Adding Certificate Retrieval to OpenLDAP

Deliverable D4: Final Report

Version 1.0. D.W.Chadwick 8 August 2004

Table of Contents

Introduction

There are a number of deficiencies in LDAP when it is used to support PKIs. Firstly it is not possible to retrieve a single attribute value from a multi-valued attribute. Secondly, it is not possible to search for an X.509 [1] attribute (i.e. a public key certificate (PKC), attribute certificate (AC) or certificate revocation list (CRL)) based on its value. These and other deficiencies have been described more fully in a paper in Communications of the ACM [2]. There were a number of Internet Drafts [3,4,5] that once implemented, would solve both of these problems. The intention of this TERENA project was to implement two of these IDs, namely matched values [4] and certificate matching rules [5] and to build them into the OpenLDAP source code.

OpenLDAP Software [6] is a community developed LDAP implementation that provides an enterprise level LDAP server as well as client tools and SDKs. OpenLDAP Software also serves as a reference implementation for the community. OpenLDAP Software's open source licensing allows vendors to incorporate enhancements into their products which furthers the availability of these technologies.

Initial Objectives

The specific objectives at the start of the project were:

1. To implement the matched values LDAPv3 control in the OpenLDAP source code
2. To implement the matched values LDAPv3 control in an LDAP client, so as to demonstrate its functionality
3. To implement (a subset of) the certificate matching rules specified in [5] in the OpenLDAP source code
4. To implement (a subset of) the certificate matching rules specified in [5] in an LDAP client so as to demonstrate its functionality
5. To progress the Internet Drafts [3, 4, 5] to RFC proposed standard status.

Initial Project Plan

The project was planned to run for 18 months, starting in September 2001. It was to consist of 3 phases. A detailed set of the tasks and resources needed to achieve this are shown in Appendix 2.
Phase 1. After 6 months to demonstrate that a certificate can be added to OpenLDAP and its fields stored in the indexes so that it can subsequently be searched for.
Phase 2. After 12 months to demonstrate the retrieval of single certificates from an entry using equality matching.
Phase 3. After 18 months to demonstrate an enhanced LDAP client that can request certificates that match on certain fields contained within them e.g. the email address of the user, the name of the CA, the key usage fields etc.

It was noted that it would never be possible to implement the complete set of fields in a certificate given the myriad of possible extensions that can be included, but that the exact set of fields would be chosen by talking to the Terena TF-LSD and PKI-COORD groups and determining their user requirements.

Initial Set of Deliverables

The project initially had 4 deliverables as shown in the Table below. This report is the fourth of these deliverable.

Phase	Deliverable number	Date due	Title / Description
1	D1	6 months from start	Detailed design document for modifying OpenLDAP source code
3	D2	End of Project	Updated Internet RFCs or IDs for [3,4,5]
3	D3	End of Project	Enhanced OpenLDAP source code
3	D4	End of Project	Project Report to TERENA (this document)

Project's Progress

Just before the project started David Chadwick had a meeting with Kurt Zeilenga and Steven Legg at the IETF meeting in London, August 2001. It was decided to fundamentally change the design of the project, to align with a new Internet Draft published by Steven Legg entitled "LDAP & X.500 Component Matching Rules", subsequently to become RFCs [7] and [8]. This would significantly improve the functionality of the certificate matching in OpenLDAP, and make it infinitely extensible, in order to cater for new certificate and CRL extensions that might be defined in the future. Of course, it would make the design more complex, and would require LDAP clients to be modified to support this instead of the original scheme, but we believed that we could handle this within the scope of the current project.
Note that subsequent to this decision, this approach has been sanctioned by the IETF as the standard way to search for certificates with LDAP, so the decision was fundamentally sound.

Component matching is pictorially illustrated in Figure 1 below.

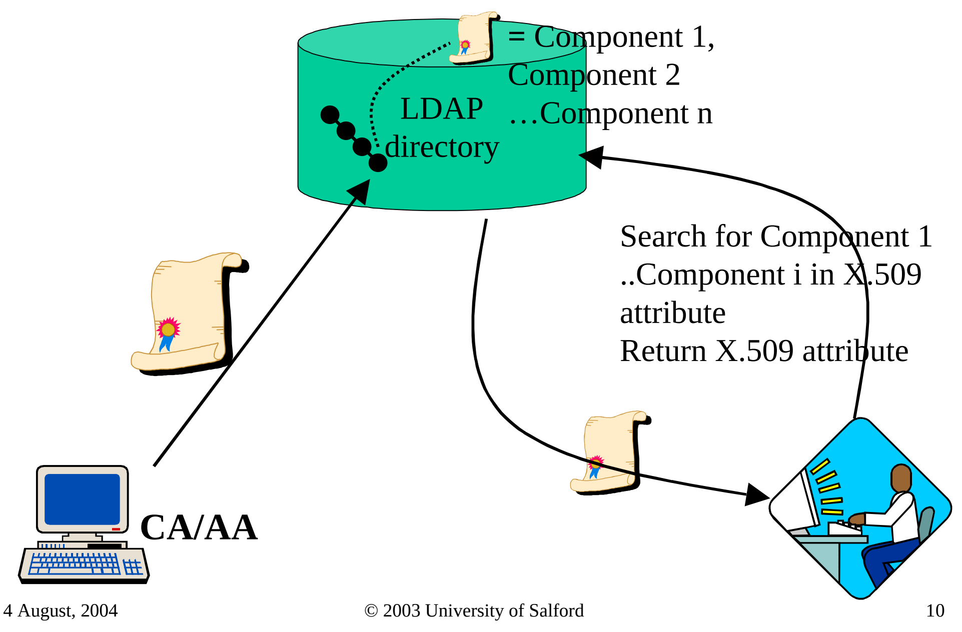


Figure 1. Component Matching

The project initially made good progress and for the first 8 months stayed broadly on track with the original plan shown in Appendix 2. The Matched Values functionality was released on time at the end of month 8 and the corresponding Internet Draft successfully passed Last Call (since then it has been with the Area Director for over 18 months, but recently has just passed the IANA considerations hurdle and should be soon issued as a standards track RFC). However, progress was hampered by the almost complete lack of documentation concerning the internal workings of OpenLDAP. In most cases the code was the documentation.

During month 9, a MACE-DIR telephone call raised the subject of certificate matching, and discussed a new approach proposed by Peter Gietz et al. [13]. This was based on creating a new entry for a certificate, and creating attributes from all the fields of a certificate. A major benefit is that LDAP servers do not need to be modified, since they can run our modified OpenLDAP XPS server as a front end processor between their LDAP server and their CA/AA. Another major benefit is that LDAP clients do not need to be modified to support this approach. The outcome was that we decided to poll the NRENs to see which design of certificate matching they preferred. They voted almost unanimously for the Gietz approach rather than the component matching approach, and so we decided to change our design to this scheme. This attribute extraction approach is shown in Figure 2 below.

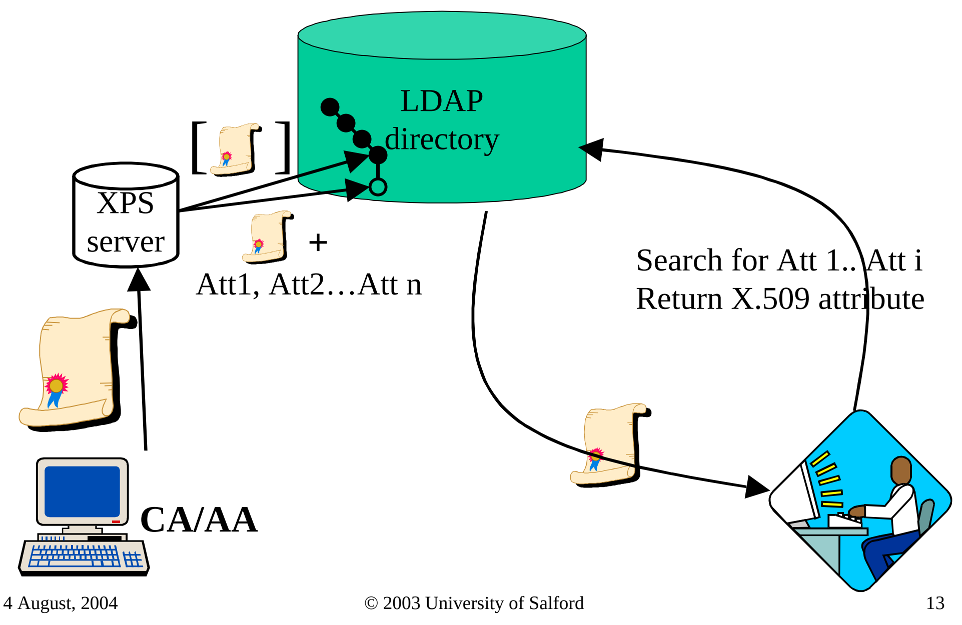


Figure 2 Attribute Extraction

This significant change of design obviously caused some delay to the project, but a revised project plan was produced (Appendix 3), which predicted that we could still finish on time.
Note that subsequent to this decision, the IETF decided that this is not to be the standard approach to certificate matching in LDAP, but rather is to be a short term expedient approach. Consequently the Internet Drafts [16, 17] upon which this approach is based, are only to become Information RFCs rather than standards track RFCs.

A new Detailed Design was produced [15] in parallel with the implementation (this was because we needed to understand the internals of OpenLDAP, so a certain amount of experimentation was necessary) and submitted to the TF-ACE group in month 15. At this stage the project was judged to be about one month behind schedule. Supporting Internet Drafts [16, 17] were produced in month 18.

However, as the detailed design was being implemented a number of problems arose. These included:

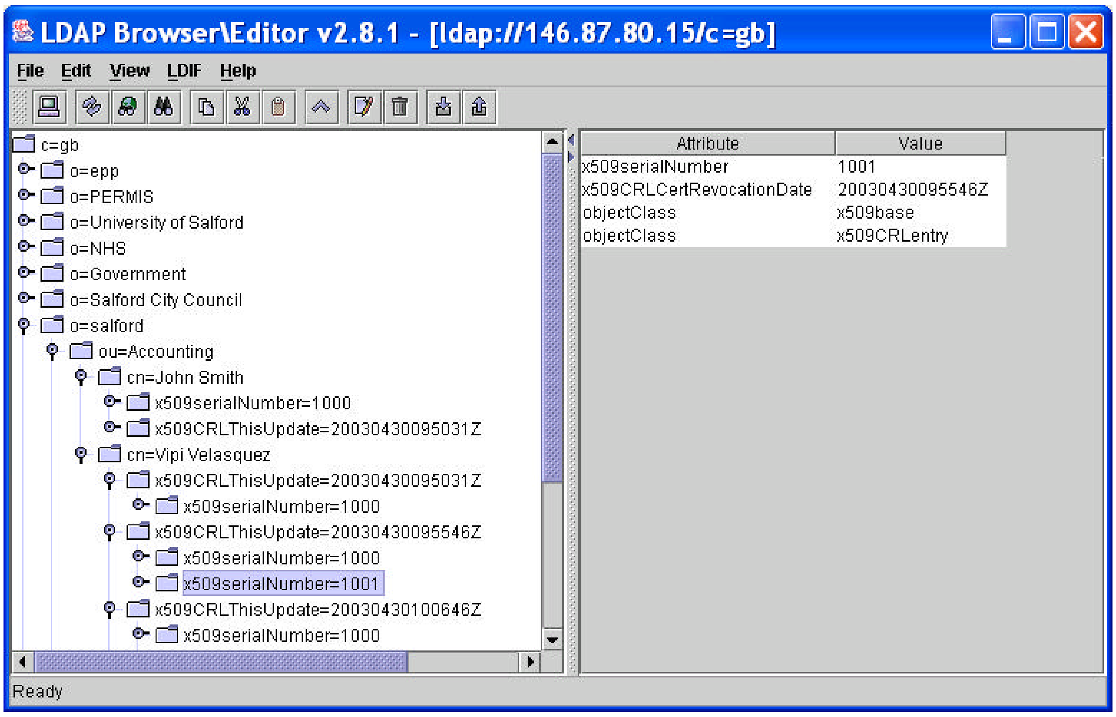
- it was realised that a CRL could produce a whole subtree of LDAP entries, rather than a single entry,

- the internal search primitive returned the search result entries directly to the client, and not to the internal calling code. This meant we needed to modify the send_search_entry function to trap the results
- during the debugging we uncovered a potential denial of service attack, whereby a malicious user could submit a spurious XPS delete attribute operation on the root (or close to the root) of the DIT. This would cause a full subtree Search to be performed, which obviously would consume huge resources.

These problems led to significant redesigns being needed and consequent delays in the project. Nevertheless we persevered and eventually overcame them all. The full bimonthly reports of the project's progress can be found in Appendix 1.

A demonstration of the XPS server was successfully made at the Terena Networking Conference in Rhodes in June 2004. A screen shot from this demonstration can be seen in Figure 3.

Figure 3 LDAP Client view of XPS



```
c=gb
  o=epp
  o=PERMIS
  o=University of Salford
  o=NHS
  o=Government
  o=Salford City Council
  o=salford
    ou=Accounting
      cn=John Smith
        x509serialNumber=1000
        x509CRLThisUpdate=20030430095031Z
      cn=Vipi Velasquez
        x509CRLThisUpdate=20030430095031Z
          x509serialNumber=1000
        x509CRLThisUpdate=20030430095546Z
          x509serialNumber=1000
          x509serialNumber=1001
        x509CRLThisUpdate=20030430100646Z
          x509serialNumber=1000
```

Attribute	Value
x509serialNumber	1001
x509CRLCertRevocationDate	20030430095546Z
objectClass	x509base
objectClass	x509CRLentry

Final Objectives

The final objectives of the project were:

1. To implement the matched values LDAPv3 control in the OpenLDAP source code
2. To implement the matched values LDAPv3 control in an LDAP client, so as to demonstrate its functionality
3. To implement attribute extraction using the PKC, CRL and AC schemas specified in [13, 16, 17] in the OpenLDAP source code
4. To progress the Internet Drafts [4, 13, 16, 17] to as near to RFC status as possible.

Final Deliverables

The final set of deliverables is as follows.

