Design and Implementation of LDAP Component Matching for Flexible and Secure Certificate Access in PKI

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
Carson City, NV
zeilenga@us.ibm.com

Abstract

Lightweight Directory Access Protocol (LDAP) is the predominant Internet directory access protocol and hence so is its use in the Public Key Infrastructure (PKI). This paper presents the design and implementation of LDAP component matching which enhances flexibility and security of the LDAP directory service when it is used for the PKI certificate repositories. The component matching together with the prerequisite ASN.1 awareness enables matching against arbitrary components of certificates and enables matching of composite values at the abstraction layer of the underlying ASN.1 type definition. This allows searching for certificates with matching components without the need of providing syntax specific parsing and matching routines (flexibility), without the need of extracting the certificate components and storing them into separate attributes which become searchable but mutable (security), and without the need of restructuring Directory Information Tree (DIT) to support multiple certificates per subject (manageability and performance). In this paper, we describe the architecture, key data structures, and the proposed methods of enhancing interoperability and performance of our component matching implementation in the OpenLDAP open source directory software suite. We also propose the use of component matching in on-line certificate validation and in Web services security. Through performance evaluation of the OpenLDAP component matching, we show that our LDAP component matching implementation exhibits the same or higher performance compared to the previous approaches.

Keywords

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

1 Introduction

The certificate repository in Public Key Infrastructure (PKI) is a means of distributing certificates and Certificate Revocation Lists (CRL) to end entities. It stores certificates and CRLs and provides efficient access methods to them by harnessing storage means with communication mechanisms. The directory technology stands out as the most befitting approach to implementing certificate repositories because the X.509 [14] PKI has been standardized in the context of the X.500 recommendations as the public key based authentication framework on the X.500 directory.

Lightweight Directory Access Protocol (LDAP) [10] renders lightweight directory service by providing direct mapping onto TCP/IP, simple protocol encoding, reduced number of operations, and string-based encoding of names and attribute values (hence of assertion values). However, these simplifications come at a price. Because the string-based encoding in LDAP generally does not carry the complete structure of abstract values, adding support for new syntaxes and matching rules requires ad-hoc developments of syntax parsing and matching routines. X.500 protocols, on the other hand, avoid this problem by use of ASN.1 (Abstract Syntax Notation One) [13] encoding rules, in particular, the Basic Encoding Rules [12].

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 serial number, issuer name, subject name, and key usage [14]. LDAP search operations do not understand ASN.1 types in the definition of the certificate attribute and assertion [14], because attributes and assertions in LDAP are encoded in octet string with syntax specific encoding rules. Not only would it require exceptional effort to support matching rules such as *certificateExactMatch* and *certificateMatch* as defined in [14], 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 large amount of effort each server vendor must undertake to support each new rule, few new rules have been introduced to LDAP since its inception. Applications had to make due with existing rules.

Foreseeing the need to be able to add new syntax 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) [17] was introduced to be used in describing and implementing new LDAP string encodings. GSER produces human readable UTF-8 [32] encoded Unicode [28] character string which preserves the complete structure of the underlying ASN.1 type and supports reuse of the existing LDAP string encodings. Provided that an LDAP server is ASN.1 aware, i.e. it can parse values in ASN.1 encoding rules into its internal representation of ASN.1 value and can perform matching in that abstraction layer, it is possible to support matching of arbitrary types without needing ad-hoc developments of parsing and matching routines.

The component matching [18] mechanism was also introduced to allow LDAP matching rules to be defined in terms of ASN.1. It introduces rules which allow arbitrary assertions to be made against selected components values of complex data types such as certificates. For example, the component matching enables matching against the selected components of certificates without the need to define a certificate component specific matching rule and without requiring custom code to implement that matching rule for the certificate attributes.

Though the directory community saw GSER and component matching as an eloquent solution to the LDAP syntax and matching rule limitations, there were 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 attribute extraction (or “data de-aggregation”) has been being utilized as a practical remedy. The attribute extraction method decomposes a certificate into individual components and stores them into separate, searchable attributes. Certificate Parsing Server (XPS) [2] automates the attribute extraction process. Although this approach has filled the interoperability gap between LDAP and PKI, it is considered to be not a workable solution for PKI applications (and certainly not a workable general solution to the component matching problem), because it introduced a number of security and management issues.

In the spring of 2004, IBM undertook an engineering effort to provide ASN.1 awareness (with GSER, BER, DER support) and component matching functionality for the OpenLDAP Project’s Standalone LDAP Daemon (*slapd*), the directory server component of OpenLDAP Software [26]. To our knowledge, this is the first implementation of the component matching technology in a pure LDAP directory server (second to the View500 [29] directory server from eB2Bcom [8] which is X.500 based). This paper presents a detailed and comprehensive description of the design and implementation of the LDAP component matching for improved PKI support, extending our previous work [19] which had described component matching in the context of WS-Security [24]. Another contribution of this paper is that it proposes key mechanisms to improve performance and interoperability – attribute / matching rule aliasing, component indexing, and selective component caching. This paper will also present a preliminary performance evaluation result which convinces us that the performance of component matching is on par with or better than those of the syntax specific parsing and attribute extraction approaches if the optimization mechanisms proposed in this paper are used. This in fact provides a strong evidential answer to the debate in the PKI standardization community on whether the component matching technology can be implemented in LDAP directory servers timely and efficiently. This paper also discusses on the possibility of using the component matching for CRL in order to support on-line certificate status checking using LDAP. It also discusses on the feasibility of using LDAP component matching for PKI in Web services security.

This paper is organized as follows. Section 2 introduces the interoperation of LDAP and PKI and describes the deficiencies of LDAP when it is used for PKI. Section 3 describes the component matching technology and its use in PKI enabling secure and flexible certificate access. It also discusses on the possibility of certificate validation against CRL using LDAP component matching. Section 4 introduces GSER (Generic String Encoding Rules) which facilitates the ASN.1 awareness in LDAP when it represents the attribute and assertion values. In Section 5, we present the design and implementation of the ASN.1 awareness and the component matching in the OpenLDAP directory server. Section 6 demonstrates the application of the component matching for PKI to the security of Web services. Section 7 shows experimental results of our prototype implementation of the LDAP component matching and proves that the component matching can be accomplished without any loss in performance. Section 8 concludes the paper.

