RC24339 (W0709-001) September 5, 2007
Computer Science

IBM Research Report

Design, Implementation, and Performance Analysis of PKI Certificate Repository Using LDAP Component Matching

Sang Seok Lim*, Jong Hyuk Choi, Kurt D. Zeilenga**
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

*Currently with Samsung Electronics, South Korea

**Currently with Isode Ltd.

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

LIMITED DISTRIBUTION NOTICE: This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g., payment of royalties). Copies may be requested from IBM T. J. Watson Research Center , P. O. Box 218, Yorktown Heights, NY 10598 USA (email: reports@us.ibm.com). Some reports are available on the internet at http://domino.watson.ibm.com/library/CyberDig.nsf/home .

Design, Implementation, and Performance Analysis of PKI Certificate Repository using LDAP Component Matching

Sang Seok Lim[*], Jong Hyuk Choi, Kurt D. Zeilenga[†]

IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598, USA

Abstract

The X.509 certificate stored in an LDAP certificate repository requires secure and flexible means to make assertions against its component values such as the identity of its owner, issuer, and the intended usage of the public key contained therein. LDAP has traditionally lacked this ability because its string based encodings do not have a standardized way to carry structural information of complex syntaxes as in X.500. The traditional remedies to this LDAP's limitation are 1) to provide certificate specific matching for a limited set of components and their combinations and 2) to extract and store the certificate components in separate searchable attributes. Neither of these remedies are considered complete because the former lacks flexibility while the latter heightens complexity in managing the integrity of the certificate repository and doubles storage requirements. Due to the significant downside of these remedies, we investigate the possibility of an ASN.1 based Component Matching alternative. In this paper, we present 1) the design and implementation of the LDAP Component Matching for an OpenLDAP directory server to facilitate its use as the certificate repository in PKI, 2) various optimization mechanisms to increase the performance of the Component Matching and their implementation in OpenLDAP, and 3) the detailed performance analysis of the LDAP directory server as a certificate repository in comparison with the traditional certificate specific matching and the attribute extraction approaches. We show that Component Matching, if equipped with the optimization techniques proposed in this paper, outperforms the traditional approaches.

[1]Sang Seok Lim is currently with Samsung Electronics, South Korea.

[2]Kurt D. Zeilenga is currently with Isode Ltd. He also serves as the Executive Director of the OpenLDAP Foundation.

1 Introduction

LDAP (Lightweight Directory Access Protocol) [8] is the predominant directory access protocol for the Internet and is also widely used as the X.509 [7] certificate repository in PKI (Public Key Infrastructure). Certificates created and signed by a CA (Certification Authority) are stored in LDAP directories so that they can be conveniently located and searched via LDAP queries on the certificate attributes. Given its pivotal role as the certificate repository in PKI, it is essential for directories to support seamless interoperation with PKI entities without any compromise in security and scalability.

The wide adoption of LDAP as a certificate repository is attributable to its ability to render lightweight directory services by the following simplifications of heavyweight X.500 DAP (Directory Access Protocol) [12].

- The reduced number of directory operations.
- Direct mapping onto a TCP/IP protocol stack.
- Simple protocol encodings.
- String-based encoding of names and attribute / assertion values.

These simplifications were intended to make DAP lightweight with minimally disturbing existing DAP applications. In fact, LDAP had first been used as the gateway to DAP DSA (Directory Service Agent) to facilitate the use of the lightweight directory client which did not necessarily equipped with a full OSI protocol stack. It evolved into the stand-alone directory service afterwards.

The last simplification, string-based encoding deviating from the ASN.1 (Abstract Syntax Notation One) [10] encoding rules of X.500, however, significantly degrades interoperability with the applications using complex data structures such as the PKI certificate repository. For instance, in PKI, a certificate needs to be located based upon the contents of its components, such as *serialNumber*, *issuer*, *subject*, and *keyUsage* [11]. LDAP search operations, however, cannot understand the ASN.1 type of the certificate as defined in [11] because attributes and assertions in LDAP are encoded in octet string with no structural information about the type. Not only would it require an exceptional effort to support certificate syntax specific matching rules such as *certificateMatch* and *certificateExactMatch* as defined in [11], 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. As a result, few new rules have been introduced to LDAP since its inception.

As a practical solution to the certificate matching in PKI, LDAP servers can store certificate components extracted from the given certificate in separate searchable attributes in parallel with the original certificate attribute. Clients, then, have to perform matching against those extracted attributes but not the original certificate. This approach is called the attribute extraction and can be performed either offline or online. Certificate Parsing Server (XPS) [2] is an online certificate extraction system which automatically extracts certificate components and stores them into separate LDAP attributes when directory entries containing certificates are being added or modified by CA. Although the attribute extraction approach has filled the interoperability gap between LDAP and PKI, it is considered as a temporary workaround because it not only introduces manageability and scalability issues but also shifts a complicated logic for structurally understanding a complex ASN.1 type from the server to the client.

In order to provide a complete solution to the component level matching and hence to enhance the interoperability between LDAP and PKI, the directory community engineered a number of extensions to LDAP. First, GSER (Generic String Encoding Rules) [14] was introduced to bring back structural information to LDAP string encodings in a standardized way. Second, the Component Matching [15] mechanism was introduced to enable component level matching to be performed against arbitrary components of composite ASN.1 types. GSER is used to represent the assertion value in an LDAP search operation which is matched against the attribute value in any ASN.1 encoding rules including GSER, BER (Basic Encoding Rules), and DER (Distinguished Encoding Rules).

Our OpenLDAP implementation of Component Matching can be considered as the first implementation of Component Matching for a pure LDAP-based directory server (overall, it can be considered as the second to the X.500-based directory server from eB2Bcom (formerly Adacel) implemented by the authors of the Component Matching standard). Since OpenLDAP, a pure LDAP-based directory server, did not originally support ASN.1 data types, we first had to bring ASN.1 awareness into OpenLDAP by providing an automatic path to generate encoders and decoders for ASN.1 encoding rules (BER, DER, and GSER) from given ASN.1 data type definitions and to generate matching and component extraction routines for each component of the composite ASN.1 data type.