Phase	Deliverable number	Date due	Title / Description
1	D1	18 months from start	Detailed design documents [15, 18, 19] (See Appendices 8, 9,10)
3	D2	End of Project	Updated Internet RFCs or IDs for [4, 13, 16, 17] (See Appendices 4, 5, 6, 7)
3	D3	End of Project	Enhanced OpenLDAP source code
3	D4	End of Project	Project Report to TERENA (this document)

Reasons for Delays

The project was delivered approximately 14 months late. This was due to a number of factors, but primarily because we underestimated the complexity of the task and the lack of documentation for OpenLDAP. We also completely changed designs twice during the project, once at the start, and once midway through. We hopelessly underestimated the amount of time that testing would take. This was originally estimated to be 2 man months, but in reality it took nearer to 12 man months. This was due to several factors: difficulty in finding out how OpenLDAP actually worked (it wasn't until it was tested that we found it worked otherwise to our beliefs), design limitations in OpenLDAP (e.g. could only return a maximum of 500 entries), security flaws in the final design (we uncovered a denial of service attack), and non-optimum design (this was improved upon as knowledge of OpenLDAP improved). All in all we had 9 versions of the XPS Detailed Design document. Furthermore we did not provision for migrating between OpenLDAP releases. During the project we had to make 2 major migrations, from OpenLDAP 2.0 to 2.1 and 2.1 to 2.2, and many minor migrations. The implementation of MatchedValues control on both client and server sides had to be migrated from version 2.0 to 2.1. Since a lot of the new code was based on the existing code, which had been altered, it took a considerable amount of time to change the new code accordingly (around 1 man month). During the process of implementing XPS, OpenLDAP 2.1 had 20 minor version releases. They included bug fixes and small changes in design. In any case the XPS code had to be moved onto the new versions since, due to the size of the OpenLDAP project, the effects that those changes and fixes could have had on XPS could not be predicted. The only way to account for them was to test them with XPS by migrating the code and testing it. Finally, version 2.2 introduced a number of significant internal redesigns (including some code contributed by third parties). It forced changes in the XPS detailed design and corresponding changes in the code.. This consumed about 4 man months in total.

Overall the project took nearer 4 man years of effort to complete rather than the 2 man years originally budgeted for. The only reasons we managed to complete the project at all was that Mikhail Sahalayev worked for nearly a year on the project without receiving any finances from the project, and that the University of Salford paid for nearly 1 man year of extra time.

Conclusion

Despite all the complexities, design changes and code migrations, nevertheless in the end the project was completed successfully. Four Internet RFCs will be published in due course as a result of this project - one standard's track and three informational. The OpenLDAP software has for over two year included the matched values code and no significant problems have been reported with its use. Finally, the OpenLDAP XPS server code has been delivered to the OpenLDAP consortium and at the time of writing it is progressing through their quality assurance procedures ready for incorporation in a subsequent release.

References

[1] ISO 9594-8/ITU-T Rec. X.509 (2001) The Directory: Public-key and attribute certificate frameworks
[2] D.W.Chadwick. "Deficiencies in LDAP when used to support a Public Key Infrastructure", Communications of the ACM, March 2003/Vol 46, No. 3 pp. 99-104
[3] Chadwick, D.W., "Internet X.509 Public Key Infrastructure Operational Protocols – LDAPv3" Internet Draft <draft-pkix-ldap-v3-05.txt>, July 2002
[4] Chadwick, D.W., Mullan, S. "Returning Matched Values with LDAPv3", <draft-ldapext-matchedval-07.txt>, July 2003 (see Appendix 4)
[5] Chadwick, D.W., Legg, S. "Internet X.509 Public Key Infrastructure, Additional LDAP Schema for PKIs and PMIs", <draft-pkix-ldap-schema-01.txt>, September 2000.
[6] See www.openldap.org
[7] Legg, S. "Generic String Encoding Rules (GSER) for ASN.1 Types", RFC 3641, October 2003.
[8] Legg, S. "Lightweight Directory Access Protocol (LDAP) and X.500 Component Matching Rules". RFC 3687. February 2004.
[9] J. Sermersheim. "LDAP – the protocol" <draft-ietf-ldapbis-protocol-07.txt>, March 2002.
[10] Chadwick, D.W., Legg, S. "Internet X.509 Public Key Infrastructure, Additional LDAP Schema for PMIs", <draft-ietf-pkix-ldap-pmi-schema-00.txt>, 27 June 2002
[11] Chadwick, D.W., Legg, S. "Internet X.509 Public Key Infrastructure, Additional LDAP Schema for PKIs", <draft-ietf-pkix-ldap-pki-schema-00.txt>, 27 June 2002
[12] D.W.Chadwick "LDAPv3 DN strings for use with PKIs" <draft-ietf-pkix-dnstrings-00.txt>, April 2002.
[13] Gietz, P., Klasen, N. "Internet X.509 Public Key Infrastructure Lightweight Directory Access Protocol Schema for X.509 Certificates",
<draft-ietf-pkix-ldap-pkc-schema-00.txt>, August 2004 (See Appendix 5)
[14] M.Wahl, S.Kille, T. Howes."Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names", RFC 2253, Dec 1997
[15] M. Sahalayev, D. W. Chadwick. "Detailed Design of an LDAP X.509 Parsing Server", Version 1.0, 15 November 2002
[16] D.W.Chadwick, M. V. Sahalayev. "Internet X.509 Public Key Infrastructure - LDAP Schema for X.509 Attribute Certificates", <draft-ietf-pkix-ldap-ac-schema-01.txt>, July 2004 (See Appendix 6)
[17] Chadwick, D.W., Sahalayev, M. V. "Internet X.509 Public Key Infrastructure - LDAP Schema for X.509 CRLs", <draft-ietf-pkix-ldap-crl-schema-02.txt>, June 2004 (See Appendix 7)
[18] M. Sahalayev, D. W. Chadwick. "Write Ahead Log (WAL) Design", Feb 2003 (See Appendix 9)
[19] E. Ball, M. Sahalayev. "Parsing X.509 Attributes", Feb 2003 (See Appendix 10)
[20] M. Sahalayev, D. W. Chadwick. "XPS Test Plan". March 2003

Appendix 1 Detailed Progress Report

First Bimonthly Period 1 Sept - 31 Oct 2001

The original contract suggested a start date of 1 August 2001. During that month David Chadwick attended the IETF meeting in London to discuss the project with Kurt Zeilenga, the leader of the OpenLDAP project, and Steven Legg, the joint author with David Chadwick of the PKI and PMI schema Internet Draft [5]. During this meeting it was agreed to fundamentally change the way that certificate matching is to be done, to take advantage of the new Internet Draft (now an RFC) by Steven on Component Matching [7]. This will have significant and fundamental ramifications for the project as follows. Once component matching has been implemented it will mean that any new ASN.1 type definition will be able to be matched against with significantly less effort than the current project. This will mean, for example, that new certificate extensions will easily be able to be matched against. This is because the certificate matching rules will be built up of generic ASN.1 components, instead of rules built specifically for certificates. The other implication is that it will no longer be necessary to ask the TF-LSD and PKI-Coord meetings which set of certificate fields to implement first, as it should now be within the scope of the project to match on any certificate field or combination of fields. However, the implementation will now be somewhat more difficult as specific matching rules will have to be made more generic, and configuration parameters will need to be provided to dictate what is to be matched upon.

During the meeting we also learnt that someone else had already implemented certificate equality matching (task 3) and that this would be in the next release of OpenLDAP (v2.1). Until we got the release, we wouldn’t be able to say how this would impact upon our project. Kurt also advised us that extensible matching had also been implemented earlier than anticipated and that this should also be available in release 2.1.

Notwithstanding the above, it proved impossible to start the project fully in August due to vacations and the delay in appointing a new researcher to work on the project. It was agreed to amend the contract to an official start date of 1 September 2001.

During September and October half of task 2 was completed and half of task 1. The project team was very disappointed to find that there was very little documentation, and in many cases no documentation, for the OpenLDAP code that we were working on. This made the project far more difficult than was at first anticipated. However, Kurt Zeilenga assured us of his support for the project, and provided us with useful input and answers to our email questions whenever we asked him.

Second Bimonthly Report for the Period 1 Nov - 31Dec 2001

During this period, we worked on the design of component matching for certificates (tasks 1 and 6). We also worked on the design for the matched values control [4] (tasks 1 and 4). Draft designs were produced in time for the December IETF meeting.

We updated two Internet Drafts [4], [5], and attended the 52nd IETF meeting in Salt Lake City to present them to the PKIX meeting. We also met with Kurt Zeilenga to discuss the project design documents produced by our team. Kurt provided good feedback on the designs, and these were subsequently incorporated by the designers.

The team members joined the LDAP developers mailing list and emailed all their subsequent designs to the list for comment and feedback.

Third Bimonthly Report for the Period 1 Jan – 28 Feb 2002

During this period, we continued to work on the design and implementation of component matching for certificates (tasks 1, 2 and 6). As it was realised that the implementation of component matching required the detailed specification of which matching rule must apply to each component of the ASN.1 entry in the directory, a modification of the ASN.1 specification for directory entries was devised. This consisted of embedding within a conventional ASN.1 specification special comments of the matching rule viz: --{ MatchingRule }-- where MatchingRule is the name of a specified matching rule.

A grammar was written to describe this new notation and a home written ASN.1 compiler was modified to include this new grammar. A back-end to the syntax analyser of this compiler was also written to allow a compiled ASN.1 specification to be output in a convenient form for use by other programs that had been and would be developed. The first of these programs was a graphical user interface that was used to form Component Filters as specified by the RFCs of S Legg [7, 8]. This program takes the output from the ASN.1 compiler and makes a graphical tree representing an entry. Using this tree with other GUI components allows a complex component matching filter to be generated in a point/click manner. The generated filter can be output either in LDAP string notation or as ASN.1 BER. This GUI was used to form LDAP searchRequests using MatchingRuleAssertions and these requests were used to help test the behaviour of the latest development version of the OpenLDAP server (2.1 alpha). Initial modifications were made to the OpenLDAP server code to allow it to recognise and deal with such searchRequests but this work was only in its initial stages. As can be appreciated this work was extremely complex and will expected to take much longer than originally anticipated, but will produce a far better product in the end since the certificate matching will be infinitely extensible. Not only will it apply to all certificate fields and current extensions, but any new extensions should easily be catered for by feeding in the new ASN.1 definitions.

We also worked on the implementation of the matched values control [4] (task 4). We also started to modify the ldapSearch client so that it could send the matched values control (task 5). We made good progress and by the end of February we had a basic system working (complete with bugs!). We also spotted an error in the Matched Values ID, so once we finished implementing the control we reissued the ID.

We updated one Internet Drafts [3], and submitted it in time for the 53nd IETF meeting in Minneapolis. Unfortunately we weren't able to attend the meeting to present this, but we planned to attend the next meeting in July. We attended the X.500 standards meeting in Geneva (26 Feb-3 March), which has a work item on LDAP alignment, as well as updates to X.509.

Fourth Bimonthly Report for the Period 1 March – 30 Apr 2002

During this period tasks 4 and 5 (matched values [4]) were completed on schedule and the source code was sent to OpenLDAP for review. Kurt Zeilenga then organised a review of this code. The matched values ID [4] was also progressed by the ISG, and they asked for an updated version to be provided for Last Call.

The LDAPv3 RFCs were being revised so that LDAPv3 could move to draft standard status. The work was assigned by the IETF to the LDAPbis group. Considerable discussion was held during this period by the LDAPbis group about the use of ;binary encodings for the transfer of certificates, and the use of transfer encodings in general by LDAPv3. The reason for this is that there are still some interoperability problems with the PKI attributes. Kurt Zeilenga set up to two design teams during this period, to review the text in the latest revision of LDAPv3 [9]. The design teams were given different remits. Design Team A was to offer replacement text consistent with the position that ;binary must be used to indicate the transfer of binary encoded values. The team members were: Mark Smith (Lead), Steven Legg and David Chadwick. Design Team B was to offer replacement text consistent with the position that the use of ;binary is not necessary to indicate the transfer of binary encoded values. The team members were: David Chadwick (Lead), Chris Oliva and Tim Hahn. The design teams finished their work at the beginning of May and reported that, given the known interoperability problems with ;binary, and its limitations, the ;binary feature (and all mention of transfer options) should be removed from the LDAPv3 specification. The LDAPbis group were then digesting and discussing the teams' conclusion, and the final decision will be reported in the next bimonthly report. The final decision effects the contents of [5], and so publication of the next version of [5] was held up pending resolution of the ;binary issue.

During this period, we continued to work on the design of component matching for certificates (tasks 1, 2 and 6). The design was also discussed with Kurt Zielenga and Steven Legg, the author of [7] and [8]. As stated previously, this feature is extremely complex, and our latest findings are that a new subschema component will need to be defined in both the X.500 and LDAP standards. We circulated the ITU-T X.500 group with notification about this, and proposed to progress the definition of the new subschema element at the next ITU-T meeting in September in the USA.

The design continually evolved during this period as we gained more understanding of the OpenLDAP code and the detailed ramifications of component matching. For example, we realised that the embedding of a matching rule within a conventional ASN.1 specification, as reported in the last bimonthly report, was insufficient as up to 3 different matching rules may be required. We then decided to embed either an attribute type OID or LDAP syntax OID and tried to work out which would be the best one to use. The home written ASN.1 compiler will need more modifications to include this new grammar once it is finalised.

Fifth Bimonthly Report for the Period 1 May – 30 June 2002

Tasks 4 and 5 (matched values) were completed on schedule during the last reporting period by Mr Sahalaev and the source code was sent to OpenLDAP for review. Kurt Zeilenga reviewed the code, which was satisfactory, and it was included in Release 2.1 of OpenLDAP at the beginning of June 2002. David Chadwick updated the Matched Values ID [4] ready for the IETF Last Call.

As reported in the Bimonthly Report 4, the LDAPbis group decided to remove ;binary encodings from the LDAPv3 protocol [9]. We updated the LDAP PKI and PMI schema Internet Draft ready for the Yokohama IETF meeting to take account of this, and, in accordance with the decision of a previous IETF meeting the ID was split into two separate IDs [10][11]. Also release 2.1 of OpenLDAP contained certificate equality matching (task 3), and so the ID [11] was updated to ensure that it agrees with how the existing implementation works. We also wrote a new Internet Draft [12] that would allow new attribute type names used in PKI distinguished names, to be sent as encoded strings in LDAP DNs. This was circulated to the PKIX list.