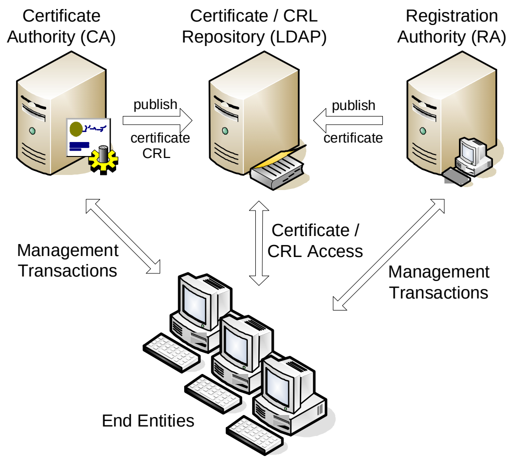


Figure 1. The Architecture of Public Key Infrastructure.

2 LDAP in PKI

2.1 LDAP Certificate Repository

X.509 certificates and CRLs are commonly distributed by the certificate repositories. LDAP directories are the most versatile mechanism of implementing the certificate repositories. Figure 1 illustrates the conceptual interoperation of four entities in PKI. In the public key registration phase, an end entity sends its identity as well as its public key to a Registration Authority (RA). If the identity is validated by the RA, the Certificate Authority (CA) will publish the end entity’s certificate, storing it in the LDAP directory. After that, the published certificate can be retrieved by any properly authenticated LDAP client. If the issued certificate is revoked by any reason, the CA is responsible for revoking the certificate by publishing CRLs to the LDAP directory. LDAP directories serve as the central place where the end entities not only can download certificates of others in order to send encrypted messages or verify digital signatures but also can be informed of the latest certificate revocation information by downloading CRLs.

2.2 Deficiencies of LDAP Certificate Access

An end entity should be able to send a request to the LDAP certificate repository searching for a certificate having matched values in specific components of the certificate. As a principle example, when it wants to retrieve the certificate having a specific serial number and issued by a specific CA, it will send an assertion against *serialNumber* and *issuer* components as specified in *certificateExactMatch* of X.509 [14]. However, the need for matching is not limited only to these two certificate components. An end entity may want to search for certificates which belong to a subject. It may also want to restrict the scope of the search for the subject’s certificates to those having a specific key usage, e.g. *nonRepudiation*, by using the *keyUsage* certificate extension. Because LDAP stores attribute and assertion values in LDAP-specific octet strings which do not generally preserve structural information of the underlying ASN.1 types, however, it is far from trivial to provide this component level matching in a generic and flexible way.

X.500 [15] satisfies this demand for component level matching by allowing matching to be defined at the ASN.1 layer. For instance, [14] defines *certificateExactMatch* and *certificateMatch* matching

```
(userCertificate:certificateExactMatch:=12345$o=IBM,c=US)
```

(a) Syntax Specific Parsing.

```
(&(x509SerialNumber=12345)(x509KeyUsage=010000000))
```

(b) Attribute Extraction.

```
(userCertificate:componentFilterMatch:=
  and:{
    item:{
      component "toBeSigned.subject",
      rule distinguishedNameMatch,
      value "cn=John Doe,o=IBM,c=US"
    }
    item:{
      component "toBeSigned.extension.*.
                          extnValue.(2.5.29.15)",
      rule bitStringMatch,
      value ‘010000000’B
    }
  }
)
```

(c) Component Matching.

Figure 2. Three LDAP Certificate Access Methods.

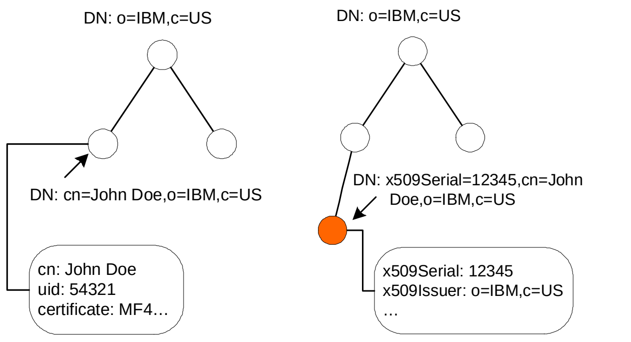
rules by specifying them in ASN.1 data type representations. The use of ASN.1 specifications is beneficial in the following respects: 1) parsing and matching can be automatically accomplished from the given ASN.1 type specification without providing ad-hoc routines; 2) simple but powerful matching rules are derivable from the strong expressive power of ASN.1, as exemplified in the use of *OPTIOANL* in *certificateMatch*; 3) new matching rules can be easily provided by specifying them in ASN.1.

The rest of this section explains how the current workarounds try to provide solution to the above mentioned interoperability gap between LDAP and PKI and introduces the component matching approach focusing on its advantages over the earlier workarounds.

2.3 LDAP Certificate Access Methods

2.3.1 Syntax Specific Parsing

A brute force approach to providing matching for arbitrary components of a certificate against an assertion is to provide certificate-syntax specific matching rules. For example, it is possible to manually write a special matching routine that matches the certificate attribute against the assertion value consisting only of *serialNumber* and *issuer* to implement *certificateExactMatch* which, in case of X.500 [15], is meant to be derived automatically from its ASN.1 specification [14]. In OpenLDAP, *certificateExactMatch* is implemented by using certificate decoding libraries provided by OpenSSL [27]. Figure 2 (a) shows an example filter in which the predetermined token '$' is used to separate the serial number 12345 and the issuer distinguished name `o=IBM,c=US`. The server can recognize the serial number and issuer name by reading two strings separated by ‘$’. The downside of this approach is obvious. It is too costly to define syntax specific matching rules for all possible components and their combinations. It is also difficult to cope with the extension mechanisms such as a certificate and CRL extensions.



(a) DIT. (b) DIT of Attribute Extraction.

Figure 3. Example Directory Information Tree (DIT).

2.3.2 Attribute Extraction

To address these deficiencies, Klasen and Gietz [22] proposed an alternative solution, based on a practical workaround that PKI administrators have been using. A set of attributes are extracted from the certificate and stored as simple, searchable attribute together with the certificate in a newly created entry which is subordinate to the original one. For this purpose, they defined a set of 30 attributes [22] for the X.509 certificate. Matching is performed on the extracted attributes. The example DIT with extracted attributes is illustrated in Figure 3. DIT (a) in Figure 3 consists of person entries under the base `o=IBM,c=US` each of which contains a certificate attribute. After attributes are extracted, the person entry will have a new subordinate entry whose DN (Distinguished Name) becomes `x509Serial=12345,cn=John Doe,o=IBM,c=US`. The attribute extraction mechanism not only makes the end entity's view of a DIT different from the CA's who published the certificates but also doubles the number of entries at minimum.