Extending our previous works which introduced the early design and implementation of Component Matching in OpenLDAP [17] and discussed its applicability to WS-Security [16], this paper presents a very detailed and comprehensive description of the design and implementation of Component Matching for better PKI support, focusing on various optimizations devised to improve the interoperability and performance of Component Matching in OpenLDAP. This paper also presents a quantitative evaluation and analysis of the advantages of the Component Matching over the preexisting attribute extraction in terms of performance and scalability. To our knowledge, this is the first attempt to quantitatively evaluate the performance and scalability of an LDAP certificate repository.

The performance evaluation results showed that LDAP Component Matching performs as well as the attribute extraction if the optimizations proposed in this paper are activated. It is also shown that Component Matching outperforms attribute extraction by a significant margin in case when memory resource becomes scarce as in a constrained system or with a large scale DIT (Directory Information Tree).

This paper is organized as follows. Section 2 introduces the interoperabil-

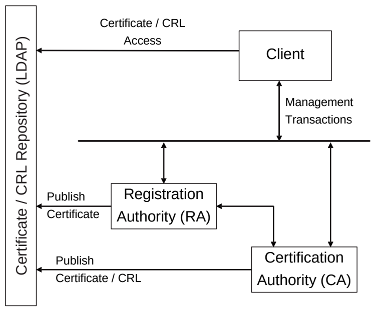

Certificate / CRL Repository (LDAP)
Certificate / CRL Access
Client
Management Transactions
Publish Certificate
Registration Authority (RA)
Publish Certificate / CRL
Certification Authority (CA)

Figure 1: The Architecture of Public Key Infrastructure.

ity issues in using LDAP as the PKI certificate repository and introduces the existing approaches to implement an LDAP certificate repository. Section 3 describes the Component Matching and GSER as the standard track approach to the component-level matching of composite attributes. Section 4 describes the use of Component Matching in PKI enabling secure and flexible certificate access. In Section 5, we present the design and implementation of the ASN.1 awareness and the Component Matching in the OpenLDAP directory server and describe various optimizations introduced to enhance Component Matching in OpenLDAP. Section 6 demonstrates the applications of the Component Matching for PKI to the security of Web services. Section 7 presents the performance evaluation results of the attribute extraction and Component Matching with varying application and system parameters and compares the previous two approaches. Section 8 concludes the paper.

2 LDAP Certificate Repository for PKI

LDAP is is the predominant directory access protocol for the Internet and is also widely used as a repository for disseminating the X.509 certificate and certificate revocation list (CRL) in PKI. Figure 1 illustrates the conceptual interoperation of four entities in PKI. In the public key registration phase, a client sends its identity as well as its public key to RA. If the identity is validated by the RA, the CA publishes the client's certificate by storing it in the LDAP directory. If the issued certificate is revoked by any reason, the CA revokes the certificate by publishing a CRL to the LDAP directory as well. Thus LDAP directories serve as the place where the clients can download certificates of other users in order to send encrypted messages or verify digital signatures. In addition, they can be informed of the latest certificate revocation information by downloading a CRL from the directory. Therefore, a number of PKIs utilize LDAP and no significant rearrangements should be done from their side to use an LDAP directory for managing processes in their environments.

When certificates or CRLs are managed in an LDAP server, a client might send an LDAP search request asserting that "find the *userCertificate* attribute

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

Figure 2a: Syntax Specific Parsing.

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

Figure 2b: Attribute Extraction.

whose issuer and serial number are o=Samsung Electronics, c=kr and 20005261". The issuer's name is contained in a certificate attribute value as a part along with other values of certificate components. Thus it is required to match the part of *userCertificate*, or the attribute value, against issuer's name in the assertion of the request. In LDAP, attribute and assertion values are represented in octet string without structural information as opposed to original ASN.1 encodings such as BER and DER. Worse, no string encodings are specified for the assertion values and there is no rule how to refer to the components of an attribute value in the request. As a result, it is very difficult to match an assertion value against a specific component of a complex attribute in a way conforming to LDAP standards. Current methods to tackle this deficiency require PKI administrator's extra effort for managing complicated Directory Information Tree (DIT) and keeping integrity of mutable attributes which will be discussed in the following subsection. These limitations of LDAP PKI interoperability have severely hampered the deployment of X.509-based PKIs.

2.1 Syntax Specific Parsing

A brute force way to providing matching for arbitrary components of a certificate against an assertion is to provide certificate-syntax specific matching rules. For instance, for an OpenLDAP directory server, an LDAP client should generate search filters in the predetermined form, not standardized, of the combination of a serial number and an issuer of *userCertificate*. For example in Figure 2 (a), "userCertificate=12345$o=ibm,c=us" is generated by a client and *slapd* [1] recognizes the serial number by reading the string right before the predetermined separator '$' followed by the issuer. The matching rule, *certificateExactMatch* in the filter is defined to be one of certificate-syntax specific matching rules that matches only the serial number and the issuer. The downside of this approach is that it will be too costly to define syntax specific matching rules for all possible components and their combinations. In addition, it is difficult to cope with changes such as certificate extensions.

[1] OpenLDAP's stand-alone LDAP daemon and associated overlays and tools,

```
(userCertificate:componentFilterMatch:=
  and:{
    item:{
      component "toBeSigned.serialNumber",
      rule intgerMatch,
      value "12345"
    }
    item:{
      component "toBeSigned.extension.*.extnValue.(2.5.29.15)",
      rule bitStringMatch,
      value ‘010000000’B
    }
  }
)
```

Figure 2c: Component Matching.

2.2 Attribute Extraction

To address the LDAP deficiencies, Klasen and Gietz [20] proposed an alternative solution, based on the workaround that PKI administrators have been using practically. A set of components are extracted from the certificate and then stored as a simple, searchable, and separate attributes in the same entry where the certificate is located. For this purpose, they defined a set of 30 attributes [20] for the X.509 certificate. On the reception of an LDAP search request in an LDAP server, matching is performed on the extracted attributes but not the original certificate. After successful completion of an attribute extraction process, writing a search filter for locating a target certificate based on its component values becomes identical to writing an ordinary attribute searching filter as shown in Figure 2 (b) where *x509SerialNumber* and *x509KeyUsage* attributes are extracted from the *userCertificate* attribute.