Peter Gietz, Kurt Zeilenga and David Chadwick were on the MACE-DIR steering committee and during one of their regular monthly conference telephone calls in May certificate retrieval by LDAP was discussed. The discussion covered both Peter's new PKI schema approach [13] and the original PKI schema [5] and component matching [7, 8] approach that Salford were currently implementing in OpenLDAP. The outcome of these discussions were that Salford decided to ask the NRENs and TERENA if Salford should change direction and implement Peter Gietz’s new schema in a new X.509 Certificate Parsing Server implementation (XPS). David Chadwick produced a high level design specification for this, which was duly circulated to TERENA and the NRENs. The support for the new proposal was virtually unanimous, and so at the end of June Salford agreed to stop development of component matching in OpenLDAP and move to a certificate parsing server instead. The work already performed on parsing and analysing ASN.1 certificates could now be used in the latter instead of the former. Appendix 3 contains a revised project plan for producing the XPS.

SIxth Bimonthly Report for the Period 1 July – 31 August 2002

The Matched Values ID [4] successfully completed the IETF Last Call. A couple of minor changes were requested by the area director at the end of August and these were be completed during the next reporting period.

Dr Chadwick attended the IETF meeting in Yokohama in July. An informal joint meeting was held between PKIX and LDAP experts, including David Chadwick, Kurt Zeilenga, Steve Kent and Tim Polk. The prime motivation for this meeting was to discuss the PKIX requirements for the use of LDAP. The three topics discussed were the use of ;binary in attribute descriptions, the registration of additional attribute type names for use in PKI LDAP DNs [12], and how X.520 matching rules should operate on non-ASCII characters e.g. when comparing subject names in X.509 certificates. These three topics were also discussed in the LDAPbis meeting. The last topic was uncontentious and everyone agreed that more work needed to be done on this. There was considerable disagreement about the first two topics. Not all LDAP experts agreed with removing ;binary for transferring certificates, as recommended by the LDAP design teams, and considerable discussion about this continued on the LDAPBis mail list during this period. The joint meeting decided that a poll would be taken of the PKIX list, to see how existing PKI implementation use LDAPv2 and LDAPv3 to transfer certificates and CRLs. When current usage is better understood, the decision about whether to remove ;binary or not will be continued.

Concerning the registration of additional attribute type names for use in PKI LDAP DNs, this discussion was started by the ID written by David Chadwick [12]. The base LDAP document for this [14] discourages other strings to be used/registered and suggests that OIDs be used instead. However, this causes problems with PKI implementers who have to use a mixture of OIDs and LDAP strings when composing LDAP DNs. But the use of new LDAP strings may cause interworking problems until all implementations register the new strings. The meeting decided to poll the PKIX list for views, which was duly done, and there was mixed support for the idea.

During this period we started to work on the detailed design of the X.509 certificate parsing server (XPS).

Seventh Bimonthly Report for the Period 1 September – 31 October 2002

It turned out that in order to write the detailed design for the X509 Parsing Server (XPS), we had to do some coding in OpenLDAP in order to discover how things actually worked. Thus the detailed design and coding ran in parallel somewhat since the beginning of September. The detailed design was almost complete and it was anticipated that it would be distributed to the steering committee by 15 November.

The following coding was completed:

- the configuration file
- the add and delete operations for an internal LDAP server,
- the extraction and creation of child entries for certificates.

We also pointed out some deficiencies in the schema document [13] and fed these back to Peter Gietz.

The revised version of Matched Values [4] was sent to the ID editor on 7 September for progression as an RFC.

Dr Chadwick attended the ITU-T X.500/509 standard's meeting at NIST in Gaithersberg in September. One item on the agenda was X.500 alignment with LDAP. During this meeting a questionnaire was composed and circulated to the IETF PKIX list requesting their input on the use of the LDAP protocols (v2 and v3) for certificate storage and retrieval. This was designed to help the IETF groups in their work on how to make LDAPv3 give better support to PKIs.

Eight Bimonthly Report for the Period 1 November – 31 December 2002

The detailed design for XPS [15] was completed and was distributed to the steering committee on 15 November.

We attended the Terena TF-ACE meeting in Bromma on 25-27 November, where we presented the results of the project so far. We also had a discussion of the Detailed Design document. This was followed by more detailed email comments about the design, which were very helpful, and as a result, an updated design was published on 28 November.

We continued to code up the detailed design, but due to the complexity of the design, we were running an estimated 1 month behind the schedule in Appendix 3.

We produced internal LDAP schema IDs for CRLs and ACs (which were subsequently issued as Internet Drafts [16, 17] during the next period).

Ninth Bimonthly Report for the Period 1 January – 28 February 2003

We revised our internal LDAP schema IDs for CRLs and ACs, by adding more features from X.509, and merging common components with those from Klasen [13]. These were duly submitted to the Internet Editor [16][17], so that they could be discussed at the San Francisco IETF meeting.

We also proposed many changes to the Klasen ID [13], to bring it into align with the new CRL and AC IDs. These were mostly accepted by its authors, and a revised version (02.txt) was published by them in March 2003. The authors also agreed to meet during the 56th IETF meeting in San Francisco to resolve any final discrepancies. It was then planned that Last Call versions could be issued in time for the next IETF meeting.

As a result of the new schema IDs [16] [17], four new versions of the Detailed Design [15] were issued during February (versions 1.2, 1.3. 1.4 and 1.5).

Version 1.2 introduced a major change to the design, as it was realised that a CRL could produce a whole subtree of LDAP entries, rather than a single entry, as in the original design. This is because each revoked certificate (CRL entry) comprises: its serial number, revocation date and optional CRL entry extensions such as the revocation reason. Thus in order to be able to search for particular revoked certificates with particular entry extensions, each revoked certificate in the CRL has to be held in a separate LDAP entry. We therefore added a configuration parameter to allow administrators to switch this subtree feature on or off.

Simultaneously with issuing the revised detailed design, we also published the design of the Write Ahead Log [18], which provides XPS with its rollback and recovery capability.

Version 1.3 of the Detailed Design corrected a minor flaw in the 1.2 design, concerning how the revoked certificate entries are to be named. Other minor editorial corrections were made.

Version 1.4 added a reference to the design about how the X.509 attributes are parsed and turned into LDAP attributes. This additional parsing document [19] describes how the ASN.1 X.509 attributes are handled, and how the parsing implementation can be modified to handle additional X.509 certificate and CRL extensions.

In version 1.5 we enhanced the design by adding a configuration file that maps the LDAP attributes into elements of the X.509 ASN.1 attributes. This allows an administrator of the XPS server to (a) determine precisely which LDAP attributes he wishes to support, and (b) as new certificate and CRL extensions and their corresponding LDAP attributes are defined, update the configuration file to incorporate them.

We continued to code up the revised detailed design, but due to the complexity of the design, and the addition of new features during this period, we estimated that we were running approximately 2 months behind schedule. This means that the development would not be completed until early March 2003, after which testing can start.

Tenth Bimonthly Report for the Period 1 March – 30 April 2003

David Chadwick attended the San Francisco IETF meeting and presented the new PKIX LDAP schema IDs [16][17]. Prior to this, he met informally with Peter Gietz, the author of [13], and they resolved the final discrepancies between their IDs. As a result of their discussions, a new configuration parameter was added to XPS to list the X.509 attribute types that would be trapped, and version 1.6 of the detailed design was released [15].

Prior to the PKIX meeting, an informal PKIX-LDAP meeting was arranged by Tim Polk the chair of PKIX. It was attended by Russ Housley (the new security Area Director), Peter Gietz, Tim Polk, Kurt Zeilenga, Bob Morgan, Steven Legg and David Chadwick. The meeting agreed that component matching (the original design for this project) is technically more elegant and preferable to splitting a certificate up into separate attributes for searching, as in the current design. However, the meeting agreed that vendors seem to be reluctant to implement this, and splitting the attributes up is a short term pragmatic solution that does not need LDAP vendors to change their servers. Consequently the meeting agreed that component matching should be progressed as a standards track RFC, and that this might encourage vendors in the longer term to implement this solution; whilst the extraction IDs from Gietz, Chadwick et al should be progressed as Experimental or Informational RFCs if the area directors agree to this.
(Note that prior to the meeting the ADs did agree to issuing the IDs as Informational RFCs, but not until after the component matching IDs have been published as standard track RFCs.)

On Friday of the same week, David Chadwick attended an OpenLDAP developers meeting, and gave a presentation about this project. At this meeting, Kurt announced that it should be possible to incorporate the certificate matching code into Release 2.2 of OpenLDAP, due to be released in September 2003.

We continued to code up the revised detailed design, and this was broadly finished by the end of March. One problem we encountered, was that prior to a Delete operation we need to Search the backend to see how many certificate entries there are. However, the current be->be_search primitive returns the search result entries directly to the client, and not to the caller. This meant we needed to modify the send_search_entry function to trap the results, but Kurt Zeilenga was not able to advise us of the best way to do this until the beginning of May.

A pre-release version of the code was distributed to Bige Nikita, a Programmer System Administrator from Signal-com, Moscow, Russia, (see http://www.signal-com.ru/eng/about/index.html), who contacted Mikhail to say he needed the PKC features that we were providing for a project he was working on. He tested some of our code and was able to parse and store public key certificates in his OpenLDAP server.

At this stage of the project we were just at the beginning of rigorous testing and debugging. We produced a Test Plan [20]. This would require us to creating a whole range of PKCs, ACs and CRLs containing various extensions and fields, as an aid to the debugging, so that as many options as possible could be tested. The code is complex, and the extensions are numerous, so there is a lot of testing to do. We anticipated that the testing would take approximately 3 months, so the project should be completed by the end of July.

Report for the Period May 2003-June 2004

May-June

Kurt Zeilenga was not able to advise us of the best way to trap search results until the beginning of May. We then set about working out the precise details of how to do this. This was complex, and took us approximately one month to sort out. Consequently we not able to start full debugging and testing until the middle of June. However, during this period more was learnt about the internal functioning of the OpenLDAP code, and this lead to a number of design changes that made the XPS code more efficient. In particular, we removed the specific operations that called an external server (for the case where XPS was to front another LDAP backend server) and used native OpenLDAP functionality instead (the back-ldap backend servers functionality). We also removed the LDAP API functions from XPS as they were no longer used. The back-ldap was used instead. Version 1.7 of the Detailed Design was issued in May. This also included a new option for ASN.1 ->X.509 attribute types mapping.

July-August 2003

Earlier in the year we produced a paper about the project, and we found that it was accepted for presentation at the IFIP Database and Applications Security conference in Colorado in August. (See http://www.cs.colostate.edu/~ifip03/). Dr Chadwick attended the conference and presented the paper.

Mikhail continued debugging the code throughout the summer. During the debugging we uncovered a potential denial of service attack, whereby a malicious user could submit a spurious XPS delete attribute operation on the root (or close to the root) of the DIT. This would cause a full subtree Search to be performed, which obviously would consume huge resources. We thus had to redesign the Delete operation to do a series of 1 level searches instead of a full subtree search. This was obviously more complicated but should stop most denial of service attacks from happening since any spurious search would be killed after the first level was completed. A new Detailed Design (v1.8) was released in August.

September-October 2003

We finished the new coding in September and resumed the debugging. At this point in time we anticipated the testing/debugging would finish by the end of November. However, the next release of OpenLDAP came out in September, so after we have finished the debugging we will then need to migrate to the latest development release of OpenLDAP. Thus we anticipated that we would finish the project by the end of 2003.

November-December 2003

In order to expedite debugging a second programmer was brought onto the project to do code inspections. This worked well, and 20 new bugs were found in the code. However the sole remaining original developer (Mikhail) was now working for no pay, since the project budget was long since exhausted. Consequently he had to take part time demonstrating work at the university in order to eke out a living. Debugging was therefore taking longer than anticipated.

January-February 2004

Adds and Deletes were now fully debugged but there were still some bugs in the Modify that needed to be fixed. In addition, the code experienced a number of memory leaks, so that in soak testing it would work for maybe a dozen times or so, and then would hang or crash. Locating and eliminating all of the memory leaks took nearly two months, and required a significant re-write of the ASN.1 parsing code. Since the original ASN.1 programmer had now left the university, it took Mikhail significantly longer to do this than it would have taken the original programmer. Once the code was debugged it was sent to a programmer in Moscow for testing, and he reported several more bugs which were duly fixed.

March-April 2004

The fully debugged code in OpenLDAP2.1 was migrated to the latest development release of OpenLDAP 2.2, (branch 2.2.11) so that it could be incorporated into the next public release. However, OpenLDAP had undergone a number of significant changes, so that the migration was not straight forward. For example, the development release had now changed the matching rule for Modify and Delete values to exact matching from binary match. Consequently we needed to publish a revised Detailed Design document (v.1.8) in April. Furthermore the OpenLDAP development code had a number of major bugs in it, which meant that even when XPS was fully debugged, the complete code would not run properly for any significant period of time. One of the major problems is that OpenLDAP has a built in limit to the number of internal search results that it can return (currently 500). This means that if XPS is being used to store CRL subtrees, and the CRL has more than 500 entries in it, it causes problems. It is expected that once the OpenLDAP bugs are fixed, XPS will work as expected.

May-June 2004

The code was finally released to the OpenLDAP development team in mid May. It then has to undergo QA inspections by Kurt Zeilenga et al before it can be scheduled for inclusion in the next OpenLDAP release (tentatively scheduled for Sept 2004).

We successfully demonstrated XSP at the Terena Networking Conference in Greece in June.

July-August 2004

We published final versions of the Internet Drafts [16] [17] and presented them along with [13] at the IETF meeting in San Diego in August. It was agreed at this meeting that all 3 could proceed to Last Call in September and then be finalised and sent to the RFC editor at the Washington meeting in November.

We produced the final report.