With the attribute extraction mechanism, performing matching against components is identical to performing matching against attributes as depicted in Figure 2 (b). Although attribute extraction facilitates matching against components of a complex attribute, it can be considered as a suboptimal approach in the following respects. First, matching is performed on the extracted attributes, not on the certificate itself. Because the contents of the extracted attributes are mutable, there is non-zero chance of returning a wrong certificate to a client if the extracted attributes were maliciously forged. It is strongly recommended for the client to verify the returned certificate again to ensure strong security. In the server side, on the other hand, the server administrator must ensure the integrity of a certificate and the corresponding extracted attributes in order to minimize this security vulnerability. Second, when there are more than one certificates in a directory entry, one per key usage for example, it is not possible to pinpoint and return the certificate having the matching component (i.e. key usage for example again) since the searched-for attribute is different from the to-be-returned attribute. The matched value control [4] does not solve this problem, because an LDAP attribute is set of values, not sequence of values. Therefore, it is inevitable to transform the DIT structure in designing a certificate DIT to avoid the need for an additional searching step in the client [3]. Third, the attribute extraction does not facilitate matching against a composite assertion value as in X.500. It is not possible to support a flexible matching as in X.509 *certificateMatch* without making LDAP directory servers ASN.1 aware.

2.3.3 Certificate Parsing Server

An automatic attribute extraction mechanism was recently proposed. The Certificate Parsing Server (XPS) designed by the University of Salford [3] extends the OpenLDAP directory server in order to automatically extract and store the certificate components. Although it can significantly relieve the PKI administrator's burden, it does not improve the attribute extraction mechanism not to suffer from the three disadvantages of described above.

2.3.4 Component Matching

Component matching is recently published in RFC 3687 [18] in an effort to provide a complete solution to the LDAP - PKI interoperability problem. All attribute syntaxes of X.500 and LDAP are originally described by ASN.1 type specifications [15, 10]. However, LDAP uses LDAP specific encodings which does not generally preserves the structural information in the original ASN.1 type, instead of relying on an ASN.1 encodings. The component matching defines a generic way of enabling matching user selected components of an attribute value by introducing a new notion of component assertion, component filter, and matching rules for components. With component matching, it becomes possible to perform matching of an assertion value against a specific component of a composite attribute value. For example, infrastructure is provided to perform matching against an arbitrary component of an X.509 certificate, such as *serialNumber*, *issuer*, *subject*, and *keyUsage*. Technical details of the component matching will be explained in the following sections. Compared to the attribute extraction approach, component matching has the following advantages:

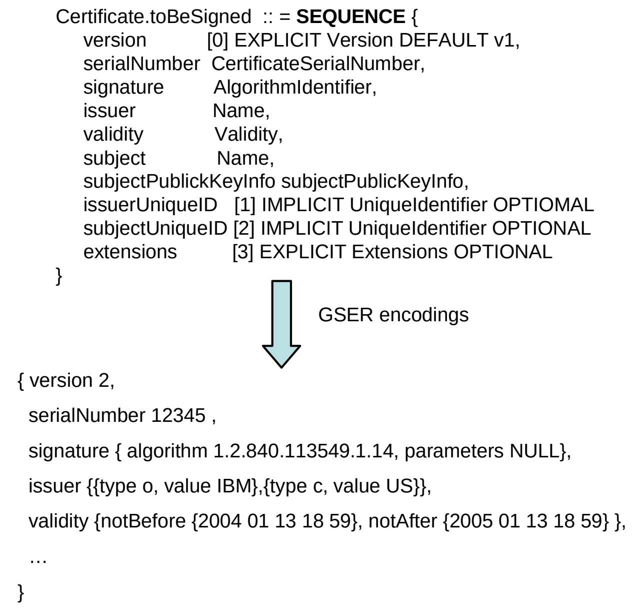
1. It does not extract and store certificate components separate from the certificates themselves. Therefore, it does not increase storage requirements and does not open a potential to the compromised integrity between a certificate and its extracted attributes.
2. Matching is performed not on the extracted attributes' contents but directly on the certificate's content. It can return only the matched certificate out of multiple certificates in a user's entry if it is used in conjunction with the matched values control [4].
3. It becomes convenient to provide a complex matching flexibly because matching between attribute and assertion values is performed at the ASN.1 layer.

3 Component Matching for PKI

3.1 Component Matching and Its Usage

The attribute syntaxes of X.500 are defined in ASN.1 types. The type is structurally constructed from basic types to composite types just like C struct definition. Every field of an ASN.1 type is a component. Based on ASN.1 types, component matching [18] defines how to refer to a component within an attribute value and how to match the referred 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 tailored for a component, or each field of the ASN.1 type. These definitions are based on ASN.1 so that they can be applied to any complex syntax, as long as it is specified in ASN.1.

The search filter for component matching is a matching rule assertion [10] whose matching rule is *componentFilterMatch* and whose



```
Certificate.toBeSigned  :: = 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
}
```

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

Figure 4. Certificate ASN.1 Specification and GSER Encoding.

assertion value is a component filter. The search filter for component matching consists of three parts.

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

3.2 Certificate Access

Component matching, as introduced in Section 2.3.4, enables matching of an assertion value against a specific component of a certificate such as *serialNumber*, *issuer*, *subject*, and *keyUsage*. If a client receives a reference to a certificate consisting of the name of the issuing CA and its serial number, the client has to search for the certificate having matching *issuer* and *serialNumber* components in a certificate repository. Alternatively, a client may want to retrieve communicating party's certificates, not all of them, but only the ones for the non-repudiation purpose, by matching its distinguished name and key usage against the *subject* and *keyUsage* components of the certificate. For instance, the client can make an LDAP search request having the search filter illustrated in Figure 2 (c) to search for the certificates of `cn=John Doe,o=IBM,c=US` that are arranged to be used for non-repudiation. The example component filter of Figure 2 (c) contains two component assertions, one for *subject* and the other for *keyUsage*. The component references to these components begin with *toBeSigned* which is a sequence of certificate components to be digitally signed for immutability. *toBeSigned.serialNumber* refers to the *serialNumber* component of a certificate while *toBeSigned.extension.*.extnValue.(2.5.29.15)* refers to any extension of *keyUsage* type. In the latter example, *(2.5.29.15)* is the object identifier (OID) of the *keyUsage* extension. It is contained in *OCTET STRING* of the *extnValue* of any components of *extensions* certificate component. In other words, the reference means identifying all *keyUsage* extension components.