Although the attribute extraction facilitates matching against components of a complex attribute, it can be considered as a suboptimal approach in the following aspects. First, matching is performed on extracted attributes, not on the certificate itself. Because the contents of the extracted attributes are mutable while the certificate are immutable due to its signature, there is a chance of returning a wrong certificate to a client. It is strongly recommended for the client to verify the returned certificate again. In order to minimize this problem, the server administrator must ensure the integrity of a certificate and extracted attributes. Second, it is inevitable to restructure the Directory Information Tree (DIT) structure by creating a set of subordinate entries to store extracted attributes. When a set of attributes is extracted and stored into subordinate entries, the view of a client toward the DIT is different from that of CAs who publish the certificates. Finally, from the view point of server performance, this approach doubles or triples storage requirements to store extracted attributes along with an original certificate. In addition, on performing matching in a server-side, all candidate entries need to be loaded into a memory in advance, increasing memory requirements as well. These increased storage and memory requirements severely degrade throughput of an LDAP directory server especially when the server lacks memory and storage, which will be discussed in detail in Section 7.

2.3 Certificate Parsing Server

As an extension of the attribute extraction that PKI administrators manually extract attributes, an automatic attribute extraction mechanism was recently proposed in the Certificate Parsing Server (XPS). XPS which is designed by the University of Kent [2], automatically extracts all the certificate attributes. It is placed in front of an LDAP server as a front-end gateway to deal with incoming directory entries containing a certificate from the CA. Although it significantly relieved the PKI administrator's effort, still has suffered from all the same problems that the attribute extraction does.

3 Component Matching

3.1 GSER: Generic String Encoding Rule

A native LDAP encoding is in the format of either an octet string for textual information such as name, address, telephone, etc. or a binary for JPEG, PNG, MP3, etc. For textual information, this encoding scheme comes in handy since a user can interpret the given encoding without any decoders. This benefit, however, comes at a price that the LDAP encoding does not carry the structure of an ASN.1 type in its representation due to the LDAP's simplification while DAP's ASN.1 encodings such as BER and DER do (conversely speaking, interpreting BER/DER-encodings always require appropriate decoders). In order to solve this problem, S. Legg [14] 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 and supports reuse of the existing LDAP string encodings. It defines UTF-8 string encodings at the lowest level of the ASN.1 built-in types such as INTEGER, BOOLEAN, and STRING types and then 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 encoding, an LDAP server becomes capable of identifying the structure of the ASN.1 specification of the attribute. Furthermore, the component filter of an LDAP request is also encoded in GSER. Hence, GSER is an essential mechanism to ASN.1 awareness and Component Matching.

3.2 Component Matching

Component Matching is published in RFC 3687 [15]. All attribute syntaxes of X.500 and LDAP are described originally by ASN.1 type specifications [12, 8]. Basically, the ASN.1 type is constructed structurally from basic types to composite types just like C struct definitions. It is much of importance to remark that every field of an ASN.1 type is a component. Based on ASN.1, Component Matching defines a generic way of performing matching on user selected components in an attribute value, or a field of a given ASN.1 type specification, by introducing new notions such as component assertion, component filter, and matching rules for components. For example, an infrastructure is enabled to perform matching against an arbitrary component of an X.509 certificate, such as *serialNumber*, *issuer*, *keyUsage*, and *subjectAltName*.

In detail, Component Matching [15] defines, firstly, 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 value against an assertion value. Secondly, matching rules are defined for the ASN.1 basic and composite types. Lastly, it defines a new assertion and filter tailored for 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 the syntax is specified in ASN.1.

In order to use Component Matching, a client needs to specify a component assertion as an extensible matching of LDAP [8]. In the component assertion, there are three key fields:

- **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** is an assertion value in GSER.

No change in an LDAP client side is required, as long as the assertion/filter of the client are not hard-coded because the component assertion/filter themselves can be carried within an attribute assertion value of an LDAPv3 extensible matching as shown in Figure 2 (c).

3.3 Advantages of Component Matching

Compared to the attribute extraction approach, Component Matching has the following advantages:

1. It does not store the X.509 attributes 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 directly on its contents 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 along with the matched values control [3].

3. Flexible matching becomes possible because matching between an attribute value and an assertion value, both represented in ASN.1, is provided.

4 Component Matching and PKI

4.1 Certificate Access

With Component Matching, as mentioned in Section 3.2, matching an assertion value against a specific component of a complex attribute such as *serialNumber*, *issuer*, *keyUsage*, and *subjectAltName* becomes possible. Therefore, a client in PKI can use arbitrary fields of a certificate in order to search for the target certificate. Assume that a user has two certificates: one for digital signature; the other for non repudiation. With Component Matching, they can be located in the same directory entry while with attribute extraction they cannot. In order to find a certificate for non repudiation, a client makes a component assertion in the second *item* shown in Figure 2.(c). The component reference "toBeSigned.extension.*" means all extensions of the certificate. In the value, "extndId 15" is the object identifier of a key usage and "'01000000'B" is to check if non repudiation bit (the second bit) is set. This component assertion in the component filter enables the client to find the certificate for pre-determined purpose (non-repudiation in the example) only.

4.2 Certificate Revocation List (CRL) Access

The certificate standard [7] provides a few mechanisms to check the status of certificates. A CRL can be generated and distributed periodically by a Certificatation Authority, making it publicly accessible in the Internet by typically using an LDAP directory server. Because the CRL contains the list of all revoked certificates, it can become quite large and hence it becomes costly to transmit over the Internet. To alleviate this problems of CRLs, Online Certificate Status Protocol (OCSP) [19] was proposed to provide a timelier and more efficient status inquiry mechanism. The OCSP responder returns the only status (either *good*, *revoked*, or *unknown*) of a requested certificate without costly downloading operation.