Appendix 2 Original Planned Tasks and Resources

Task 1. Familiarisation with OpenLDAP code, ASN.1 toolkits, JNDI toolkits, Certificate parsing and field extraction etc. **3 man months**
Task 2. Write a module to parse a certificate and extract the fields that a user is likely to filter on. Add these fields to the indexes in the OpenLDAP source code, to aid the rapid searching for particular certificates. **1 man month**
(Note. The exact set of fields will be chosen by talking to the Terena TF-LSD and PKI-COORD groups and determining their user requirements.)
Task 3. Simple Searching. Implement the certificate EQUALITY matching rule. **1 man month**
Task 4. Implement matched values control in OpenLDAP as a new module. **3 man months**
Task 5. Modify an LDAP client so that it can a) pass the new matched values control in the protocol, and b) allow the user to ask for this control in a user friendly manner. We will attempt to do this in Netscape or MS LDAP client, but if it proves too difficult, we will build our own simple Java client or extend the OpenLDAP ldapsearch(1) client. **3 man months**.
Task 6. Complex searching. Implement the new extensible certificateMatch matching rule. This will extract the new filter items from the incoming Search, will scan the new indexes looking for a match and will find the entry with the correct certificate. Return the entry. **3 man months.**
Note. Kurt Zeilenga advises that the OpenLDAP extensible matching infrastructure is still immature, so placing this task after tasks 4 and 5 rather than before them, which would seem more natural, will give him time to stabilise this code.
Task 7. Modify an LDAP client so that it can a) pass the new certificate filter item in the protocol and b) allow the user to type in the request in a user friendly manner. We will attempt to do this in Netscape or MS LDAP client, but if it proves too difficult, we will either build our own simple Java client or extend the OpenLDAP ud(1) client. **3 man months**.
Task 8. Update the Internet Drafts as the implementation proceeds, and progress these through the IETF. **0.5 man months**
Task 9. Contingency 20% **3 man months**
Task 10. Production of Final Report for Terena. **0.5 man months**
Taks 11. Project Management 10% **2 man months**
Total 23 man months

Appendix 3 Revised Project Plan

Task 1. Produce a detailed design for the X.509 Certificate Parsing Server (XPS). (July-Sept 2002)
Task 2. Implement the XPS server (Oct-Dec 2002)
Task 3. Test the XPS server (Jan-Feb 2003)
Task 4. Write and progress new PKIX LDAP schemas for the XPS (Oct-April 2003)
Task 5. Write Final Terena report plus Contingency (March 2003)

Appendix 4. Returning Matched Values with LDAPv3

```
Internet-Draft                                     David Chadwick
LDAPExt WG                                   University of Salford
Intended Category: Standards Track                     Sean Mullan
                                                 Sun Microsystems
Expires: 16 January 2004                              16 July 2003
```

Returning Matched Values with LDAPv3
<draft-ietf-ldapext-matchedval-07.txt>

STATUS OF THIS MEMO

Comments and suggestions on this document are encouraged. Comments on this document should be sent to the LDAPEXT working group discussion list:

ietf-ldapext@netscape.com

or directly to the authors.

ABSTRACT

This document describes a control for the Lightweight Directory Access Protocol version 3 that is used to return a subset of attribute values from an entry, specifically, only those values that match a "values return" filter. Without support for this control, a client must retrieve all of an attribute's values and search for specific values locally.

1. Introduction

When reading an attribute from an entry using the Lightweight Directory Access Protocol version 3 (LDAPv3) [2], it is normally only possible to read either the attribute type, or the attribute type and all its values. It is not possible to selectively read just a few of the attribute values. If an attribute holds many values, for example, the userCertificate attribute, or the subschema publishing operational attributes objectClasses and attributeTypes [3], then it may be desirable for the user to be able to selectively retrieve a subset of the values, specifically, those attribute values that match some user defined selection criteria. Without the control specified in this document a client must read all of the attribute's values and filter out the unwanted values, necessitating the client to implement the matching rules. It also requires the client to potentially read and process many irrelevant values, which can be inefficient if the values are large or complex, or there are many values stored per attribute.

This document specifies an LDAPv3 control to enable a user to return only those values that matched (i.e. returned TRUE to) one or more elements of a newly defined "values return" filter. This control can be especially useful when used in conjunction with extensible matching rules that match on one or more components of complex binary attribute values.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [4].

2. The valuesReturnFilter Control

The valuesReturnFilter control is either critical or non-critical as determined by the user. It only has meaning for the Search operation, and SHOULD only be added to the Search operation by the client. If the server supports the control and it is present on a Search operation, the server MUST obey the control regardless of the value of the criticality flag.

If the control is marked as critical, and either the server does not support the control or the control is applied to an operation other than Search, then the server MUST return an unavailableCriticalExtension error. If the control is not marked as critical, and either the server does not support the control or the control is applied to an operation other than Search, then the server MUST ignore the control.

The object identifier for this control is 1.2.826.0.1.3344810.2.3.

The controlValue is an OCTET STRING, whose value is the BER encoding [15], as per Section 5.1 of RFC 2251 [2], of a value of the ASN.1 [14] type ValuesReturnFilter.

```
        ValuesReturnFilter ::= SEQUENCE OF SimpleFilterItem

        SimpleFilterItem ::= CHOICE {
                equalityMatch   [3] AttributeValueAssertion,
                substrings      [4] SubstringFilter,
                greaterOrEqual  [5] AttributeValueAssertion,
                lessOrEqual     [6] AttributeValueAssertion,
                present         [7] AttributeDescription,
                approxMatch     [8] AttributeValueAssertion,
                extensibleMatch [9] SimpleMatchingAssertion }

         SimpleMatchingAssertion ::= SEQUENCE {
                matchingRule    [1] MatchingRuleId OPTIONAL,
                type            [2] AttributeDescription OPTIONAL,
--- at least one of the above must be present
                matchValue      [3] AssertionValue}
```

All the above data types have their standard meanings as defined in [2].

If the server supports this control, the server MUST make use of the control as follows:

(1) The Search Filter is first executed in order to determine which entries satisfy the Search criteria (these are the filtered entries). The control has no impact on this step.

(2) If the typesOnly parameter of the Search Request is TRUE, the control has no effect and the Search Request is processed as if the control had not been specified.

(3) If the attributes parameter of the Search Request consists of a list containing only the attribute with OID "1.1" (specifying that no attributes are to be returned), the control has no effect and the Search Request is processed as if the control had not been specified.

(4) For each attribute listed in the attributes parameter of the Search Request, the server MUST apply the control as follows to each entry in the set of filtered entries:

i) Every attribute value that evaluates TRUE against one or more elements of the ValuesReturnFilter is placed in the corresponding SearchResultEntry.
ii) Every attribute value that evaluates FALSE or undefined against all elements of the ValuesReturnFilter is not placed in the corresponding SearchResultEntry. An attribute that has no values selected is returned with an empty set of vals.

Note. If the AttributeDescriptionList is empty or comprises "*" then the control MUST be applied against every user attribute. If the AttributeDescriptionList contains a "+" then the control MUST be applied against every operational attribute.

3. Relationship to X.500

The control is a superset of the matchedValuesOnly (MVO) boolean of the X.500 Directory Access Protocol (DAP) [5] Search argument, as amended in the latest version [6]. Close examination of the matchedValuesOnly boolean by the LDAP Extensions (LDAPEXT) Working Group revealed ambiguities and complexities in the MVO boolean that could not easily be resolved. For example, it was not clear if the MVO boolean governed only those attribute values that contributed to the overall truth of the filter, or all of the attribute values even if the filter item containing the attribute evaluated to false. For this reason the LDAPEXT group decided to replace the MVO boolean with a simple filter that removes any uncertainty as to whether an attribute value has been selected or not.

4. Relationship to other LDAP Controls

The purpose of this control is to select zero, one or more attribute values from each requested attribute in a filtered entry, and to discard the remainder. Once the attribute values have been discarded by this control they MUST NOT be re-instated into the Search results by other controls.

This control acts independently of other LDAP controls such as server side sorting [10] and duplicate entries [7]. However, there might be interactions between this control and other controls so that a different set of Search Result Entries are returned, or the entries are returned in a different order, depending upon the sequencing of this control and other controls in the LDAP request. For example, with server side sorting, if sorting is done first, and value return filtering second, the set of Search Results may appear to be in the wrong order since the value filtering may remove the attribute values upon which the ordering was done. (The sorting document specifies that entries without any sort key attribute values should be treated as coming after all other attribute values.) Similarly with duplicate entries, if duplication is performed before value filtering, the set of Search Result Entries may contain identical duplicate entries, each with an empty set of attribute values, because the value filtering removed the attribute values that were used to duplicate the results.

For these reasons the ValuesReturnFilter control in a SearchRequest SHOULD precede other controls that affect the number and ordering of SearchResultEntrys.

5. Examples

All entries are provided in LDAP Data Interchange Format (LDIF)[8].

The string representation of the valuesReturnFilter in the examples below uses the following ABNF [12] notation:

```
 valuesReturnFilter = "(" 1*simpleFilterItem ")"
 simpleFilterItem = "(" item ")"
```

where item is as defined below (adapted from RFC2254 [11]).

```
        item       = simple / present / substring / extensible
        simple     = attr filtertype value
        filtertype = equal / approx / greater / less
        equal      = "="
        approx     = "~="
        greater    = ">="
        less       = "<="
        extensible = attr [":" matchingrule] ":=" value
                     / ":" matchingrule ":=" value
        present    = attr "=*"
        substring  = attr "=" [initial] any [final]
        initial    = value
        any        = "*" *(value "*")
        final      = value
        attr       = AttributeDescription from Section 4.1.5 of [1]
        matchingrule = MatchingRuleId from Section 4.1.9 of [1]
        value      = AttributeValue from Section 4.1.6 of [1]
```

(1) The first example shows how the control can be set to return all attribute values from one attribute type (e.g. telephoneNumber) and a subset of values from another attribute type (e.g. mail).

The entries below represent organizationalPerson object classes located somewhere beneath the distinguished name dc=ac,dc=uk.

```
dn: cn=Sean Mullan,ou=people,dc=sun,dc=ac,dc=uk
cn: Sean Mullan
sn: Mullan
objectClass: organizationalPerson
objectClass: person
objectClass: inetOrgPerson
mail: sean.mullan@hotmail.com
mail: mullan@east.sun.com
telephoneNumber: + 781 442 0926
telephoneNumber: 555-9999

dn: cn=David Chadwick,ou=isi,o=salford,dc=ac,dc=uk
cn: David Chadwick
sn: Chadwick
objectClass: organizationalPerson
objectClass: person
objectClass: inetOrgPerson
mail: d.w.chadwick@salford.ac.uk
```

An LDAP search operation is specified with a baseObject set to the DN of the search base (i.e. dc=ac,dc=uk), a subtree scope, a filter set to (sn=mullan), and the list of attributes to be returned set to "mail,telephoneNumber" or "*". In addition, a ValuesReturnFilter control is set to ((mail=*hotmail.com)(telephoneNumber=*))

The search results returned by the server would consist of the following entry:

```
dn: cn=Sean Mullan,ou=people,dc=sun,dc=ac,dc=uk
mail: sean.mullan@hotmail.com
telephoneNumber: + 781 442 0926
telephoneNumber: 555-9999
```

Note that the control has no effect on the values returned for the "telephoneNumber" attribute (all of the values are returned), since the control specified that all values should be returned.

(2) The second example shows how one might retrieve a single attribute type subschema definition for the "gunk" attribute with OID 1.2.3.4.5 from the subschema subentry

Assume the subschema subentry is held below the root entry with DN cn=subschema subentry,o=myorg and this holds an attributeTypes operational attribute holding the descriptions of the 35 attributes known to this server (each description is held as a single attribute value of the attributeTypes attribute).

```
dn: cn=subschema subentry,o=myorg
cn: subschema subentry
objectClass: subschema
attributeTypes: ( 2.5.4.3 NAME 'cn' SUP name )
attributeTypes: ( 2.5.4.6 NAME 'c' SUP name SINGLE-VALUE )
attributeTypes: ( 2.5.4.0 NAME 'objectClass' EQUALITY obj
 ectIdentifierMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.38 )
attributeTypes: ( 2.5.18.2 NAME 'modifyTimestamp' EQUALITY gen
 eralizedTimeMatch ORDERING generalizedTimeOrderingMatch SYN
 TAX 1.3.6.1.4.1.1466.115.121.1.24 SINGLE-VALUE NO-USER-
 MODIFICATION USAGE directoryOperation )
```

```
attributeTypes: ( 2.5.21.6 NAME 'objectClasses' EQUALITY obj
 ectIdentifierFirstComponentMatch SYNTAX 1.3.
 6.1.4.1.1466.115.121.1.37 USAGE directoryOperation )
attributeTypes: ( 1.2.3.4.5 NAME 'gunk' EQUALITY caseIgnoreMat
 ch SUBSTR caseIgnoreSubstringsMatch SYNTAX 1.3.
 6.1.4.1.1466.115.121.1.44{64} )
attributeTypes: ( 2.5.21.5 NAME 'attributeTypes' EQUALITY obj
 ectIdentifierFirstComponentMatch SYNTAX 1.3.
 6.1.4.1.1466.115.121.1.3 USAGE directoryOperation )
```

plus another 28 - you get the idea.

The user creates an LDAP search operation with a baseObject set to cn=subschema subentry,o=myorg, a scope of base, a filter set to (objectClass=subschema), the list of attributes to be returned set to "attributeTypes", and the ValuesReturnFilter set to ((attributeTypes=1.2.3.4.5))

The search result returned by the server would consist of the following entry:

```
dn: cn=subschema subentry,o=myorg
attributeTypes: ( 1.2.3.4.5 NAME 'gunk' EQUALITY caseIgnoreMat
 ch SUBSTR caseIgnoreSubstringsMatch SYNTAX 1.3.
 6.1.4.1.1466.115.121.1.44{64} )
```

(3) The final example shows how the control can be used to match on a userCertificate attribute value. Note that this example requires the LDAP server to support the certificateExactMatch matching rule defined in [9] as the EQUALITY matching rule for userCertificate.

The entry below represent a pkiUser object class stored in the directory.

```
dn: cn=David Chadwick,ou=people,o=University of Salford,c=gb
cn: David Chadwick
objectClass: person
objectClass: organizationalPerson
objectClass: pkiUser
objectClass: inetOrgPerson
sn: Chadwick
mail: d.w.chadwick@salford.ac.uk
userCertificate;binary: {binary representation of a certificate with
a serial number of 2468 issued by o=truetrust ltd,c=gb}
userCertificate;binary: {binary representation of certificate with a
serial number of 1357 issued by o=truetrust ltd,c=gb}
userCertificate;binary: {binary representation of certificate with a
serial number of 1234 issued by dc=certsRus,dc=com}
```

An LDAP search operation is specified with a baseObject set to o=University of Salford,c=gb, a subtree scope, a filter set to (sn=chadwick) and the list of attributes to be returned set to "userCertificate;binary". In addition, a ValuesReturnFilter control is set to ((userCertificate=1357$o=truetrust ltd,c=gb)).

The search result returned by the server would consist of the following entry:

```
dn: cn=David Chadwick,ou=people,o=University of Salford,c=gb
```

userCertificate;binary: {binary representation of certificate with a serial number of 1357 issued by o=truetrust ltd,c=gb}

6. Security Considerations

This document does not primarily discuss security issues.

Note however that attribute values MUST only be returned if the access controls applied by the LDAP server allow them to be returned, and in this respect the effect of the ValuesReturnFilter control is of no consequence.