The component matching rule specifies which matching rules will

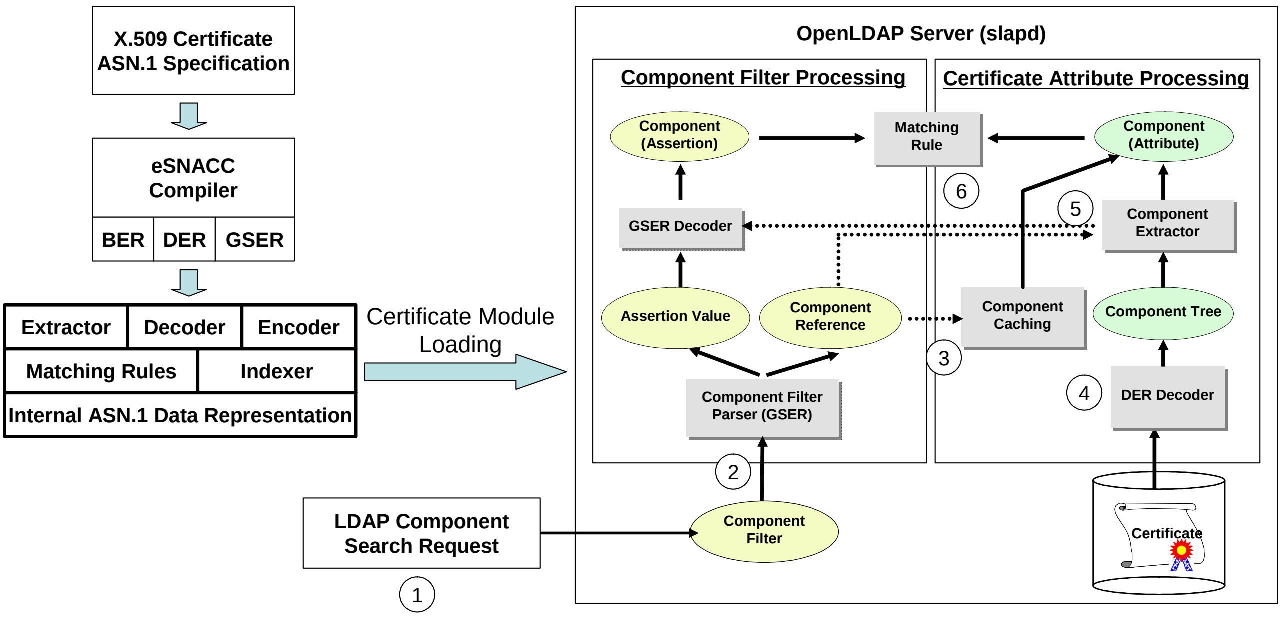


Figure 5. Architecture of Component Matching in OpenLDAP.

be used to perform matching a certificate's component against an assertion value. Either existing matching rules or newly defined matching rules can be used as the component matching rules. Matching rules for composite types can be provided by combining those of their subordinate types. *allComponentsMatch* implements matching at the ASN.1 layer whereas derived matching rules can be defined to override it with specific syntaxes.

RFC 3687 [18] defines which matching rules can be applied to each of the ASN.1 types. In the example component filter in Figure 2 (c), *distinguishedNameMatch* is used for *subject* and *bitStringMatch* is used for *keyUsage*. The component assertion value is a GSER-encoded value asserted against the component selected by the component reference. In Figure 2 (c), the value `cn=John Doe,o=IBM,c=US` is the GSER encoded ASN.1 *UTF8 STRING* and the value '010000000'B is the GSER encoded ASN.1 *BIT STRING* value.

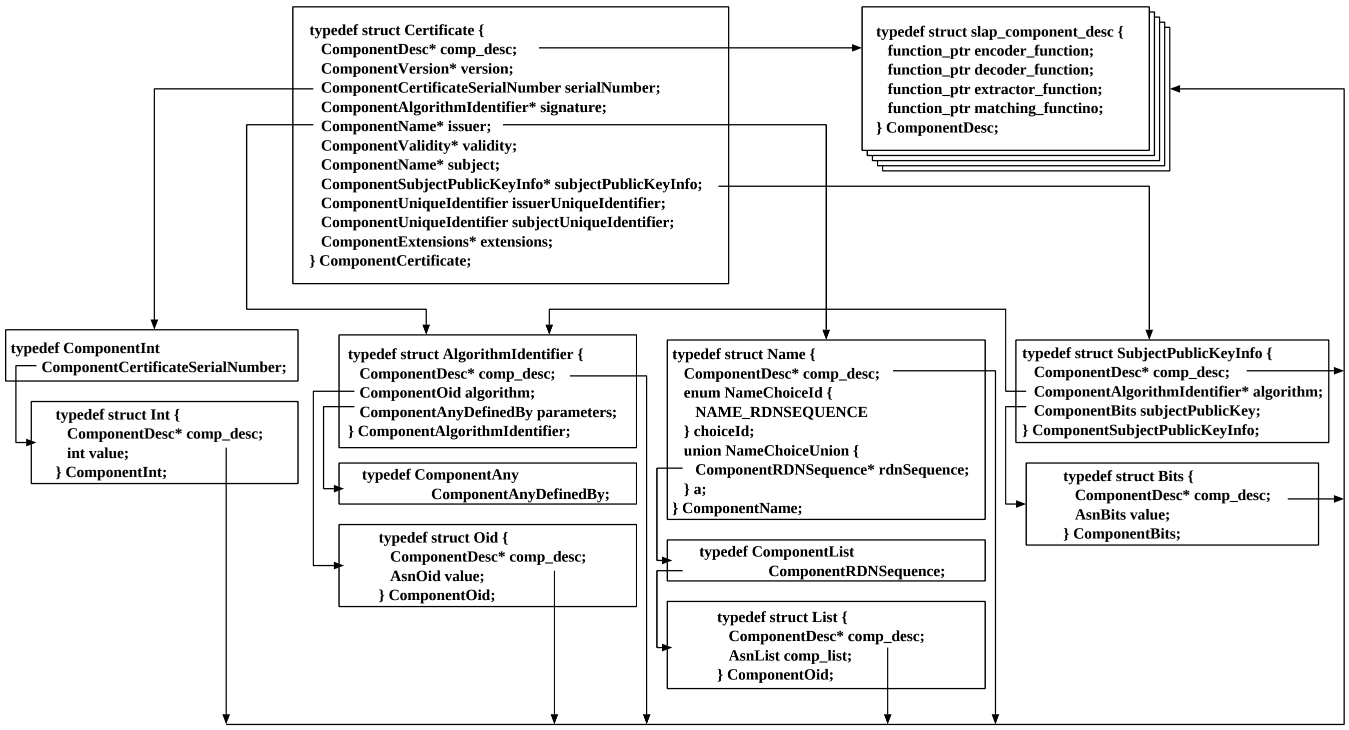
The client sends a search request containing the component filter to a component matching enabled LDAP directory server. In response, the client will be returned with those entries having the matching certificate if there is any. After checking the authenticity and integrity of the returned certificate, the client can extract the public key out of the certificate for further use.