Based on Component Matching, we also proposed an on-line certificate validation method as an alternative of OCSP [5, 6]. A CRL is a signed sequence of pairs of a revoked certificate's serial number and revocation time [7]. In order to check the status of the certificate by using Component Matching, the client needs to make a component assertion by using the serial number of the desired certificate. Then the LDAP server will perform Component Matching on the CRL, against the assertion to find the serial number of the asserted certificate in the CRL to check if the serial number is in the CRL or not. This is possible

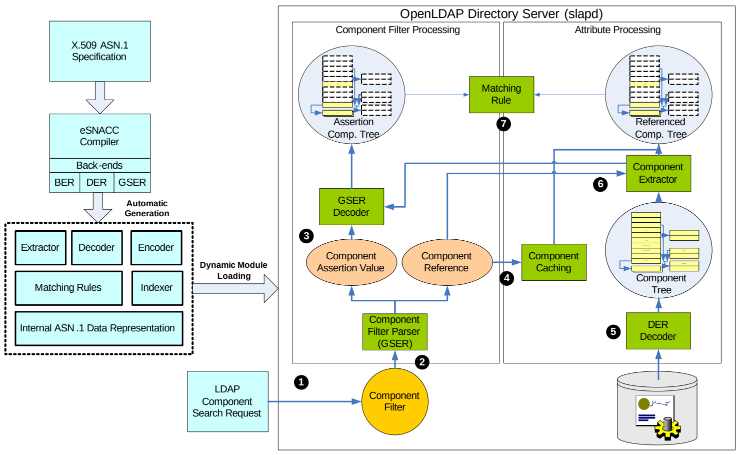


Figure 3: Architecture of Component Matching in OpenLDAP.

with Component Matching, since the LDAP server understands the structure of the CRL and is able to compare the serial number, or the specific components of the CRL against the component assertion. Therefore, the client does not necessarily have to download and scan the CRLs any more. Furthermore, an LDAP server already has been widely used for distributing CRLs and certificates. Hence, if the server can perform validity checking over the CRLs as well, it will be very practical and an efficient alternative to OCSP. Please refer to our previous studies [5, 6] for more detailed information about Component Matching based certificate validation.

5 Component Matching in OpenLDAP

The overall architecture of the Component Matching in the OpenLDAP *slapd* directory server is illustrated in Figure 3. 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 [13] to support the GSER in addition to the originally supported BER and DER [13]. 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 to *slapd*.

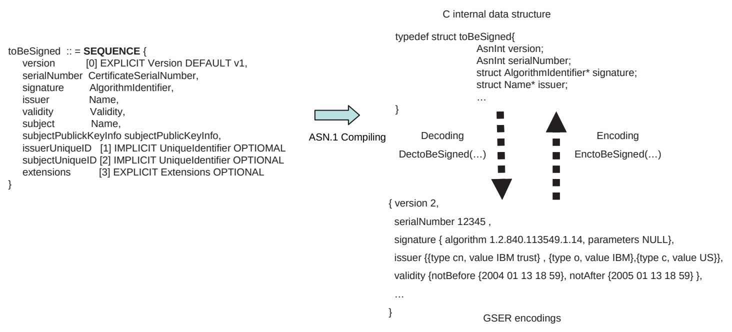
After loading the eSNACC-compiler generated module into the slapd, it becomes able to deal with the new attribute syntax supported by the module. With Figure 3, we summarize the overall processing steps of incoming LDAP search requests in the OpenLDAP directory server as follows,

1. A search for components of an X.509 certificate attribute is initiated by the inclusion of the component filter in the filter of the search request.
2. On the reception of the search request, the component filter contained in the request is parsed to obtain component assertion values and component references.
3. The component assertion value is decoded to the internal representation for ASN.1 by the GSER decoder. If the values is a composite type, it is also converted into a component tree.
4. With the obtained component reference, the component cache is retrieved to check if the target certificate already exists or not in a decoded form. If hit, goto 7.
5. Candidate directory entries having certificates are loaded and then the included certificates are decoded by an appropriate ASN.1 decoder according to the encoding rule of the corresponding attribute syntax. Because X.509 certificate is DER encoded, a DER decoder is used to decode the certificate into an internal representation of an ASN.1 type, or a component tree.
6. The component reference is fed into the component extractor to obtain a sub tree out of the attribute component tree.
7. The assertion component and the extracted attribute component are then matched against each other by the matching rule which is specified in the component filter.
8. After matching is completed at the server, the client is returned with entries including the target certificates. The client MUST verify the signature of the certificates to see if they are maliciously altered or not.