Note that the ValuesReturnFilter control may have a positive effect on the deployment of public key infrastructures. Certain PKI operations, like searching for specific certificates, become more practical when combined with X.509 certificate matching rules at the server, and more scalable, since the control avoids the downloading of potentially large numbers of irrelevant certificates which would have to be processed and filtered locally (which in some cases is very difficult to perform).

7. IANA Considerations

Registrigration of the Matched Values control as an LDAP Protocol Mechanism [16] is requested:

```
    Subject: Request for LDAP Protocol Mechanism Registration

    Object Identifier: 1.2.826.0.1.3344810.2.3
    Description: Matched Values Control
    Person & email address to contact for further information:
      David Chadwick <d.w.chadwick@salford.ac.uk>
    Usage: Control
    Specification: RFCxxxx
    Author/Change Controller: IESG
    Comments: none
```

This document uses the OID 1.2.826.0.1.3344810.2.3 to identify the matchedValues control described here. This OID was assigned by TrueTrust Ltd, under its BSI assigned English/Welsh Registered Company number [13].

8. Acknowledgements

The authors would like to thank members of the LDAPExt list for their constructive comments on earlier versions of this document, and in particular to Harald Alvestrand who first suggested having an attribute return filter and Bruce Greenblatt who first proposed a syntax for this control.

9. Copyright

10. References

Normative

[1] S. Bradner. "The Internet Standards Process -- Revision 3", RFC 2026, October 1996.
[2] M. Wahl, T. Howes, S. Kille, "Lightweight Directory Access Protocol (v3)", Dec. 1997, RFC 2251
[3] M. Wahl, A. Coulbeck, T. Howes, S. Kille, "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions", RFC 2252, Dec 1997
[4] S.Bradner. "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.
[14] ITU-T Recommendation X.680 (1997) | ISO/IEC 8824-1:1998, Information Technology - Abstract Syntax Notation One (ASN.1): Specification of Basic Notation
[15] ITU-T Recommendation X.690 (1997) | ISO/IEC 8825-1,2,3:1998 Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)
[16] K. Zeilenga. "IANA Considerations for LDAP" RFC 3383, September 2002

Informative

[5] ITU-T Rec. X.511, "The Directory: Abstract Service Definition", 1993.
[6] Draft ISO/IEC 9594 / ITU-T Rec X.511 (2001) The Directory: Abstract Service Definition.
[7] J. Sermersheim. "LDAP Control for a Duplicate Entry Representation of Search Results", Internet Draft <draft-ietf-ldapext-ldapv3-dupent-08.txt>, September 2002.
[8] G. Good. "The LDAP Data Interchange Format (LDIF) - Technical Specification". RFC 2849, June 2000.
[9] D. Chadwick, S.Legg. "Internet X.509 Public Key Infrastructure - Additional LDAP Schema for PKIs", Internet Draft <draft-pkix-ldap-pki-schema-00.txt>, June 2002
[10] T. Howes, M. Wahl, A. Anantha, "LDAP Control Extension for

Server Side Sorting of Search Results", RFC 2891, August 2000
[11] T. Howes. "The String Representation of LDAP Search Filters". RFC 2254, December 1997.
[12] D. Crocker, Ed. "Augmented BNF for Syntax Specifications: ABNF." RFC 2234. November 1997.
[13] BRITISH STANDARD BS 7453 Part 1. Procedures for UK Registration for Open System Standards Part 1: Procedures for the UK Name Registration Authority.

11. Authors Addresses

David Chadwick
IS Institute
University of Salford
Salford M5 4WT
England

Email: d.w.chadwick@salford.ac.uk
Tel: +44 161 295 5351

Sean Mullan
Sun Microsystems
East Point Business Park
Dublin 3
Ireland
Tel: +353 1 853 0655
Email: sean.mullan@sun.com

12. Changes since version 2

i) Revised the examples to be more appropriate
ii) Section on interactions with other LDAP controls added
iii) Removed Editor's note concerning present filter
iv) Tightened wording about its applicability to other operations and use of criticality field

Changes since version 3

i) Mandated that at least one of type and matchingRule in simpleMatchingAssertion be present
ii) Fixed LDIF mistakes in the examples
iii) Additional minor editorials only

Changes since version 4

i) corrected the ABNF for single items of valuesReturnFilter

Changes since version 5

i) added some adapted BNFL from [11] into the examples (specifically the [":dn"] component was removed)
ii) general editorial tidying up prior to Last Call

Changes since version 6

i) First example had all attributes (*) added to it
ii) References to ASN.1 standards added
iii) Syntax error in third example corrected
iv) IANA considerations section added

Appendix 5. LDAP Schema for X.509 Certificates

```
Network Working Group                                           P. Gietz
Internet-Draft                                  DAASI International GmbH
Expires: November 30, 2004                                     N. Klasen
                                                                  Avinci
                                                               June 2004
```

Internet X.509 Public Key Infrastructure Lightweight Directory Access Protocol Schema for X.509 Certificates
draft-ietf-pkix-ldap-pkc-schema-00

Status of this Memo

Copyright Notice

Abstract

This document describes an LDAP schema which can be used to implement a certificate store for X.509 certificates. Specifically, two structural object classes for X.509 user and CA certificates are defined. Key fields of a certificate are stored in LDAP attributes so that applications can easily retrieve the certificates needed by using basic LDAP search filters. Multiple certificates for a single entity can be stored and retrieved.

Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The following syntax specifications use the augmented Backus-Naur Form (ABNF) as described in [RFC2234].

Schema definitions are provided using LDAPv3 description formats [RFC2252]. Definitions provided here are formatted (line wrapped) for readability.

Table of Contents

1. Introduction

A key component in the wide-spread adoption of a Public Key Infrastructure is the general availability of public keys and their certificates. Today, certificates are often published in an X.500 compliant directory service. These directories are accessed by applications using the LDAP v3 [RFC3377] protocol. An LDAPv3 schema for PKI repository objects is specified in [pkix-ldap-schema], where a set of object classes, attribute types, syntaxes, and extended matching rules are defined. For storing certificates, the "userCertificate" and "cACertificate" attribute types are used. All certificates of an entity are stored as values in these multi-valued attributes. This solution has a serious drawback. In LDAP, the smallest granularity of data access is the attribute. The directory server will therefore always return the full list of certificates of an entry to clients dealing with certificates. If the number of certificates for an entity is large this will result in considerable overhead and burden to the client.

This document proposes to solve this problem by the use of the structural object classes x509userCertificate and x509caCertificate for storing certificates. Each certificate will be stored in a separate entry in the directory. Having each certificate stored in a separate entry provides flexibility in structuring the Directory Information Tree. The certificate entries can be stored either below a person entry or below a CA entry as a certificate only repository, as shown in figure 1.

1.) below Person entry:

```
                            person
                           /  |    \
                          /   |     \
                     cert1  cert2   cert3
```

2.) below CA cert repository:

```
                            CA
                             |
                             |
                      certificate repository
                     /       |          \
                    /        |           \
                 cert1     cert2 ...    cert1008
```

Figure 1: examples of possible DIT-structures

Fields of certificates which are needed to identify a certificate and those which are often used in searching for an appropriate certificate, are extracted from the certificate and stored as attributes of the entry. Applications can thus search for specific certificates with simple LDAP filters. This approach could be named a metadata approach, since data (attributes) about data (certificate) are stored.

The use of simple attributes also makes a large scale widely distributed certificate repository service possible by using an indexing service based on The Common Indexing Protocol (CIP) [RFC2651], which defines a protocol between index servers for exchanging index objects in order to facilitate query routing. The Tagged Index Object format as specified in [RFC2654] was specified to carry directory server information, by collecting the single attributetypes and values. By using the schema proposed in this document, index objects can include certificate information in attributes.

If certificates are stored redundantly in person entries and in certificate entries below the person entries, maintainers of repositories MUST make sure that the same certificates are stored in the person entry and the respective certificate entries and keep this consistency. Alternatively they MUST leave out any certificates in the person entry.

This document is one of a set following this approach comprising:
1. the LDAP schema for X.509 public key certificates (this document)
2. the LDAP schema for X.509 attribute certificates [ldap-ac-schema]
3. the LDAP schema for X.509 CRLs [ldap-crl-schema]

Future documents may be written that use the same method for Qualified certificates as described in [RFC3039] or any other evolving pkix certificate standard. An auxiliary object class for including additional metadata that are not included in the certificate is also outside the scope of this document.

Two alternative approaches are discussed in the next two sections.

2. Comparison with Values Return Filter Control

In [matchedval] a control has been defined that allows for only a subset of values of a specified attribute to be returned from a matching entry, by defining a filter for the returned values. In this section, this approach is compared with the one proposed in this document.

The major benefit of the Values Return Filter Control is that it does not require any changes to the DIT.

While it is a simple matter to modify the DIT in such a way that all certificate information is removed from the entries and placed in the container directly beneath the entries according to the definitions of this specification, it is less simple to simultaneously modify all of the applications that depend on certificates being stored in the entry. Thus, it may be desirable to duplicate the certificate information, by having it appear in the entry, as well as in the container beneath the entry for a short period of time, in order to allow for migration of the applications to the new LDAP schema. As in any situation in which information is duplicated, great care must be taken in order to ensure the integrity and consistency of the information.

There are several advantages in using the x509certificate object class. No special matching rules are needed to retrieve a specific certificate. Any field in the certificate can be used in the search filter. Even information that doesn't appear in the certificate can be used in a search filter. It is easier to remove certificates from the DIT, since the entire certificate BER/DER encoding does not have to be supplied in the modify operation. Searches that don't need extensible matching rules and Values Return Filter Control will perform faster.

Another advantage of the solution proposed here is that it will not be necessary to modify existing server implementations to support this schema. The extended matching rules proposed in [pkix-ldap-schema] would require substantial changes in the servers' indexing mechanisms. In contrast, servers implementing the x509certificate schema can easily leverage their indexing support for standard LDAPv3 syntaxes.

A CIP-based indexing system for a wide scale distributed certificate repository will rather be possible by using the solution proposed here due to its dependency on attribute values.

3. Comparison with Component Matching approach

[RFC3687] defines a new mechanism for matching in complex syntaxes, by defining generic matching rules that can match any user selected component parts in an attribute value of any arbitrarily complex attribute syntax. We believe that this might be the proper way to solve search problems in the longer term, but that it will take a long time until such ASN.1 based mechanisms will be implemented in LDAP servers and clients. Even if this has happened the mechanism proposed here, will still be useful in the frame of CIP. A simple and easy to implement mechanism is needed today and this is what this memo wants to provide.

4. The attribute types of the x509certificate object classes

The description of all attributes with relevance to fields and extensions of an X.509 certificate include a respective reference to [X.509-2000] and to [RFC3280].

4.1 Attributes for mandatory fields of an X.509 certificate

4.1.1 X.509 version

X.509 Version of the encoded certificate (See X.509(2000) 7, RFC3280 4.1.2.1.) or of the CRL.

```
( 1.3.6.1.4.1.10126.1.5.3.1
      NAME 'x509version'
      DESC 'X.509 Version of the certificate, or of the CRL'
      EQUALITY integerMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
      SINGLE-VALUE )
```

Values of this attribute may either be 0, 1, 2 or 3 corresponding to X.509 v1, v2, v3, or v4.

4.1.2 Serial number

The serial number is an integer assigned by the CA to each certificate. It is unique for each certificate issued by a given CA (i.e., the issuer name and serial number uniquely identify a certificate). See X.509(2000) 7, RFC3280 4.1.2.2

```
( 1.3.6.1.4.1.10126.1.5.3.2
      NAME 'x509serialNumber'
      DESC 'Unique integer for each certificate issued by a
            particular CA'
      EQUALITY integerMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 )
```

4.1.3 Signature algorithm

OID identifying the algorithm used by the CA in signing the certificate (see X.509(2000) 7, RFC3280 4.1.2.3) or the CRL.

```
   ( 1.3.6.1.4.1.10126.1.5.3.3
         NAME 'x509signatureAlgorithm'
         DESC 'OID of the algorithm used by the CA in
               signing the CRL or the certificate'
         EQUALITY objectIdentifierMatch
         SYNTAX 1.3.6.1.4.1.1466.115.121.1.38
         SINGLE-VALUE )
```

4.1.4 Issuer

String representation of the issuer's distinguished name (see X.509(2000) 7, RFC3280 4.1.2.4)

```
   ( 1.3.6.1.4.1.10126.1.5.3.4
         NAME 'x509issuer'
         DESC 'Distinguished name of the entity who has signed and
               issued the certificate'
         EQUALITY distinguishedNameMatch
         SYNTAX 1.3.6.1.4.1.1466.115.121.1.12
         SINGLE-VALUE )
```

Values of this attribute type must be encoded according to the syntax given in [RFC2253].

4.1.5 Validity

The "validity" attribute in an X.509 certificate (see X.509(2000) 7, RFC3280 4.1.2.5) consists of an ASN.1 sequence of two timestamps which define the begin and end of the certificate's validity period. This sequence has been split up into two separate attributes "x509validityNotBefore" and "x509validityNotAfter". The times are represented in string form as defined in [RFC2252].

```
   ( 1.3.6.1.4.1.10126.1.5.3.5
         NAME 'x509validityNotBefore'
         DESC 'Date on which the certificate validity period begins'
         EQUALITY generalizedTimeMatch
         ORDERING generalizedTimeOrderingMatch
         SYNTAX 1.3.6.1.4.1.1466.115.121.1.24
         SINGLE-VALUE )


   ( 1.3.6.1.4.1.10126.1.5.3.6
         NAME 'x509validityNotAfter'
         DESC 'Date on which the certificate validity period ends'
         EQUALITY generalizedTimeMatch
         ORDERING generalizedTimeOrderingMatch
         SYNTAX 1.3.6.1.4.1.1466.115.121.1.24
         SINGLE-VALUE )
```

Note that the field in the certificate may be in UTC or GeneralizedTime format. If in UTC format, the creator of this attribute MUST convert the UTC time into GeneralisedTime format when creating the attribute value.

4.1.6 Subject

String representation of the subject's distinguished name (see X.509(2000) 7, RFC3280 4.1.2.6).

```
( 1.3.6.1.4.1.10126.1.5.3.7
      NAME 'x509subject'
      DESC 'Distinguished name of the entity associated with this
         public-key'
      EQUALITY distinguishedNameMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.12
      SINGLE-VALUE )
```

Values of this attribute type must be encoded according to the syntax given in [RFC2253].