3.3 Certificate Revocation List (CRL) Access

Although certificates were valid at the time when they were issued, CA must revoke certificates occasionally because the key pair for a certificate can be compromised or the binding between an identity and a certificate become invalid. As a result, a certificate should be validated when it is used. Otherwise, the client might incur not only incomplete, but also insecure transactions. The CRL mechanism [11] provides a means of performing validation of certificates against periodically published list of revoked certificates, or Certificate Revocation List (CRL). A CRL is periodically generated by the CA and is made available through certificate repositories such as LDAP directories. Although the CRL mechanism has been carefully revised to reduce the CRL download traffic which can degrade the scalability of PKI significantly, it still requires the end entities to store CRLs in its local storage in order to facilitate efficient and off-line operation. On the other hand, on-line certificate validation protocols are also proposed in order to cope with the on-line end entities which need more fresh information on certificate validity. Because the on-line end entities need not store the certificate status information in its storage, the on-line protocols also eliminate the requirement for the hefty local certificate storage. Online Certificate Status Protocol (OCSP) [21] and Simple Certificate Validation Protocol (SCVP) [9] are two examples of the on-line certificate validation protocols.

We conceive that component matching enabled LDAP can also be used as an on-line certificate validation protocol. CRL is a sequence of pairs of a revoked certificate's serial number and revoked time [11]. In order to check status of the certificate, the client needs to make a component assertion against the serial number of the certificate under scrutiny. Then, the LDAP server will perform component matching on the CRL against the assertion to find the asserted serial number in the CRL. This is possible with component matching, since the LDAP server understands the structure of the CRL and is able to compare specific components of the CRL against the component assertion. In the attribute extraction approach, however, the serial numbers of all the elements of the revoked certificate list must be extracted as separate attributes which need to be stored in the individual subordinate entries. This not only increases the amount of storage and increases the complexity of managing directory significantly, but also makes the server vulnerable to malicious attacks as explained in Section 2.3.4.

With component matching, the whole CRL does not necessarily have to be downloaded to the client and scanned by the client so as to save the network bandwidth and the client's computing power significantly. Especially for the clients which have limited computing power and low bandwidth such as mobile devices, component matching will be very efficient solution for the client to access PKI. Furthermore, an LDAP server already has been widely used



```
typedef struct Certificate {
   ComponentDesc* comp_desc;
   ComponentVersion* version;
   ComponentCertificateSerialNumber serialNumber;
   ComponentAlgorithmIdentifier* signature;
   ComponentName* issuer;
   ComponentValidity* validity;
   ComponentName* subject;
   ComponentSubjectPublicKeyInfo* subjectPublicKeyInfo;
   ComponentUniqueIdentifier issuerUniqueIdentifier;
   ComponentUniqueIdentifier subjectUniqueIdentifier;
   ComponentExtensions* extensions;
} ComponentCertificate;

typedef struct slap_component_desc {
   function_ptr encoder_function;
   function_ptr decoder_function;
   function_ptr extractor_function;
   function_ptr matching_functino;
} ComponentDesc;

typedef ComponentInt
   ComponentCertificateSerialNumber;

typedef struct Int {
   ComponentDesc* comp_desc;
   int value;
} ComponentInt;

typedef struct AlgorithmIdentifier {
   ComponentDesc* comp_desc;
   ComponentOid algorithm;
   ComponentAnyDefinedBy parameters;
} ComponentAlgorithmIdentifier;

typedef ComponentAny
            ComponentAnyDefinedBy;

typedef struct Oid {
   ComponentDesc* comp_desc;
   AsnOid value;
} ComponentOid;

typedef struct Name {
   ComponentDesc* comp_desc;
   enum NameChoiceId {
      NAME_RDNSEQUENCE
   } choiceId;
   union NameChoiceUnion {
      ComponentRDNSequence* rdnSequence;
   } a;
} ComponentName;

typedef ComponentList
            ComponentRDNSequence;

typedef struct List {
   ComponentDesc* comp_desc;
   AsnList comp_list;
} ComponentOid;

typedef struct SubjectPublicKeyInfo {
   ComponentDesc* comp_desc;
   ComponentAlgorithmIdentifier* algorithm;
   ComponentBits subjectPublicKey;
} ComponentSubjectPublicKeyInfo;

typedef struct Bits {
   ComponentDesc* comp_desc;
   AsnBits value;
} ComponentBits;
```

Figure 6. Certificate Component Tree.

for distributing CRLs and certificates. Hence, if the server can perform on-line validity checking over the CRL as well, it will be very practical and efficient alternative to OCSP which needs additional software, or an OCSP responder.

In [6], we also propose to structure the internal representation of CRL as an authenticated data structure such as the Certificate Revocation Tree (CRT) [16] and the authenticated 2-3 tree [23]. Together with the component matching, it makes certificate validation result from an LDAP server unforgeable while not requiring to have the LDAP server as a trusted entity nor to sign every LDAP response on the fly as in OCSP.

4 GSER (Generic String Encoding Rules)

A native LDAP encoding does not represent structure of an ASN.1 type. Instead, it is 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, S. Legg [17] recently proposed GSER (Generic String Encoding Rules). Component matching uses GSER as its basic encoding for the component assertion value. GSER generates a human readable UTF-8 character string encoding of a given ASN.1 specification with predetermined set of characters to keep the structure such as '{', '}', and ','. It defines UTF8 string encodings at the lowest level of the ASN.1 built-in 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 by using the characters. Thus, the structural information of an ASN.1 specification is maintained in encodings so that it can be recovered in the decoding process easily. By using GSER to store attribute values instead of the native LDAP encoding, an LDAP server is capable of identifying the structure of ASN.1 specification of the attribute. Furthermore, the component filter itself is also encoded in GSER. Hence, GSER is an essential mechanism to ASN.1 awareness and component matching.

Figure 4 shows the ASN.1 type specification of a *toBeSigned* and its GSER encodings. The certificate is 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, there is *version* and *2*, or its value, followed by comma which separates the subsequent field encoding. GSER defines each basic type's encoding and then combines them structurally to a more complex one by using "{", "," and "}". On the other hand, a native LDAP encoding does not have any systematic rule to construct the structure information of attribute value in it.