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

5.1 ASN.1 Awareness

As explained in the previous section, Component Matching is based on ASN.1. A current OpenLDAP directory server, however, is incapable of dealing with ASN.1 encodings such as BER, DER, and GSER due to the LDAP's adoption of string-based encodings (The server does not necessarily have to be prepared with necessary functions to handle the ASN.1 encodings). In an effort to make the server ASN.1 aware, we employ the widely used ASN.1 compiler, eSNACC, and extend it to automatically generate all necessary functions and data structures in C required for processing a given ASN.1 encoding in the server. In the following section, the methodologies to bring ASN.1 awareness into the server will be elaborated.



```
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
}
```

```
typedef struct toBeSigned{
                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 4: ASN.1 TBSCertificate specification, its compiler output, and example GSER encodings (*certificate should be DER-encoded).

5.1.1 eSNACC Compiler

An ASN.1 compiler translates ASN.1 modules (specification) into C / C++ data structures and their encoding / decoding routines. The eSNACC ASN.1 compiler generates them for BER and DER. In Component Matching, a component assertion value is encoded in GSER, so it is required to extend the compiler to have a GSER backend. The encoding / decoding routines enable back and forth data conversion between BER / DER / GSER encodings and the instantiation of C / C++ data struct, making the server be capable of handling any component of given ASN.1 encodings internally. For example, Figure 4 illustrates the ASN.1 specification of an X.509 certificate, example GSER encodings, internal C data struct definition, and encoder/decoder functions. The generated internal C data struct 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 `DDectoBeSgined()`.

Besides an encoder, decoder, C data struct, following functions are also generated automatically by our ASN.1 compiler.

- Component extractor: Component Matching enables matching an arbitrary component of attribute values against a component assertion, which requires the capability of extracting any component out of a given attribute value.
- Component matching rule: Component Matching defined a new matching rule, *allComponentMatch* required for matching against a composite ASN.1 type as specified in RFC 3687 [15]. Further explanation will be given in the Section 5.1.3.
- Component indexer: Most LDAP directory servers collaborate with a database that acts as an underlying infrastructure to store directory entries. When it comes to database access, indexing on entries is an essential function from the perspective of performance. The OpenLDAP

has supported only attribute-level indexing, leading us to newly engineer component-level value indexing. Further explanation will be given in Section 5.4.

5.1.2 Internal Representation of ASN.1 Types

A new data structure of *slapd* is needed to represent an attribute value as its components because the original data structure for an attribute type 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, extract, index, and match the values.

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 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 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 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 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.

5.2 Component Assertion and Filter

RFC 3687 [15] 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. The 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. The component filter is an expression of component assertions, which evaluates to either *TRUE*, *FALSE*, or *Undefined* while performing matching. For example, in Figure 5, the component reference or *toBeSigned.serialNumber* identifies

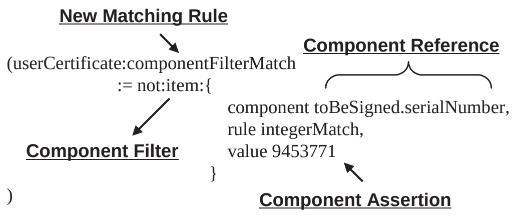


Figure 5: Example Component Filter.

Table 1: Attribute Aliasing Table.

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

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 [12] 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 and integrated into slapd while the routines for the component assertion values are automatically generated from a given ASN.1 type.

5.3 Component 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, 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.

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 aliasing table is shown in Table 1. *X509certificateSerialNumber* attribute is aliased to "*userCertificate.toBeSigned.serialNumber*" with the *integerMatch* matching rule. Hence, the filter "(*x509certificateSerialNumber=1*)" is considered equivalent to "(*userCertificate:ComponentFilter:=item:component toBe-*

Signed.serialNumber, rule integerMatch, value 1)". With the attribute aliasing, clients only have to form simple assertions to utilize Component Matching. Matching rule aliasing works in a similar way, so an aliasing matching rule is mapped into the corresponding component reference and the matching rule.

5.4 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.

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. Once a component is selected for indexing, corresponding syntax, component matching rule, and component value are combined to generate a hash key value. On generating the hash key, a component value can be either a basic ASN.1 type or composite ASN.1 type, and it is encoded in BER, DER, and GSER. In our OpenLDAP implementation, we decide to use GSER when generating a hash key of a component since an incoming component assertion value from a client is GSER-encoded. If a component is a basic ASN.1 type, it is quite straightforward to generate a hash key value since a GSER-encoded component value is used as it is without any manipulation. For a composite component, however, careful consideration is required to generate a hash key, depending on which constructors (SET, SEQUENCE, SET OF, SEQUENCE OF) are used. 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 hash key value. For <all> component reference of SET OF and SEQUENCE OF constructed types, it is needed to union the indices for each value element of SET OF and SEQUENCE OF.

5.5 Component Caching

The OpenLDAP server is equipped with two caches for the performance improvement [4]; one is an entry cache, the other is a BDB database cache. The entry cache is to store frequently requested entries to reduce the latency of accessing a back-end BDB database. A database cache is to hide disk access latency. The sizes of both caches should be configured accordingly to maximize throughput, considering the size of available physical memory and the number of entries. The current entry cache only stores the DER form of a certificate. Whenever matching on a certificate is performed, the DER-encoded certificate needs to be decoded repeatedly to a internal representation, a component tree. The DER decoding latency of a human-optimized OpenSSL decoder and our ASN.1 compiler generated decoder for Component Matching are measured and shown in Table II.

In an effort to remove the decoding latency, we devise various caching policies for the entry cache storing a decoded component tree. In brief, the overall

Table 2: Decoding Time

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

policies are summarized as follows;

- *CM-NC*: Decoded component tree is not cached.
- *CM-FC*: Decoded component tree as a whole is cached.
- *CM-VC*: Values of components which are unlikely to be used for matching are excluded from caching.
- *CM-SC*: Only indexed components are cached.

In early implementation, we decided to cache a decoded certificate component tree, as a whole, along with its entry into the entry cache, which is named as a *CM-FC* scheme. The size of *userCertificate* is around 1Kbbyte and that of the corresponding decoded component tree is approximately 3Kbytes. Caching all the decoded component tree consumes more than three times as much memory as the non-caching scheme (CM-NC). Hence, in order to reduce the memory consumption of *CM-FC*, we devise *CM-VC* which caches only selected component values of a given certificate. In *CM-VC*, the values of a certificate such as a public key and a signature that are highly unlikely to be used for matching upon by a client are excluded from the cache. In other word, *CM-VC* does not cache those values specified in an ASN.1 BIT STRING type, saving around 273Bytes for each certificate. Even with this improvement, it still appears that the memory requirement of *CM-VC* is still very high as compared to *CM-NC*, leading us to devise more memory-efficient scheme. In a directory, all attributes that are highly likely to be requested by clients are always indexed by a server administrator. Therefore, we came to the conclusion that caching only the indexed components would be a very practical solution to relieve the high memory requirement which is named as a *CM-SC* scheme. In our experiment, the serial number component was indexed and cached which took only 148Bytes. As a conclusion, it is observed that *CM-SC* can reduce memory requirement in the proportion to how many components are indexed.

6 Applications of Component Matching

Component Matching facilitates the development of emerging applications encompassing WS-security [21] and XKMS [23]. In the following two subsections, how the applications can take full advantage of Component Matching will be described. Those readers who are not interested in Web services can skip this section.

6.1 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 [21]. 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 message 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) [23] 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 [22], 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 6 that the <*ds:X509Data*> element of <*ds:KeyInfo*> is used as the security token. <*ds:X509Data*> defined in [24] contains various references such as *X509IssuerSerial*, *X509SubjectName*, *X509SKI*, and so on. With the ASN.1 awareness and the Component Matching supported 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 6 shows one such an example. Here, *GenericCertificateReference* element from *dsext* namespace is used to provide a generic reference mechanism which implements *CertificateMatch* in the X.509 recommendation [11]. The reference consists of a sequence of certificate attributes, *serialNumber, issuer, subjectKeyIdentifier, authorityKeyIdentifier, certificateValid, privateKeyValid, subjectPublicKeyAlgID, keyUsage, subjectAltName, policy, pathToName* each

```
<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>
```

Figure 6a: 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>
```

Figure 6b: GSER.