4.1.7 Subject public key info algorithm

OID identifying the algorithm associated with the certified public key (see X.509(2000) 7, RFC3280 4.1.2.7).

```
( 1.3.6.1.4.1.10126.1.5.3.8
      NAME 'x509subjectPublicKeyInfoAlgorithm'
      DESC 'OID identifying the algorithm associated with the certified
         public key'
      EQUALITY objectIdentifierMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.38
      SINGLE-VALUE )
```

4.2 Attributes for selected extensions

As this specification intends to only facilitate applications in finding certificates, only those extensions have to be defined that might be searched for. Thus extensions described in [RFC3280] like the following are not dealt with here:

- o private key usage period extension
- o policy mappings extension
- o subject directory attributes extension

- o basic constraints extension
- o name constraints extensions
- o policy constraints extensions
- o inhibit any policy extension
- o freshest CRL extension
- o authority information access extension
- o subject information access extension

4.2.1 Authority key identifier extension

This attribute identifies the public key to be used to verify the signature on this certificate or CRL (see X.509(2000) 8.2.2.1, RFC3280 4.2.1.1). The key may be identified by an explicit key identifier in the keyIdentifier component, by identification of a certificate for the key (giving certificate issuer in the authorityCertIssuer component and certificate serial number in the authorityCertSerialNumber component), or by both explicit key identifier and identification of a certificate for the key.

4.2.1.1 Authority key identifier

```
( 1.3.6.1.4.1.10126.1.5.3.11
      NAME 'x509authorityKeyIdentifier'
      DESC 'Key Identifier field of the Authority Key Identifier
         extension'
      EQUALITY octetStringMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.40
      SINGLE-VALUE )
```

4.2.1.2 Authority cert issuer

```
( 1.3.6.1.4.1.10126.1.5.3.12
      NAME 'x509authorityCertIssuer'
      DESC 'Authority Cert Issuer field of the Authority Key Identifier
         extension'
      EQUALITY distinguishedNameMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.12
      SINGLE-VALUE )
```

In this specification, only the "Name" choice, encoded according to [RFC2253], of the "GeneralName" type may be used.

4.2.1.3 Authority cert serial number

```
   ( 1.3.6.1.4.1.10126.1.5.3.13
         NAME 'x509authorityCertSerialNumber'
         DESC 'Authority Cert Serial Number field of the
            Authority Key Identifier extension'
         EQUALITY integerMatch
         SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
         SINGLE-VALUE )
```

4.2.2 Subject key identifier extension

This attribute identifies the public key being certified (see X.509(2000) 8.2.2.2, RFC3280 4.2.1.2). It enables distinct keys used by the same subject to be differentiated.

```
   ( 1.3.6.1.4.1.10126.1.5.3.14
         NAME 'x509subjectKeyIdentifier'
         DESC 'Key identifier which must be unique with respect to all
            key identifiers for the subject'
         EQUALITY octetStringMatch
         SYNTAX 1.3.6.1.4.1.1466.115.121.1.40
         SINGLE-VALUE )
```

4.2.3 Key usage extension

This attribute defines the purpose (e.g., encipherment, signature, certificate signing) of the key contained in the certificate (see X.509(2000) 8.2.2.3, RFC3280 4.2.1.3).

```
   ( 1.3.6.1.4.1.10126.1.5.3.15
         NAME 'x509keyUsage'
         DESC 'Purpose for which the certified public key is used'
         EQUALITY caseIgnoreIA5Match
         SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
```

Values of this type are encoded according to the following BNF, so that each value corresponds to the respective bit in the ASN.1 "KeyUsage" bitstring:

```
   x509keyUsage-value ="digitalSignature" / "nonRepudiation" /
      "keyEncipherment" / "dataEncipherment" / "keyAgreement" /
      "keyCertSign" / "cRLSign" / "encipherOnly" / "decipherOnly"
```

4.2.4 Policy information identifier extension

This attribute contains OIDs which indicate the policy under which the certificate has been issued and the purposes for which the certificate may be used (see X.509(2000) 8.2.2.6, RFC3280 4.2.1.5).

```
( 1.3.6.1.4.1.10126.1.5.3.16
      NAME 'x509policyInformationIdentifier'
      DESC 'OID which indicates the policy under which the
         certificate has been issued and the purposes for which
         the certificate may be used'
      EQUALITY objectIdentifierMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.38
      SINGLE-VALUE )
```

4.2.5 Subject alternative name extension

The subject alternative name extension allows additional identities to be bound to the subject of the certificate (see X.509(2000) 8.3.2.1, RFC3280 4.2.1.7). Separate attribute types are defined for all choices of the ASN.1 type "GeneralName" except for "otherName", "x400Address" and "ediPartyName".

4.2.5.1 Subject RFC822 name

```
( 1.3.6.1.4.1.10126.1.5.3.17
      NAME 'x509subjectRfc822Name'
      DESC 'Internet electronic mail address of the entity
         associated with this public-key'
      EQUALITY caseIgnoreIA5Match
      SUBSTR caseIgnoreIA5SubstringsMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute must be encoded according to the syntax given in [RFC0822].

4.2.5.2 Subject DNS name

```
( 1.3.6.1.4.1.10126.1.5.3.18
      NAME 'x509subjectDnsName'
      DESC 'Internet domain name of the entity
         associated with this public-key'
      EQUALITY caseIgnoreIA5Match
      SUBSTR caseIgnoreIA5SubstringsMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute must be encoded as Internet domain names in accordance with [RFC1035].

4.2.5.3 Subject directory name

```
( 1.3.6.1.4.1.10126.1.5.3.19
      NAME 'x509subjectDirectoryName'
      DESC 'Distinguished name of the entity
         associated with this public-key'
      EQUALITY distinguishedNameMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 )
```

Values of this attribute type must be encoded according to the syntax given in [RFC2253].

4.2.5.4 Subject Uniform Resource Identifier

```
( 1.3.6.1.4.1.10126.1.5.3.20
   NAME  'x509subjectURI'
      DESC 'Uniform Resource Identifier for the World-Wide Web
              of the entity associated with this public-key'
      EQUALITY caseExactIA5Match
      SUBSTR caseExactIA5SubstringsMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute must be encoded according to the syntax given in [RFC2396].

4.2.5.5 Subject IP address

```
( 1.3.6.1.4.1.10126.1.5.3.21
      NAME 'x509subjectIpAddress'
      DESC 'Internet Protocol address of the entity
         associated with this public-key'
      EQUALITY caseIgnoreIA5Match
      SUBSTR caseIgnoreIA5SubstringsMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute type must be stored in the syntax given in Appendix B of [RFC2373].

4.2.5.6 Subject registered ID

```
   ( 1.3.6.1.4.1.10126.1.5.3.22
         NAME 'x509subjectRegisteredID'
         DESC 'OID of any registered object identifying the entity
            associated with this public-key'
         EQUALITY objectIdentifierMatch
         SYNTAX 1.3.6.1.4.1.1466.115.121.1.38 )
```

registeredID is an identifier of any registered object assigned in accordance with ITU-T Rec. X.660.

4.2.6 Issuer alternative name extension

The issuer alternative names extension allows additional identities to be bound to the subject of the certificate or CRL (see X.509(2000) 8.3.2.2, RFC3280 4.2.1.8). Separate attribute types are defined for all choices of the ASN.1 type "GeneralName" except for "otherName", "x400Address" and "ediPartyName".

4.2.6.1 Issuer RFC 822 name

```
   ( 1.3.6.1.4.1.10126.1.5.3.23
         NAME 'x509issuerRfc822Name'
         DESC 'Internet electronic mail address of the entity who has
            signed and issued the certificate'
         EQUALITY caseIgnoreIA5Match
         SUBSTR caseIgnoreIA5SubstringsMatch
         SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute must be encoded according to the syntax given in [RFC0822].

4.2.6.2 Issuer DNS name

```
   ( 1.3.6.1.4.1.10126.1.5.3.24
         NAME 'x509issuerDnsName'
         DESC 'Internet domain name of the entity who has
            signed and issued the certificate'
         EQUALITY caseIgnoreIA5Match
         SUBSTR caseIgnoreIA5SubstringsMatch
         SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute must be encoded as Internet domain names in accordance with [RFC1035].

4.2.6.3 Issuer directory name

```
   ( 1.3.6.1.4.1.10126.1.5.3.25
         NAME 'x509issuerDirectoryName'
         DESC 'Distinguished name of the entity who has
            signed and issued the certificate'
         EQUALITY distinguishedNameMatch
         SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 )
```

Values of this attribute type must be encoded according to the syntax given in [RFC2253].

4.2.6.4 Issuer Uniform Resource Identifier

```
   ( 1.3.6.1.4.1.10126.1.5.3.26
         NAME 'x509issuerURI'
         DESC 'Uniform Resource Identifier for the World-Wide Web
                 of the entity who has signed and issued the certificate'
         EQUALITY caseExactIA5Match
         SUBSTR caseExactIA5SubstringsMatch
         SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute must be encoded according to the syntax given in [RFC2396].

4.2.6.5 Issuer IP address

```
   ( 1.3.6.1.4.1.10126.1.5.3.27
         NAME 'x509issuerIpAddress'
         DESC 'Internet Protocol address of the entity who has
            signed and issued the certificate'
         EQUALITY caseIgnoreIA5Match
         SUBSTR caseIgnoreIA5SubstringsMatch
         SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute type must be stored in the syntax given in Appendix B of [RFC2373].

4.2.6.6 Issuer registered ID

```
   ( 1.3.6.1.4.1.10126.1.5.3.28
         NAME 'x509issuerRegisteredID'
         DESC 'OID of any registered object identifying the entity who has
            signed and issued the certificate'
         EQUALITY objectIdentifierMatch
         SYNTAX 1.3.6.1.4.1.1466.115.121.1.38 )
```

registeredID is an identifier of any registered object assigned in accordance with ITU-T Rec. X.660.

4.2.7 Basic constraints extension

This attribute indicates whether the subject of the certificate is a CA (see X.509(2000) 8.4.2.1, RFC3280 4.2.1.10). If the value of this attribute is "TRUE", the certificate MUST be stored in the "cacertificate" attribute.

```
( 1.3.6.1.4.1.10126.1.5.3.29
      NAME 'x509basicConstraintsCa'
      DESC 'Identifies whether the subject of the certificate is a
         CA'
      EQUALITY booleanMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
      SINGLE-VALUE )
```

4.2.8 Extended key usage extension

This attribute indicates one or more purposes for which the certified public key may be used, in addition to or in place of the basic purposes indicated in the "x509keyUsage" attribute (see X.509(2000) 8.2.2.4, RFC3280 4.2.1.13). These purposes are identified by their OID.

```
( 1.3.6.1.4.1.10126.1.5.3.30
      NAME 'x509extKeyUsage'
      DESC 'Purposes for which the certified public key may be used,
         identified by an OID'
      EQUALITY objectIdentifierMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.38 )
```

4.2.9 CRL distribution points extension

This attribute identifies how the full CRL information for this certifacte can be obtained (see X.509(2000) 8.6.2.1, RFC3280 4.2.1.14).

```
( 1.3.6.1.4.1.10126.1.5.3.32
      NAME 'x509fullCRLDistributionPointURI'
      DESC 'URI type of DistributionPointName for the full CRL'
      EQUALITY caseExactIA5Match
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

In this specification, only the "uniformResourceIdentifier" choice of "distributionPoint.fullName" field is supported. If this attribute exists in an entry, both the "reasons" and "cRLIssuer" fields MUST be absent from the certificate, i.e. the CRL distributed by the distribution point contains revocations for all revocation reasons and the CRL issuer is identical to the certificate issuer.

Values of this attribute must be encoded according to the URI syntax given in [RFC2396].

4.3 Additional attributes

4.3.1 Certificate location

This attribute contains a pointer to the directory entry of a certificate. Thus it is possible to point to the certificate from an, e.g., white pages entry.

```
( 1.3.6.1.4.1.10126.1.5.4.74
      NAME 'x509certLocation'
      DESC 'Pointer to a x509certificate Entry'
      EQUALITY distinguishedNameMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 )
```

4.3.2 Certificate holder

This attribute contains a pointer to the directory entry of the end entity to which this certificate was issued. Thus it is possible to link a certificate entry in a certificate repository to, e.g., a white pages entry of the subject.

```
( 1.3.6.1.4.1.10126.1.5.4.75
      NAME 'x509certHolder'
      DESC 'Pointer to the directory entry of the end entity to which this
            certificate was issued'
      EQUALITY distinguishedNameMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 )
```

5. X.509 schema object classes

The object classes have been designed to form a logical set and be extensible in an orderly way as new PKC/CRL/AC extensions are defined. The methodology is as follows. Every X.509 entry (for a PKC, CRL or AC) is of the x509base abstract object class. There is then an additional abstract object class for each, derived from x509base, which holds the attributes extracted from the basic PKC/AC/CRL ASN.1 structure (excluding all extensions). The PKC object class is then instantiated by two structural object classes for user certificates and for CA certificates.

5.1 X.509 base object class

The x509base object class is the abstract object class that is the superior of all of the x.509 entry object classes

```
( 1.3.6.1.4.1.10126.1.5.4.2.1
      NAME 'x509base'
      ABSTRACT
      MAY x509version )
```

5.2 X.509 PKC object class

This abstract object class contains the fields of an X.509 user certificate or CA certificate that are used in searches as attributes and in name forms. It is derived from the abstract object class x.509base as specified in [ldap-crl-schema] and is base for the two following object classes.

```
( 1.3.6.1.4.1.10126.1.5.4.2.3
      NAME 'x509PKC'
      SUP x509base
      ABSTRACT
      MUST ( x509serialNumber $ x509validityNotBefore $
             x509validityNotAfter $ x509subjectPublicKeyInfoAlgorithm $
             x509signatureAlgorithm $ x509issuer )
      MAY  ( x509authorityKeyIdentifier $
        x509authorityCertIssuer $ x509authorityCertSerialNumber $
        x509subjectKeyIdentifier $ x509keyUsage $
        x509policyInformationIdentifier $
        x509subjectRfc822Name $ x509subjectDnsName $
        x509subjectDirectoryName $ x509subjectURI $
        x509subjectIpAddress $ x509subjectRegisteredID $
        x509issuerRfc822Name $ x509issuerDnsName $
        x509issuerDirectoryName $ x509issuerURI $
        x509issuerIpAddress $ x509issuerRegisteredID $
        x509extKeyUsage $ x509FullcRLDistributionPointURI $
             x509certHolder $
             x509issuerSerial $ x509basicConstraintsCa ) )
```

The attribute description of x509issuerSerial can be found in [ldap-ac-schema]

5.3 X.509 user certificate object class

This object class is for storing user certificates.