5 Component Matching Implementation in OpenLDAP

The overall conceptual architecture of the component matching in the OpenLDAP *slapd* directory server is illustrated in Figure 5. Given the ASN.1 specification of the X.509 certificate as an input, the extended *eSNACC* ASN.1 compiler generates the *slapd* internal data representation of the X.509 certificate and their encoding / decoding routines. We extended the *eSNACC* ASN.1 compiler [7] to support GSER in addition to the originally supported BER and DER [7]. It also generates component equality matching rules, component extract functions, and component indexer functions which will be discussed later in this 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 into *slapd*. The overall flows of LDAP component search is explained as follows;

1. On the client side, a search for components of X.509 certificate is initiated by the inclusion of the *ComponentFilter* in the filter of the search request. A *ComponentFilter* consists of *ComponentAssertion*s each of which is in turn comprised of *component*, *rule*, and *value*.
2. On the server side, whenever *slapd* detects that the search request contains *ComponentFilter*, it parses the incoming component filter to obtain assertion values and component references. The assertion values are also converted to the ASN.1 internal representation by the GSER decoder.
3. Retrieve the entry cache to see if the target certificate's decoded component tree is cached. If so, skip the following steps upto the step 6.
4. If it is not cached, by using an appropriate ASN.1 decoder, *slapd* decodes the *certificate* attribute into the component tree, the ASN.1 internal representation, when loading the candidate entries containing certificate for matching. Because a certificate is DER encoded, DER decoder is used to construct a certificate's component tree.
5. The component reference is fed into the component extractor to obtain the component subtree which is referenced by *component reference* out of the attribute component tree.
6. The assertion component and the extracted attribute component are then matched together by the matching rule corresponding to the component which is generated also by the extended eSNACC compiler. Matching is performed at the abstract level using the internal ASN.1 data representation.

The rest of the section provide detailed description of the component matching in two steps. After first describing how to make the OpenLDAP directory server ASN.1 aware in detail, component filter processing, aliasing, component indexing, and component caching will be described.

5.1 ASN.1 Awareness

5.1.1 eSNACC Compiler

Figure 6 shows the internal data representation of the *toBeSigned* ASN.1 type along with the representations of some of its key components. The data structures for this ASN.1 data representation are automatically generated by the eSNACC compiler from the given ASN.1 specification of *toBeSigned*. The generated data structure for the *toBeSigned* has data fields corresponding to components of the *toBeSigned* ASN.1 type. Once the internal data structure for *toBeSigned* is instantiated, it can be converted to DER by `DEnctoBeSigned()` and back to the internal representation by `DDectoBeSigned()`.

Component matching can be performed for any composite attributes which are encoded as one of the ASN.1 encoding rules. In addition to the DER used for a certificate, we have implemented a GSER backend in the extended eSNACC compiler. GSER can be used as an LDAP-specific encodings for newly defined attribute types. With GSER, string-based LDAP-specific encodings can maintain the structure of their corresponding ASN.1 types. The assertion values in the component filter are also represented in GSER and the extended eSNACC compiler is used to decode them into their internal representations.

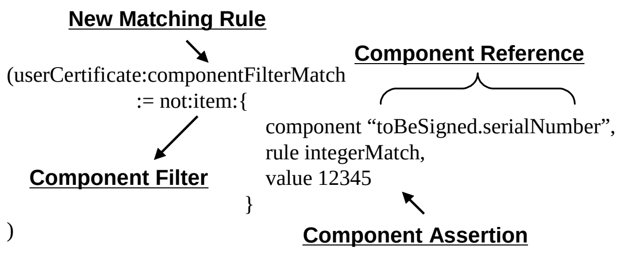


Figure 7. Example Component Filter.

5.1.2 Internal Representation of ASN.1 Type

A new data structure of *slapd* is needed to represent an attribute value as its components because the original data structure for attribute types does not contain the structural 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 values.

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.

Figure 6 illustrates the component data structure for *Certificate.toBeSigned*. For the convenience of illustration, only *serialNumber*, *signature*, *issuer*, and *subjectPublicKeyInfo* are shown with their component subtrees among ten components of *Certificate.toBeSigned*. Let's look at *subjectPublicKeyInfo* in more detail. Its component data structure, *ComponentSubjectPublicKeyInfo*, contains a pointer to its component descriptor and its own subordinate components, *algorithm* and *subjectPublicKey*. *Algorithm* is represented by *ComponentAlgorithmIdentifier* and *subjectPublicKey* is of the ASN.1 *BIT STRING* type which is represented by *ComponentBits*. Leaf nodes of the component tree, such as *ComponentBits* and *ComponentInt*, contain the values of the ASN.1 basic types.

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 the rules that attribute must follow. The first field is syntax which defines the data format used by the attribute type. The second field is matching rule which is used by an LDAP server to compare an attribute value with an assertion value supplied by LDAP client search or compare operations. Attributes must include the matching rules in their definition. At least, 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. To fully automate the component matching in which the

Table 1. Attribute Aliasing Table.

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

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 [18]. *allComponentMatch* matching rule evaluates to true only when the corresponding components of the assertion and the attribute values are the same. It can be implemented by performing matching from the topmost 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 the existing matching functions of *slapd* which might be more desirable. In order to support this refining mechanism, *slapd* checks the refinement table whether it is overridden by looking up the table, whenever a matching functions are executed.

5.2 Component Matching

5.2.1 Component Assertion and Filter

RFC 3687 [17] 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 presence or values of components within an ASN.1 value. It has a component reference to identify one component within an attribute value. Component filter is an expression of component assertion, which evaluates to either *TRUE*, *FALSE*, or *Undefined* while performing matching. Figure 7 illustrate the example component filter. The component reference or *toBeSigned.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 [15] 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. Especially, the clients that 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.

Table 2. X509 Certificate Decoding Time.

	d2i_X509() OpenSSL	ASN.1 Decoder
Time (usec)	32.74	40.20

Attribute alias 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.toBeSigned.serialNumber*" with the *integerMatch* matching rule. Hence, the filter "(*x509certificateSerialNumber=12345*)" is considered equivalent to "(*userCertificate:ComponentFilter:=item:component userCertificate.toBeSigned.serialNumber, rule caseExactMatch, value 12345*)". With the attribute aliasing, clients only have to form simple assertions to utilize component matching. Matching rule alias works in a similar way. An alias matching rule is mapped into the corresponding component reference and matching rule.

5.2.3 Component Indexing

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 maintain 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 encoding 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 constructed types, its is needed to union the indices for each value element of SET OF and SEQUENCE OF.

5.2.4 Component Caching

Whenever a certificate is matched against an incoming component filter, it is repeatedly decoded into the internal representation from DER. This requires non-negligible CPU cycles as presented in Table 2.