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 *subjectAltName* with a specific *keyUsage*. Figure 6 (a) shows that the reference is encoded in XML while Figure 6 (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.

6.2 Component Matching and XKMS

Figure 7 illustrates an overall 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) [23] 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 [23]. It defines two key services: locate and validate. In the following subsection, it will be presented how the Component Matching can be used in the X-KISS services with respect to certificate.

6.2.1 Certificate Access

Figure 8 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 8. 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 from the request by the XKMS services as shown in Figure 2. The

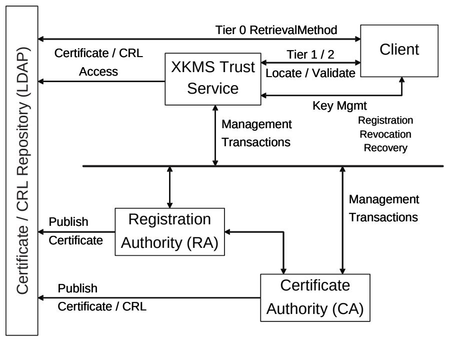
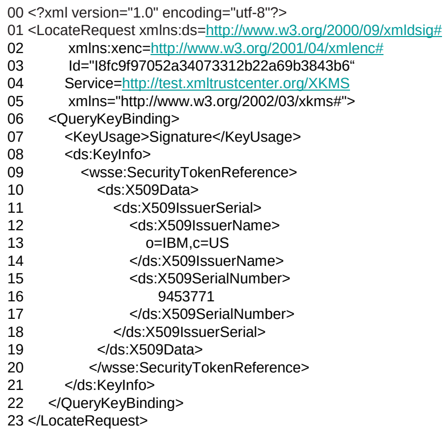


Figure 7: XKMS in PKI.



```
<?xml version="1.0" encoding="utf-8"?>
<LocateRequest xmlns:ds=http://www.w3.org/2000/09/xmldsig#
          xmlns:xenc=http://www.w3.org/2001/04/xmlenc#
          Id="I8fc9f97052a34073312b22a69b3843b6“
         Service=http://test.xmltrustcenter.org/XKMS
          xmlns="http://www.w3.org/2002/03/xkms#">
     <QueryKeyBinding>
         <KeyUsage>Signature</KeyUsage>
         <ds:KeyInfo>
             <wsse:SecurityTokenReference>
                 <ds:X509Data>
                     <ds:X509IssuerSerial>
                         <ds:X509IssuerName>
                             o=IBM,c=US
                         </ds:X509IssuerName>
                         <ds:X509SerialNumber>
                             9453771
                         </ds:X509SerialNumber>
                     </ds:X509IssuerSerial>
                 </ds:X509Data>
               </wsse:SecurityTokenReference>
         </ds:KeyInfo>
     </QueryKeyBinding>
</LocateRequest>
```

Figure 8: Example XKMS Locate Service Request.

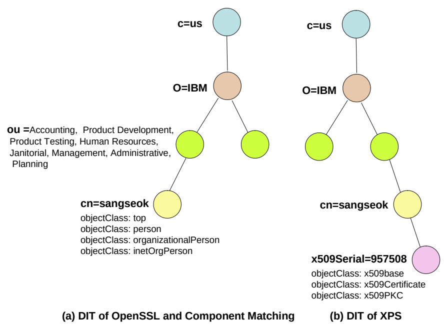

c=us
O=IBM
ou =Accounting, Product Development, Product Testing, Human Resources, Janitorial, Management, Administrative, Planning
cn=sangseok
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
(a) DIT of OpenSSL and Component Matching
c=us
O=IBM
cn=sangseok
x509Serial=957508
objectClass: x509base
objectClass: x509Certificate
objectClass: x509PKC
(b) DIT of XPS

Figure 9: DIT Structures of the LDAP Certificate Repository.

component reference "toBeSigned.extension.*.extnValue.content.(2.5.29.15)" refers to all extensions of the certificate of which *KeyUsage* is '100000000'B. It is used to check if a certain bit (the first bit for signature) is set. The corresponding component filter of Figure 2 enables the XKMS service to find a certificate of a subject 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.

7 Performance Analysis of LDAP for PKI

This section analyzes the performance of the three different approaches to the LDAP certificate repositories: Component Matching, dedicated certificate matching, and attribute extraction.

7.1 Experiment Environment

The SUT (System Under Test) is 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 MindCraft's DirectoryMark [18] tools to generate the directory entries and client scripts containing a list of LDAP operations. We developed a new multi-threaded client program in Linux that reads the DirectoryMark script and requests corresponding LDAP operations to the LDAP server. The client program was run on an 8-way IBM xSeries 445 server with Intel Xeon 2.8GHz processors. 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.

Figure 9 shows the DIT structures used in the evaluation. The DIT for the Component Matching and the dedicated certificate matching (Figure 9 (a)) consists of *inetOrgPerson* entries each having a *userCertificate* attribute under 10 *organizationalUnits* under the base entry. The *userCertificate* attributes was

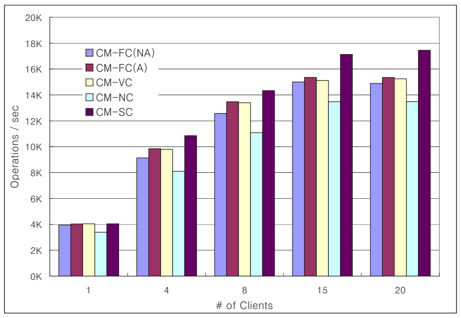
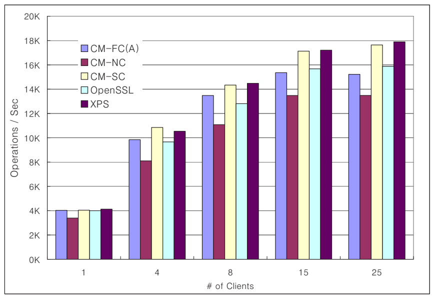
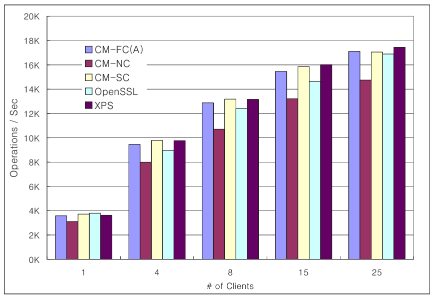


Figure 10: Performance of Component Caching Policies. (NA and A denote no aliasing and aliasing respectively



(a) 10K Entries



(b) 100K Entries

Figure 11: Performance of LDAP Certificate Repositories without Memory Pressure.

created by using the OpenSSL toolkit. In the DIT for the attribute extraction mechanism, on the other hand, each *inetOrgPerson* has one additional subordinate entry for one *userCertificate* it has. This additional entry contains *userCertificate* along with the corresponding extracted attributes. Each *userCertificate* should exist as a separate object.