```
( 1.3.6.1.4.1.10126.1.5.4.2.4
      NAME 'x509userCertificate'
      SUP x509PKC
      STRUCTURAL
      MUST userCertificate
      MAY  x509subject )
```

The attribute description of userCertificate can be found in [pkix-ldap-schema]. Although this attribute type is specified as multi-valued it MUST NOT contain more than one certificate if used with this object class.

The attribute type x509subject is specified here as a MAY attribute. Nevertheless if this attribute is not used at least one of the following attributes MUST be filled in: x509subjectRfc822Name, x509subjectDnsName, x509subjectDirectoryName, x509subjectURI, x509subjectIpAddress, or x509subjectRegisteredID.

5.4 X.509 CA certificate object class

This object class is for storing CA certificates.

```
( 1.3.6.1.4.1.10126.1.5.4.2.5
      NAME 'x509caCertificate'
      SUP x509PKC
      STRUCTURAL
      MUST ( caCertificate $ x509subject ) )
```

The attribute description of caCertificate can be found in [pkix-ldap-schema]. Although this attribute type is specified as multi-valued it MUST NOT contain more than one certificate if used with this object class.

5.5 X.509 certificate holder object class

This auxiliary object class has an attribute that contains a pointer to an entry with x509certicate objectclass. Thus it is possible to link, e.g., an entry of a white pages directory to an entry in a certificate store.

```
( 1.3.6.1.4.1.10126.1.5.4.2.2
      NAME 'x509certificateHolder'
      AUXILIARY
      MAY  ( x509certLocation ) )
```

6. DIT structure and naming

If the schema presented in this document is used to store certificate information in a directory that contains entries for organizations, persons, services, etc., each certificate belonging to an entity SHOULD be stored as a direct subordinate to the entity's entry. In this case, these entries MUST be named by a multi-valued RDN formed by the certificate issuer and serial number, as this is the only way to enforce unique RDN under the siblings. This is expressed in the following two name forms:

```
( 1.3.6.1.4.1.10126.1.5.5.6
      NAME 'x509userCertificateNameform'
      OC x509userCeriticate
      MUST ( x509serialNumber $ x509issuer ) )


( 1.3.6.1.4.1.10126.1.5.5.7
      NAME 'x509caCertificateNameform'
      OC x509caCertificate
      MUST ( x509serialNumber $ x509issuer ) )
```

There are some LDAP implementations that don't support multi-valued RDNs. These can use following alternative two name forms:

```
( 1.3.6.1.4.1.10126.1.5.5.8
      NAME 'x509PKCAltNameForm'
      OC x509PKC
      MUST x509issuerSerial )
```

For public directories of CAs that, besides the data stored in the certificates, do not hold any additional data about end entities the following DIT structure might be preferable. Entries for certificates are stored directly below the issuing CA's entry. In this case these entries SHOULD be named by the certificate serial number. This is expressed in the following two name forms:

```
( 1.3.6.1.4.1.10126.1.5.5.10
      NAME 'x509PKCSerialNumberNameForm'
      OC x509PKC
      MUST x509serialNumber )
```

Care must be taken when encoding DNs that contain an x509issuer attribute. Such a value is a string representation according to [RFC2253]. These strings contain RFC2253 special characters and must therefore be escaped. For example, the issuer name in a certificate may be:

```
   x509issuer: OU=VeriSign Trust Network,OU=(c) 1998 VeriSign\2c Inc. -
     For authorized use only,OU=Class 1 Public Primary Certification Au
     thority - G2,O=VeriSign\2c Inc.,C=US
```

When used in a DN, this will be appear as:

```
   dn: x509serialNumber=123456+x509issuer=OU\3dVeriSign Trust Network
    \2cOU\3d(c) 1998 VeriSign\5c\2c Inc. - For authorized use only\2cOU\3d
    Class 1 Public Primary Certification Authority - G2\2cO\3dVeriSig
    n\5c\2c Inc.\2cC\3dUS,cn=Joe Example,...
```

7. Security Considerations

Attributes of directory entries are used to provide descriptive information about the real-world objects they represent which can be people, organizations, or devices. Most countries have privacy laws regarding the publication of information about people.

Without additional mechanisms such as Operation Signatures [RFC2649] which allow a client to verify the origin and integrity of the data contained in the attributes defined in this document, a client MUST NOT treat this data as authentic. Clients MUST only use - after proper validation - the data which they obtained directly from the certificate. Directory administrators MAY deploy ACLs which limit access to the attributes defined in this document to search filters.

Transfer of cleartext passwords is strongly discouraged where the underlying transport service cannot guarantee confidentiality and may result in disclosure of the password to unauthorized parties.

In order to protect the directory and its contents, strong authentication MUST have been used to identify the Client when an update operation is requested.

8. IANA Considerations

This document uses the OID 1.3.6.1.4.1.10126.1.5 to identify the LDAP schema elements described here. This OID was assigned by DAASI International, under its IANA-assigned private enterprise allocation [PRIVATE], for use in this specification.

9. Acknowledgments

This document borrows from a number of IETF documents, including [certinfo-schema].

The authors wish to thank David Chadwick, Russ Housley, Mikhail

Sahalayev, Michael Stroeder, and Kurt Zeilenga for their contributions to this document.

This work is part of the DFN Project "Ausbau und Weiterbetrieb eines Directory Kompetenzzentrums" funded by the German Ministry of Research (BMBF).

The last versions of this work were made in the frame of the project "PKI/LDAP" funded by the Ministry of Science, Research and The Arts of Baden-Wuerttemberg, Germany.

This document has been written in XML according to the DTD specified in RFC2629. xml2rfc has been used to generate an RFC2033 compliant plain text form. The XML source and a HTML version are available on request.

10. References

10.1 Normative references

[PRIVATE] IANA, "Private Enterprise Numbers", <http://www.iana.org/assignements/enterprise-numbers>.

[RFC0822] Crocker, D., "Standard for the format of ARPA Internet text messages", STD 11, RFC 822, August 1982.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[RFC2252] Wahl, M., Coulbeck, A., Howes, T. and S. Kille, "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions", RFC 2252, December 1997.

[RFC2253] Wahl, M., Kille, S. and T. Howes, "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names", RFC 2253, December 1997.

[RFC2256] Wahl, M., "A Summary of the X.500(96) User Schema for use with LDAPv3", RFC 2256, December 1997.

[RFC2373] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 2373, July 1998.

[RFC2396] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[RFC2798] Smith, M., "Definition of the inetOrgPerson LDAP Object Class", RFC 2798, April 2000.

[RFC3280] Housley, R., Polk, T., Ford, W. and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and CRL Profile", RFC 3280, April 2002.

[RFC3377] Hodges, J. and RL. Morgan, "Lightweight Directory Access Protocol (v3): Technical Specification", RFC 3377, September 2002.

[ldap-ac-schema]
Chadwick, D. and M. Sahalayev, "Internet X.509 Public Key Infrastructure - LDAP Schema for X.509 Attribute Certificates", Internet Draft (work in progress), May 2004, <draft-ietf-pkix-ldap-ac-schema-02.txt>.

[ldap-crl-schema]
Chadwick, D. and M. Sahalayev, "Internet X.509 Public Key Infrastructure - LDAP Schema for X.509 CRLs", Internet Draft (work in progress), May 2004, <draft-ietf-pkix-ldap-crl-schema-02.txt>.

[pkix-ldap-schema]
Chadwick, D. and S. Legg, "Internet X.509 Public Key Infrastructure - LDAP Schema and Syntaxes for PKIs", Internet Draft (work in progress), June 2002, <draft-ietf-pkix-ldap-pki-schema-00.txt>.

10.2 Non-normative references

[RFC2312] Dusse, S., Hoffman, P., Ramsdell, B. and J. Weinstein, "S/MIME Version 2 Certificate Handling", RFC 2312, March 1998.

[RFC2649] Greenblatt, B. and P. Richard, "An LDAP Control and Schema for Holding Operation Signatures", RFC 2649, August 1999.

[RFC2651] Allen, J. and M. Mealling, "The Architecture of the Common Indexing Protocol (CIP)", RFC 2651, August 1999.

[RFC2654] Hedberg, R., Greenblatt, B., Moats, R. and M. Wahl, "A Tagged Index Object for use in the Common Indexing Protocol", RFC 2654, August 1999.

```
   [RFC3039]  Santesson, S., Polk, T., Barzin, P. and M. Nystrom,
              "Internet X.509 Public Key Infrastructure Qualified
              Certificate Profile", RFC 3039, January 2001.

   [RFC3687]  Legg, S., "Lightweight Directory Access Protocol and X.500
              Component Matching Rules", RFC 3687, February 2004.

   [X.509-2000]
              ITU, "Information  Technology - Open Systems
              Interconnection - The  Directory: Public-key and attribute
              certificate frameworks", ITU-T  Recommendation   X.509,
              March 2000.

   [certinfo-schema]
              Greenblatt, B., "LDAP Object Class for Holding Certificate
              Information", Internet Draft (expired), Februar 2000,
              <http://www.watersprings.org/pub/id/draft-greenblatt-ldap-
certinfo-schema-02.txt>
              .

   [matchedval]
              Chadwick, D. and S. Mullan, "Returning Matched Values with
              LDAPv3", Internet Draft (work in progress), September
              2002, <draft-ietf-ldapext-matchedval-07.txt>.
```

Authors' Addresses

```
   Peter Gietz
   DAASI International GmbH
   Wilhelmstr. 106
   Tuebingen  72074
   DE

   Phone: +49 7071 29 70336
   EMail: peter.gietz@daasi.de
   URI:   http://www.daasi.de/


   Norbert Klasen
   Avinci
   Halskestr. 38
   Ratingen  40880
   DE

   EMail: norbert.klasen@avinci.de
```

Appendix A. Sample directory entries

A sample x509certificate directory entry for an intermediate CA certificate in LDIF format:

```
dn: x509serialNumber=4903272,EMAILADDRESS=certify@pca.dfn.de,CN=DFN T
 oplevel Certification Authority,OU=DFN-PCA,OU=DFN-CERT GmbH,O=Deutsc
 hes Forschungsnetz,C=DE
objectclass: x509caCertificate
x509version: 2
x509serialNumber: 4903272
x509issuer: EMAILADDRESS=certify@pca.dfn.de,CN=DFN Toplevel Certifica
 tion Authority,OU=DFN-PCA,OU=DFN-CERT GmbH,O=Deutsches Forschungsnet
 z,C=DE
x509validityNotBefore: 20020110170112Z
x509validityNotAfter: 20060110170112Z
x509subject: EMAILADDRESS=ca@daasi.de,OU=DAASI CA,O=DAASI Internation
 al GmbH,C=DE
x509subjectPublicKeyInfoAlgorithm: 1.2.840.113549.1.1.1
x509basicConstraintsCa: TRUE
x509keyUsage: keyCertSign
x509keyUsage: cRLSign
x509subjectKeyIdentifier:: 5nrZFpVK4RKfIglqQ4N4JXBS4Bk=
x509cLRdistributionPointURI: http://www.dfn-pca.de/certification/x509
 /g1/data/crls/root-ca-crl.crx
x509cLRdistributionPointURI: http://www.dfn-pca.de/certification/x509
 /g1/data/crls/root-ca-crl.crl
x509policyInformationIdentifier: 1.3.6.1.4.1.11418.300.1.1
x509caCert:: MIIHTTCCBjWgAwIBAgIDStFoMA0GCSqGSIb3DQEBBQUAMI
 GsMQswCQYDVQQGEwJERTEhMB8GA1UEChMYRGV1dHNjaGVzIEZvcnNjaHVuZ3NuZXR6MR
 YwFAYD VQQLEw1ERk4tQ0VSVCBHbWJIMRAwDgYDVQQLEwdERk4tUENBMS0wKwYDVQQDE
 yRERk4gVG9 wbGV2ZWwgQ2VydGlmaWNhdGlvbiBBdXRob3JpdHkxITAfBgkqhkiG9w0B
 CQEWEmNlcnRpZnlAcGNhLmRmbi5kZTAeFw0wMjAxMTAxNzAxMTJaFw0wNjAxMTAxNzAx
 MTJaMF8xCzAJBgNVBAYTAkRFMSEwHwYDVQQKExhEQUFTSSBJbnRlcm5hdGlvbmFsIEdt
 YkgxETAPBgNVBAsTCERB QVNJIENBMRowGAYJKoZIhvcNAQkBFgtjYUBkYWFzaS5kZTC
 CASIwDQYJKoZIhvcNAQEBBQA DggEPADCCAQoCggEBAKmQBp+Gr28/qlEWjnJoiH3Awm
 hNEYMRWgXMXMMjM3S4mSmXZ8FZfTSPhi5O1zx5nyHecfl01fAO79Kpc6XkOTOl4iKBwu
 7+DM6my9Iizp2puhOQ6iuuchAIyJQPR0lfWAvvW+4n7Nf13Js5qFHvXBDqvgt6fud1l8
 XZ4nPWBSbs6OnB4EUDlRLx5fdCX2sEPQINKeu0INMtjHI6eGbspmahup0ArPA9RYZVjV
 q6ZHkh4205/JAhji9KtFifKCztXNTRMba7AHd2uS6GbF9+chGLPWGNZKtMhad1SvU7Zl
 w/ySHkFbBFZMu6x3kAVgwW8gKQa5qSFnMw/WTKATJRPekCAwEAAaOCA8IwggO+MA8GA1
 UdEwEB/wQFMAMBAf8wCwYDVR0PBAQDAgEGMB0GA1UdDgQWBBTmetkWlUrhEp8iCWpDg3
 glcFLgGTCB2wYDVR0jBIHTMIHQgBQGC/q1+Eh4oyCxCz7PoNDE0X990KGBsqSBrzCBrD
 ELMAkGA1UEBhMCREUxITAfBgNVBAoTGERldXRzY2hlcyBGb3JzY2h1bmdzbmV0ejEWMB
 QGA1UECxMNREZOLUNFUlQgR21iSDEQMA4GA1UECxMHREZOLVBDQTEtMCsGA1UEAxMkRE
 ZOIFRvcGxldmVsIENlcnRpZmljYXRpb24gQXV0aG9yaXR5MSEwHwYJKoZIhvcNAQkBFh
 JjZXJ0aWZ5QHBjYS5kZm4uZGWCAxXP/TCBpQYDVR0fBIGdMIGaMEugSaBHhkVodHRwOi
 8vd3d3LmRmbi1wY2EuZGUvY2VydGlmaWNhdGlvbi94NTA5L2cxL2RhdGEvY3Jscy9yb2
```