In order to eliminate the repeated decoding overhead, we decided to cache certificates in their decoded form, i.e. in the component

```
<ds:KeyInfo>
  <wsse:SecurityTokenReference>
    <ds:X509Data>
      <dsext:GenericCertificateReference xmlns:dsext="..." EncodingType="...#XER">
        <dsext:CertificateAssertion>
          <dsext:serialNumber>8fb2adb53a9056a511d356947cedeec0</dsext:serialNumber>
          <dsext:issuer>o=IBM,c=US</dsext:issuer>
          <dsext:keyUsage>0</dsext:keyUsage>
        </dsext:CertificateAssertion>
      </dsext:GenericCertificateReference>
    </ds:X509Data>
  </wsse:SecurityTokenReference>
</ds:KeyInfo>
```

(a) XER.

```
<ds:KeyInfo>
  <wsse:SecurityTokenReference>
    <ds:X509Data>
      <dsext:GenericCertificateReference xmlns:dsext="..." EncodingType="...#GSER">
        { serialNumber "8fb2adb53a9056a511d356947cedeec0", issuer "o=IBM,c=US" , keyUsage ‘010000000’B }
      </dsext:GenericCertificateReference>
    </ds:X509Data>
  </wsse:SecurityTokenReference>
</ds:KeyInfo>
```

(b) GSER.

Figure 8. Example GenericCertificateReference.

tree structure explained in Section 5.1.2. In the OpenLDAP directory server, the entry cache is provided to store frequently requested entries to enable very low latency access [5]. We extended the current entry cache to store decoded certificates along with other attributes of the entry. We devised various caching policies for the entry cache. In early implementations, we decided to cache a decoded certificate as a whole, along with its entry in the entry cache. The size of a certificate is 899Bytes and that of the corresponding component tree is 3KBytes. Caching all the decoded component tree consumes more than three times as much memory compared to the base entry caching. To reduce the memory requirements, we devised an indexing based caching policy. Since it is a common practice to index those attributes that are likely to be asserted, caching only those indexed components is a very practical solution to reduce the memory requirement. In our experiment, the serial number component was cached which takes only 148Bytes of memory.

6 Component Matching in WS-Security

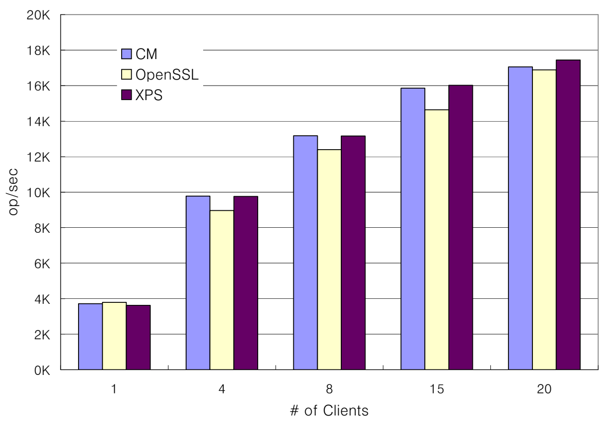
SOAP (Simple Object Access Protocol) is a protocol for invoking methods on servers, services, components, and objects [1]. It is a way to create widely distributed, complex computing environments that run over the Internet using existing Internet infrastructure, enabling Web service developers to build Web services by linking heterogeneous components over the Internet. For interpretability over heterogeneous platforms, it is built on top of XML and HTTP which are universally supported in most services. WS-Security is recently published as the standard for secure Web Services [24]. It provides a set of mechanisms to help Web Services exchange secure SOAP message. WS-Security provides a general purpose mechanism for signing and encrypting parts of a 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. There are two types of PKI clients in WS-Security: one directly accesses PKI; the other indirectly accesses it by using service proxies such as XML Key Management System (XKMS) [30] which provides clients with a simple-to-use interface to a PKI so as to hide the complexities of the underlying infrastructure.

In the X.509 token profile of WS-Security [25], it is defined that the following three types of token references can be used:

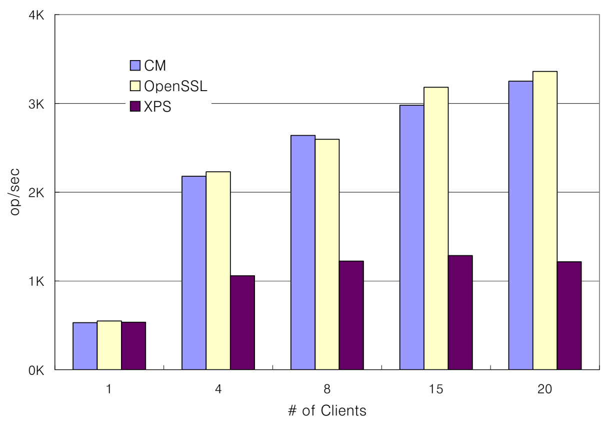
1. Reference to a Subject Key Identifier: value of certificate's *X.509SubjectKeyIdentifier.*
2. Reference to a Security Token: either an internal or an external URI reference.
3. Reference to an Issuer and Serial Number: the certificate issuer and serial number.

Because it is defined as extensible, any security token can also be used based on schemas. It is shown in Figure 8 that the <*ds:X509Data*> element of <*ds:KeyInfo*> is used as the security token. <*ds:X509Data*> defined in [31] contains various references such as *X509IssuerSerial*, *X509SubjectName*, *X509SKI*, and so on. With the ASN.1 awareness and the component matching support in the OpenLDAP directory server, these references can be used without the need of implementing syntax specific matching rules for various types of references. It is also possible in <*ds:X509Data*> to use elements from external namespace for further flexibility.

Figure 8 shows one such example. Here, *GenericCertificateReference* element from *dsext* namespace is used to provide a generic reference mechanism which implements *CertificateMatch* in the X.509 recommendation [14]. The reference consists of a sequence of certificate attributes, *serialNumber, issuer, subjectKeyIdentifier, authorityKeyIdentifier, certificateValid, privateKeyValid, subjectPublicKeyAlgID, keyUsage, subjectAltName, policy, pathToName* each of which is defined optional. By using the example reference, it would be possible to perform security key reference in a very flexible way. It would be possible to search for a certificate having



(a) 100K Entries (No Memory Pressure).



(b) 500K Entries (High Memory Pressure).

Figure 9. The Performance of Three Approaches.

a *subjectAltName* with a specific *keyUsage*. Figure 8(a) shows that the reference is encoded in XML while Figure 8 (b) shows that the reference is encoded in GSER.

With the component matching enabled LDAP server, the GSER encoded reference value can be used as an LDAP assertion value in a component filter. With the ASN.1 awareness support, the LDAP server is now capable of understanding the structure of the *CertificateAssertion* type when configured with its ASN.1 definition. Because encoders / decoders for various encoding rules (GSER, DER, XER ...) are automatically generated and integrated into the LDAP server, it is possible to use ASN.1 values encoded in those encoding rules as an assertion value in an LDAP search operation.