Three different DIT sizes, 10K, 100K, and 500K entries were used for evaluation. Entries were searched randomly so that the memory working set size is increased in proportion to the number of entries. 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.

7.2 Performance of Component Caching

Figure 10 shows the throughput of Component Matching with various caching policies for 10K entry directory. Search was performed against *serialNumber* component of the *userCertificate* attribute. The entry cache size and database cache size are set to 10K entries and to 1GB respectively so that all the entries were able to be cached into both caches. *CM-FC* improves the throughput by 15% as compared to *CM-NC*. *CM-SC* not only saves memory but also outperforms other component caching polices by as much as 20%. This is because caching extracted components reduces the code path of *slapd* by eliminating the BER decoding operations out of it. It only checks if the component reference of an incoming component assertion is identical to the extracted component's in the entry cache and directly compares the assertion value with the cached component value. *CM-VC* shows almost the same performance as does *CM-FC* with a slightly decreased memory usage.

7.2.1 Performance of Component Aliasing

In Section 5.3, we discussed the component aliasing mechanism in the context of providing backward compatibility to legacy client applications. Interestingly, it also turned out that aliasing also has significant impact on throughput. Aliasing improves the throughput of *CM-FC* by up to 11% as depicted in Figure 10. This is mainly because the component aliasing helps to bypass both the overhead of the extensible filter parsing step which has a very long code path to read from BER stream and that of the component filter parsing step including GSER stream decoding.

7.3 Performance Analysis of Three LDAP Mechanisms for Certificate Repository

Figure 11 shows the throughput of three approaches with varying the number of entries and clients. As the number of clients increases, throughput of all the methods increases up to the point of 20 clients. When it reaches to 20 clients, the server becomes saturated, restricting its throughput lower than the peak throughput. The throughput of *CM-SC* and attribute extraction mechanisms follow almost the same characteristics. The certificate specific matching (*OpenSSL* decoder) exhibits slightly lower performance than the other two methods. We attribute the reason of the lower throughput of the certificate specific matching to longer code path of *slapd* such as normalization and sanity checks of assertion values using the OpenSSL library. *CM-NC* shows the worst throughput among them as expected due to the overhead of the repeated certificate decodings. *CM-FC* performs well for both 10K and 100K entries even

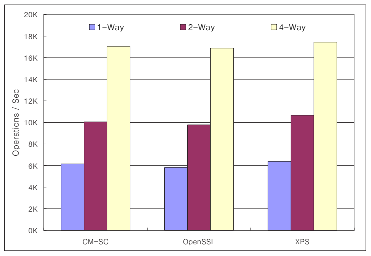


Figure 12: SMP Scalability (1-way, 2-way, and 4-way).

though its memory consumption is high because sufficient amount of memory resources were provided.

Figure 12 shows the scalability of *slapd* with 100k entries on an SMP machine, varying the number of processors from 1 to 4. As the number of processors increases, the throughput also increases in three methods at almost the same scaling factors. The scaling factor is approximately 65% of the ideal linear scaling.

In order to observe the behavior of the three methods in the presence of memory pressure, we increased the number of entries to 500K entries. The database cache size is reduced to 200MB from 1GB and the entry cache size is set to 50,000 from 100,000. With this configuration, only small portion of the entries can be cached and hence the system suffers from frequent memory swapping. Figure 14 (a) shows that the throughput of all three methods are degraded significantly. The peak throughput of *CM-SC* is 3250 ops/sec, significantly degraded from 17,507 ops/sec throughput with sufficient memory caching. The attribute extraction mechanism was hit by even further performance degradation than the other two mechanisms. This is because the number of entries becomes doubled by extracting attributes and having them as a separate entry subordinate to the original entry. This performance degradation confirms the intrinsic limitation of the attribute extraction approach which multiplies the number of directory entries as many entries should be created as there are *userCertificate* attribute values. When we increased the database cache size to 800MB, the throughput bounced back as shown in Figure 14 (b). The peak throughput of *CM-SC* becomes 4500 ops/sec. The throughput of the attribute extraction mechanism is still the worst among the three even though it becomes doubled by the increase in the database cache size.

Figure 13 shows that the scaling factors of the three methods remain almost the same as that without memory pressure. However, the performance of the attribute extraction mechanism dropped almost to the half of those of the other two mechanisms. It is obvious in all system sizes that the attribute extraction approach is more susceptible to memory resource constraint than the others because of the increased directory size.

Table 3 shows the directory population times with 100k entries. XPS is about 5 times slower than the other two. This is because it extracts attributes

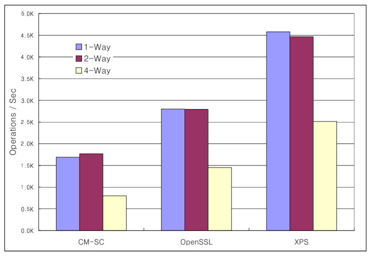


Figure 13: SMP Scalability under Memory Pressure
(Database Cache Size = 800MB).

