. Modifying LDAP to support x.509-based PKIs. In *17th Annual IFIP WG 11.3 Working Conference on Database and Applications Security*, August 2003.

[4] W. Ford and D. Solo. Internet x.509 public key infrastructure certificate and certificate revocation list (CRL) profile. RFC 3280, 2002.

[5] J. Hodges, R. Morgan, and M. Wahl. Lightweight directory access protocol (v3): Technical specification. RFC 3377, September 2002.

[6] R. Housley, W. Ford, W. Polk, and D. Solo. Internet X.509 public key infrastructure certificate and CRL profile. RFC 2459, January 1999.

[7] ITU-T Rec. X.511, The directory: Abstract service definition, 1993.

[8] ITU-T Rec. X.690, ASN.1 encoding rules: Specification of basic encoding rules (BER), canonical encoding rules (CER), and distinguished encoding rules (DER), 1994.

[9] ITU-T Rec. X.680, Abstract syntax notation one (ASN.1): Specification of basic notation, December 1997.

[10] ITU-T Rec. X.509, The directory: Public-key and attribute certificate frameworks, March 2000.

[11] ITU-T Rec. X.500, The directory: Overview of concepts, models and service, February 2001.

[12] R. Joop. Snacc 1.2rj. http://www.fokus.gmd.de/ovma/freeware/snacc/entry.html.

[13] A. Krennmair and R. Lischka. Testing OpenLDAP server, March 2004.

[14] S. Legg. Generic string encoding rules. RFC 3641, October 2003.

[15] S. Legg. X.500 and LDAP component matching rules. RFC 3687, February 2004.

[16] M. Myers, R.Ankney, A.Malpani, and C.Adams. Internet X.509 public key infrastructure online certificate status protocol - OCSP. RFC 2560, June 1999.

[17] OASIS. Web services security: SOAP message security 1.0 (WS-Security 2004). OASIS Standard 200401, March 2004.

[18] OASIS. Web services security: X.509 certificate token profile. OASIS Standard 200401, January 2004.

[19] The Unicode Consortium. *The Unicode Standard, Version 4.0.* Addison-Wesley, Boston, 2003.

[20] W3C. XML key management specification (XKMS). W3C Standard, March 2001.

[21] W3C. XML - signature syntax and processing. W3C Standard, February 2002.

[22] F. Yergeau. UTF-8, a transformation format of ISO 10646. RFC 3629, November 2003.