With the ASN.1 aware and component matching enabled LDAP server, flexible reference formats for X.509 certificates can now be defined in ASN.1 to configure 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 increased flexibility will foster the flexible use of security token references in the LDAP server by making it easy to create and update references.

7 Experimental Results

We used MindCraft's DirectoryMark [20] tools to generate the directory entries and client scripts containing a list of LDAP operations. The client scripts were run on an 8-way IBM xSeries 445 server with Intel Xeon 2.8GHz processors and the directory server was run on an IBM xSeries 445 server with 4 Intel Xeon 2.8Ghz processors and with 12GB of main memory running SUSE SLES9 (Linux kernel version 2.6.5). We used the transactional backend (back-bdb) of OpenLDAP version 2.2.22 together with Berkeley DB 4.3 and OpenSSL 0.9.7 for the evaluation. Two different size DITs with 100K and 500K entries were used for evaluation. Our intension of using two different size DITs was to observe the throughput of *slapd* with and without memory pressure. With 100k entries, all the entries was able to be cached into the DB cache. On the other hand, with 500k entries, we observed that the server experienced a number of memory swapping and disk I/O due to memory shortage. The directory was indexed for *cn*, *sn*, *email* of *inetOrgPerson* and for *serialNumber* and *issuer* of *userCertificate* (or the corresponding extracted attributes in the case of attribute extraction mechanism).

In the experiment, OpenLDAP stand-alone directory server, *slapd*, was used as an LDAP certificate repository testbed for all three methods. *Slapd* as of OpenLDAP version 2.2.22 supports both the component matching and the certificate specific matching. The attribute extraction mechanism was tested by using the XPS patch to OpenLDAP which was contributed to the OpenLDAP project by University of Salford. XPS was used to automatically generate the DIT for the attribute extraction. The same version of *slapd* was tested for all three mechanisms for the LDAP certificate repository.

Fgure 9 (a) shows the throughput of three approaches, varying the number of clients. With 100k entries, the peak throughput of component matching and attribute extraction mechanisms are almost the same. The certificate-syntax specific matching (*OpenSSL* decoder) exhibits slightly lower performance than the other two methods. We attribute the reason of lower throughput to longer code path of *slapd* such as normalization and sanity checks of assertion values when it uses the OpenSSL library. In order to observe the behavior of the three methods in the presence of memory pressure, we increased the number of entries to 500K and the database cache size is reduced from 1GB to 200MB. With this configuration, only small portion of the entries can be cached and hence the system suffers from frequent memory swapping. Figure 9 (b) shows that the throughput of all three methods are degraded significantly compared to Figure 9 (a). The peak throughput of component matching is 3250 ops/sec, significantly degraded from 17,057 ops/sec with no memory constraint. The attribute extraction mechanism is hit by even further performance degradation than the other two mechanisms. This is because the number of entries becomes doubled by extracting attributes and by having them as separate entries subordinate to the original ones. This results confirms that the component matching is a superior approach to the attribute extraction with respect to performance as well as to security and manageability.

8 Conclusion

Although it is a general consensus in the PKI standardization working group that the component matching is a complete solution to the LDAP - PKI interoperability problem, it was under debate that its adoption in LDAP servers might be slow and an alternative solution needed be pursued in the interim. In this paper, we have presented the design and implementation of the component matching in OpenLDAP *slapd*. Our work provided a strong evidence that the component matching can be implemented in pure LDAP-based directory servers without exploding complexity and degrading performance. Our work also proposed a number of enhancements to the component matching technology to improve performance and interoperability with legacy clients. In this paper, we also proposed the use of the component matching enabled LDAP as a secure on-line certificate validation protocol. We further demonstrated the usefulness of the component matching in WS-Security as a key application. As PKIs are being adopted in larger scale and in more critical deployments, it becomes more important to provide as complete a solution as possible, especially when it comes to security. The component matching technology enables more secure and flexible implementation of LDAP certificate repositories for PKI without compromising performance.

9 Availability

The component matching software is included in OpenLDAP release as a module and can be downloaded at `http://www.openldap.org/software/download/`. The eSNACC ASN.1 compiler can be obtained from DigitalNet at `http://digitalnet.com/knowledge/download.htm`.

10 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. V. 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] D. W. Chadwick and S. Mullan. Returning matched values with LDAPv3. RFC 3876, September 2004.

[5] J. H. Choi, H. Franke, and K. D. Zeilenga. Performance of the OpenLDAP directory server with multiple caching. In *Proceedings of International Symposium on Performance Evaluation of Computers and Telecommunication Systems*, July 2003.

[6] J. H. Choi, S. S. Lim, and K. D. Zeilenga. On-line certificate revocation via LDAP component matching. To be presented in DIMACS Workshop on Security of Web Services and E-Commerce, May 2005.

[7] DigitalNet. Enhanced SNACC ASN.1 software. `http://www.digitalnet.com/knowledge/snacc_home.htm`.

[8] eB2Bcom. `http://www.e2b2com.com`.

[9] T. Freeman, R. Housley, A. Malpani, D. Cooper, and T. Polk. Simple certificate validation protocol (SCVP). <`draft-ietf-pkix-scvp-18.txt`>, Feburuary 2005.

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

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

[12] 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.

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

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

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

[16] P. C. Kocher. On certificate revocation and validation. In *Proc. of the 2nd Int'l Conference on Financial Cryptography (Lecture Notes in Computer Science, Vol. 1465)*, pages 172–177, 1998.

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

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

[19] S. S. Lim, J. H. Choi, and K. D. Zeilenga. Secure and flexible certificate access in WS-Security through LDAP component matching. In *ACM Workshop on Secure Web Services held in conjunction with the 11th ACM Conference on Computer and Communications Security*, October 2004.

[20] Mindcraft. DirectoryMark. `http://www.mindcraft.com/directorymark/`.

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

[22] N. Klasen and P. Gietz. Internet X.509 public key infrastructure lightweight directory access protocol schema for X.509 certificates, <`draft-ietf-pkix-ldap-pkc-schema-01.txt`>, October 2004.

[23] M. Naor and K. Nissim. Certificate revocation and certificate update. In *Proc. of the 7th USENIX Security Symposium*, pages 217–228, January 1998.

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

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

[26] OpenLDAP. `http://www.openldap.org`.

[27] OpenSSL. `http://www.openssl.org`.

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

[29] View500. `http://www.view500.com`.

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

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

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