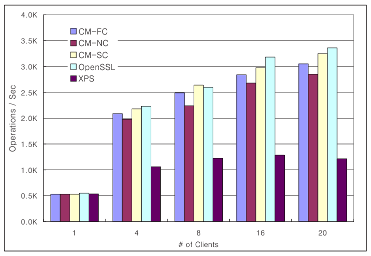
and stores them in the subordinate entry separately. This clearly multiplies the number of database update operations. Additional overhead comes from excessive WAL (Write Ahead Logging) [2] on top of the transaction logging of the underlying Berkeley DB to maintain transactional semantics across the addition and certificate extraction of the original object and the addition of the certificate objects. The directory loading time shown in Table 3 already reflects the indexing overhead for *serialNumber* and *issuer* component of *userCertificate* in Component Matching and certificate specific matching. Indexing requires decoding of the DER encoded certificates into the indexed components to obtain the component values to generate indexing keys.

7.4 Assessment of Three LDAP Mechanisms for Certificate Repository

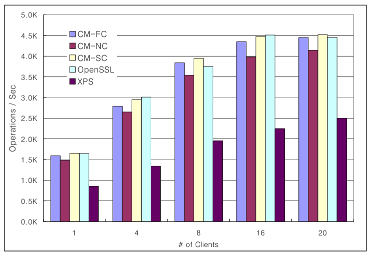
Configuring a directory server's entry cache and database cache has a significant impact on the overall performance. As observed in the series of experiments, each mechanism has different memory and storage requirements which requires careful consideration of the characteristics of the three mechanisms in the capacity planning process. Table 4 summarizes the characteristics of the three mechanisms for LDAP certificate repository. Regarding the storage requirement, the attribute extraction mechanism has multiplied storage requirement to provide the same flexibility as does Component Matching. It has to extract a priori all the certificate components which can be matched by the client applications for maximum flexibility. Component matching does not need additional storage

Table 3: Directory Load Performance (100K entries).

	Component Matching	Certificate Matching	XPS
Time	178sec	167sec	815sec
DB Size	234MB	234MB	410MB



(a) Database Cache: 200MB



(b) Database Cache: 800MB

Figure 14: Performance of LDAP Mechanisms Certificate Repositories under Memory Pressure (500K Entries).

except for the certificate itself. Component matching's memory requirement is determined by the caching policy. If CM-SC is used, it is almost the same as that of the certificate specific matching. The multiplied number of entries in attribute extraction also increases the memory requirement accordingly.

When multiple certificates are stored for a subject entry, the attribute extraction mechanism need to create one subordinate entry for each certificate. This not only complicates the management of DIT but also increases the search latency because the LDAP server has to visit more entries during DIT traversal. With the certificate specific matching and Component Matching, on the other hand, multiple certificates can be stored in a single object.

The certificate specific matching lacks the flexibility of the Component Matching and the attribute extraction approaches. Currently, the matching on "*serialNumber$issuer*" is the only supported certificate specific matching in OpenLDAP. In order to support matching against various components and their combinations, vendors should provide matching rule codes for each combination to be processed in *slapd* by using an OpenSSL library. The other two methods, on the other hand, provide a very flexible way to handle this demand. Both use ASN.1 compilers to provide an automatic path to generate necessary codes for various matching.

The search performances of the three methods are comparable when there is no memory constraint. In the presence of memory resource constraints, however, the search performance of the attribute extraction becomes about 40% of the other two methods. Finally, the directory population performance of the attribute extraction mechanism is almost five times lower than those of the other two methods. If a large number of certificates needs to be published frequently, it is discouraged to use the attribute extraction mechanism unless the directory loading performance is significantly improved.

Table 4: Characteristics of Three LDAP Certificate Repositories.

	Component Matching	OpenSSL	XPS
Normalized Storage Requirement	1	1	2
Normalized Memory Requirement	1 - 3	1	2
Multiple Certificate Support	Simple	Simple	Complex
Matching Flexibility	Good	Bad	Good
Security	High	High	Low
Peak Search Perf. wo memory pressure	0.84 -0.97	0.97	1
Peak Search Perf. w memory pressure	1	1.02	0.4
Normalized Directory Loading Time	1.00	0.94	4.58

8 Conclusion

This paper presented how Component Matching enables flexible and secure certificate access when LDAP directories are used as the certificate repositories in PKI. This paper also presented a detailed description of the design and implementation of the Component Matching in OpenLDAP directory server focusing on various optimizations we proposed to improve the performance and interoperability of the Component Matching. Another important contribution of this paper is the quantitative performance evaluation and analysis of the Component Matching compared with the preexisting practices of certificate specific matching and the attribute extraction. The result convinced us that higher manageability and flexibility can be accomplished by using the Component Matching without any compromise in performance. It also shows that the Component Matching outperforms the attribute extraction approach in a resource constrained system. The evaluation results presented in this paper can be found useful in evaluating the three different options to implement LDAP certificate repository in PKI and can be used as base data for capacity planning to build such systems as well. The Component Matching technology and its implementation in OpenLDAP enable flexible and high performance LDAP certificate repository for PKI. Finally, it is noteworthy that even though our paper focused on describing the use of Component Matching for X.509 certificate but it does not necessarily mean that its use is limited to X.509 certificate. This is because, fundamentally, Component Matching is a very general solution to store and retrieve any data via LDAP as long as its syntax is defined by ASN.1.

8.1 Availability

The Component Matching is included in OpenLDAP release which 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.

References

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

[2] 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.

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

[4] J. H. Choi, H. Franke, and K. D. Zeilenga. Enhancing the performance of openldap directory server with multiple caching. In *International Symposium on Performance Evaluation of Computers and Telecommunications Systems*, July 2003.

[5] J. H. Choi, S. S. Lim, and K. D. Zeilenga. A new on-line certificate validation method for improved security, scalability, and interoperability. In *IEEE Information Assurance Workshop*, June 2005.

[6] J. H. Choi, S. S. Lim, and K. D. Zeilenga. On-line certificate revocation via ldap component matching. In *DIMACS Workshop on Security of Web Services & E-Commerce*, May 2005.

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

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

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

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

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

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

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

[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] 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.

[17] S. S. Lim, J. H. Choi, and K. D. Zeilenga. Design and implementation of ldap component matching for flexible and secure certificate access in pki. In *Internet2 PKI R&D Workshop*, April 2005.

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

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

[20] N. Klasen and P. Gietz. An ldapv3 schema for x.509 certificates, <draft-klasenldap- x509certificate-schema-00.txt>, February 2002.

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

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

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

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