```
    90LWNhLWNybC5jcngwS6BJoEeGRWh0dHA6Ly93d3cuZGZuLXBjYS5kZS9jZXJ0aWZpY2
    F0aW9uL3g1MDkvZzEvZGF0YS9jcmxzL3Jvb3QtY2EtY3JsLmNybDARBglghkgBhvhCAQ
    EEBAMCAQYwSwYJYIZIAYb4QgEIBD4WPGh0dHA6Ly93d3cuZGZuLXBjYS5kZS9jZXJ0aW
    ZpY2F0aW9uL3BvbGljaWVzL3g1MDlwb2xpY3kuaHRtbDCB+QYJYIZIAYb4QgENBIHrFo
    HoVGhpcyBjZXJ0aWZpY2F0ZSB3YXMgaXNzdWVkIGJ5IHRoZSBERk4tUENBLCB0aGUgVG
    9wCkxldmVsIENlcnRpZmljYXRpb24gQXV0aG9yaXR5IG9mIHRoZSBHZXJtYW4gUmVzZW
    FyY2gKTmV0d29yayAoRGV1dHNjaGVzIEZvcnNjaHVuZ3NuZXR6LCBERk4pLgpUaGUga2
    V5IG93bmVyJ3MgaWRlbnRpdHkgd2FzIGF1dGhlbnRpY2F0ZWQgaW4KYWNjb3JkYW5jZS
    B3aXRoIHRoZSBERk4tUENBIHg1MDkgUG9saWN5LjA3BglghkgBhvhCAQMEKhYoaHR0cH
    M6Ly93d3cuZGZuLXBjYS5kZS9jZ2kvY2hlY2stcmV2LmNnaTBkBgNVHSAEXTBbMFkGCy
    sGAQQB2RqCLAEBMEowSAYIKwYBBQUHAgEWPGh0dHA6Ly93d3cuZGZuLXBjYS5kZS9jZX
    J0aWZpY2F0aW9uL3BvbGljaWVzL3g1MDlwb2xpY3kuaHRtbDANBgkqhkiG9w0BAQUFAA
    OCAQEAU9GmwCW6LwsyHfC241afldqj/GULv8mfSkUEpK2OtYU1JAYFzmQx69iweOKHbg
    XZKZA2Wox+9AydIe98MJCSCOFKYjkzgXU4fEZbEgnZBo+/1+W2BoB6gFAWy77KVHgimA
    7AqCcfbObeyCmyfLg1ro8/KpE01OjNr0S+EfZ3gX9sezjVkCy12HBNQknz/hT2af25UU
    hyFTcvUY4xvlKAQpla29qyO28sfO93Qhkum6SU2XPlsKU+3lyqF33Xy84Y2z8ScVlsMu
    VWbUGtmVshnpT5K91n42pu/f0rLtkZDssEDbcLnQDLWEz1aUDkLC++4CeFJxC/Dd/SOr
    E0yR0hNQ=
```

A sample x509certificate directory entry for an end identity certificate in LDIF format:

```
   dn: x509serialNumber=158163180827231005435325711272171 3,EMAILADDRESS=
    certificate@trustcenter.de,OU=TC TrustCenter Class 1 CA,O=TC TrustCe
    nter for Security in Data Networks GmbH,L=Hamburg,ST=Hamburg,C=DE
   objectclass: x509userCertificate
   x509version: 2
   x509serialNumber: 1581631808272310054353257112721713
   x509issuer: EMAILADDRESS=certificate@trustcenter.de,OU=TC TrustCente
    r Class 1 CA,O=TC TrustCenter for Security in Data Networks GmbH,L=
    Hamburg, ST=Hamburg,C=DE
   x509validityNotBefore: 20011030180757Z
   x509validityNotAfter: 20021030180757Z
   x509subject: EMAILADDRESS=norbert.klasen@daasi.de,CN=Norbert Klasen,C
    =DE
   x509subjectPublicKeyInfoAlgorithm: 1.2.840.113549.1.1.1
   x509userCert:: MIIDOTCCAqKgAwIBAgIOTfsAAAACxOstmlOu2TEwDQYJ
    KoZIhvcNAQEEBQAwgbwxCzAJBgNVBAYTAkRFMRAwDgYDVQQIEwdIYW1idXJnMRAwDgYD
    VQQHEwdIYW1idXJnMTowOAYDVQQKEzFUQyBUcnVzdENlbnRlciBmb3IgU2VjdXJpdHkg
    aW4gRGF0YSBOZXR3b3JrcyBHbWJIMSIwIAYDVQQLExlUQyBUcnVzdENlbnRlciBDbGFz
    cyAxIENBMSkwJwYJKoZIhvcNAQkBFhpjZXJ0aWZpY2F0ZUB0cnVzdGNlbnRlci5kZTAe
    Fw0wMTEwMzAxODA3NTdaFw0wMjEwMzAxODA3NTdaME4xCzAJBgNVBAYTAkRFMRcwFQYD
    VQQDEw5Ob3JiZXJ0IEtsYXNlbjEmMCQGCSqGSIb3DQEJARYXbm9yYmVydC5rbGFzZW5A
    ZGFhc2kuZGUwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAL8+XK98p4YjD7Wq7Apm
    hAN/j2tfVsFCS0ufy12vGpEtG4ny1tbbBORCJI8vIlDr2/vVTESl6UjzceloVUCib5V8
    55mKUVmLL9Ay4qQLFd4wAoRSPAu9DkfbR+ygjzaYq+MUKMwaB61sG6911xk/e2/IIq8/
    IHKrRoYQGmHkaaJpAgMBAAGjgaowgacwMwYJYIZIAYb4QgEIBCYWJGh0dHA6Ly93d3cu
    dHJ1c3RjZW50ZXIuZGUvZ3VpZGVsaW5lczARBglghkgBhvhCAQEEBAMCBaAwXQYJYIZI
    AYb4QgEDBFAWTmh0dHBzOi8vd3d3LnRydXN0Y2VudGVyLmRlL2NnaS1iaW4vY2hlY2st
    cmV2LmNnaS80REZCMDAwMDAwMDJDNEVCMkQ5QTUzQUVEOTMxPzANBgkqhkiG9w0BAQQF
    AAOBgQCrAzuZzLztupeqcHa8OUOcnRuTacMpBEeICbZMKv6mN9rMYkAxFKerj/yXbdCE
    8/3X3L00eGj+a8A7PumATiJSfmvYqa4EMZwHC2FFqPxYyAj+xVuSlL5AC4HGHu4SOCp/
    UJu1xysoD16chOOLpj7+ZWZWLHIjA3zeXwUGl4kFvw==
```

Appendix B. Sample searches

This section details how clients should access the certstore. The searches are presented in LDAP URL format.

Retrieve all certificates for an end entity from a certstore using the first DIT structure:

```
   ldap:///CN=Norbert%20Klasen,O=DAASI%20International%20GmbH,C=de?
      x509userCert?one?(objectClass=x509userCertificate)
```

Find a certificate in a trustcenter's certstore suitable for sending an encrypted S/MIME message to "norbert.klasen@daasi.de"

```
   ldap:///O=TC%20TrustCenter%20for%20Security%20in%20Data%20Networks
      %20GmbH,L=Hamburg,ST=Hamburg,C=de?x509userCert?sub?
      ((&(objectClass=x509userCertificate)
        (x509subjectRfc822Name=norbert.klasen@daasi.de) )
       (|(x509keyUsage=keyEncipherment)(x509keyUsage=keyAgreement)
         (x509extendedKeyUsage=1.3.6.1.5.5.7.3.4)))
```

Find a CA certificate by its "subjectKeyIdentifier" obtained from the "keyIdentifier" field of the "autorityKeyIdentifier" extension in an end entity certificate:

```
   ldap:///?caCertificate?sub?
      (&(objectClass=x509caCertificate)(x509subjectKeyIdentifier=%5CE6
       %5C7A%5CD9%5C16%5C95%5C4A%5CE1%5C12%5C9F%5C22%5C09%5C6A%5C43%
       5C83%5C78%5C25%5C70%5C52%5CE0%5C19))
```

Appendix C. Changes from previous Drafts

C.1 Changes in draft-klasen-ldap-x509certificate-schema-01

- o Included new Attributes x509authorityKeyIdentifier, x509authorityCertissuer, x509authorityCertSerialNumber, x509certificateLocation, x509certificateHolder, and new objectclass x509certificateHolder
- o Fixed bug in definition of objectclass x509certificate
- o Changed references from RFC 2459 to RFC 3280 and included some respective language in 3.2.
- o Changed references from RFC 2251 to RFC 3377 and deleted all references to LDAPv2.
- o Deleted ";binary" in examples
- o Included new section: Comparision with component matching approach
- o Some changes in wording and section titles, and elimination of typos
- o Changed order of authors, and one author's address

C.2 Changes in draft-klasen-ldap-x509certificate-schema-02

- o abstract object class x509PKC
- o aligned to [ldap-ac-schema] and [ldap-crl-schema]

C.3 Changes in draft-klasen-ldap-x509certificate-schema-03

- o Changed Matching Rules from caseIgnoreMatch to caseIgnoreIA5Match etc.
- o moved the references to RFC 3280 from the DESC part of the attribute definition to the text
- o added some additional text about CIP in Introduction
- o reworded text for x509subjectPublicKeyInfoAlgorithm
- o changed x509userCert and x509caCert to be inherited from userCertificate and caCertificate respectively

- o added clarification about x509subject and subject alternative names
- o added attribute type x509issuerSerial to x509PKC object class
- o added attribute type x509basicConstraintsCa to x509PKC object class
- o renamed attributetype x509cRLDistributionPointURI to x509FullcRLDistributionPointURI
- o devided references in normative and non normative
- o deleted attributetype mail from x509PKC objectclass
- o created separate Name Forms for x509userCertificate and x509caCertificate object classes.
- o changed attributetype x509SerialNumber to MULTI-VALUE
- o adjusted examples to new schema
- o Fixed more typos

C.4 Changes in draft-ietf-pkix-ldap-pkc-schema-00

- o renamed the title and file name
- o deleted attribute types x509userCert and x509caCert and replaced them with the standard userCertificate and caCertificate attribute types.
- o included the description of x509base object class assigning a new OID to it.
- o renamed x509issuerUniforResourceIdentifier and x509subjectUniforResourceIdentifier to x509issuerURI and x509subjectURI respectivly.
- o replaced separate Name Forms for x509userCertificate and x509caCertificate object classes by a single x509PKCNameForm.
- o included a super section x509 Schema Object Classes with introductory remarks (taken from [ldap-ac-schema])
- o added some additional wording and some ASCII art in the introduction
- o updated references
- o added an IANA considerations section
- o added another acknowledgement

Intellectual Property Statement

Full Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

Appendix 6. LDAP Schema for X.509 Attribute Certificates

INTERNET-DRAFT D. W. Chadwick
PKIX WG M. V. Sahalayev
Intended Category: Informational University of Salford
Expires on 9 January 2005 10 July 2004

Internet X.509 Public Key Infrastructure
LDAP Schema for X.509 Attribute Certificates
<draft-ietf-pkix-ldap-ac-schema-01.txt>

ABSTRACT

This document describes an LDAP schema for X.509 attribute certificates (ACs). Each AC is broken down into a set of attribute types. These attributes can then be stored in an AC entry. An object class is defined for this AC entry. Each attribute type uses an existing LDAP syntax, so that no new matching rules need to be defined.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [2].

TABLE OF CONTENTS

1. Introduction

It currently isn't possible to search LDAP servers for X.509 [6] attributes (public key certificates, CRLs etc.) as no matching rules have been defined for them. A couple of Internet Drafts [9,10] have been specified, but implementation of them is complex. Component matching [19] defines a mechanism for matching against complex syntaxes, by defining generic matching rules that can match against any user selected component parts in an attribute value of any arbitrarily complex attribute syntax. This might prove to be the proper way to solve LDAP search problems in the longer term, but it will take a long time until such ASN.1 based mechanisms are implemented in all LDAP servers and clients. Even when this has happened the mechanism proposed in this document will still be useful to some applications such as CIP [20].

A simple and easy to implement mechanism is needed today to search for X.509 attributes. Rather than search for an X.509 attribute in an entry, it suggests the directory administrative user creates an entry (in the case of pubic key and attribute certificates) or a subtree (in the case of CRLs) from the X.509 attribute. The attributes of these new entries will be created from fields of the X.509 attribute (e.g. the issuer field), and if these new attributes are defined using existing LDAP syntaxes and matching rules, then it will be possible to use existing LDAP server technology to search for fields in X.509 attributes.

This document is one of a set comprising:

i) the LDAP schema for X.509 public key certificates [7]
ii) the LDAP schema for X.509 attribute certificates (this document)
iii) the LDAP schema for X.509 CRLs [8]

Schema definitions are provided using LDAPv3 description formats from RFC2252 [3]. Definitions provided here are formatted (line wrapped) for readability. The specifications use the augmented Backus-Naur Form (ABNF) as described in RFC2234 [4].

2. DIT Structure and Naming

If the schema presented in this document is used to store information about ACs in an LDAP directory, each AC SHOULD be stored as a direct subordinate of the AC holder's entry. These entries SHOULD be named using either the x509ACNameForm i.e. by a multi-valued RDN formed by the AC issuer and serial number, or by the x509ACAltNameForm i.e. by a single valued RDN formed by concatenating the AC issuer and serial number, as these are the only ways to enforce unique RDNs under the holder's entry. Exceptionally, if it can be guaranteed that only ACs from a single issuer will be stored under the holder's entry, the x509ACserialNumberNameForm MAY be used, i.e. the single valued RDN formed from the AC serial number.

```
  (1.2.826.0.1.3344810.1.3.3
      NAME 'x509ACNameForm'
      OC x509AC
      MUST ( x509serialNumber $ x509issuer ) )

  (1.2.826.0.1.3344810.1.3.4
      NAME 'x509ACAltNameForm'
      OC x509AC
      MUST ( x509issuerSerial ) )

 (1.2.826.0.1.3344810.1.3.5
      NAME 'x509ACserialNumberNameForm'
      OC x509AC
      MUST ( x509serialNumber ) )
```

The following attribute description describes the attribute used to hold the alternative RDN name form.

```
  (1.2.826.0.1.3344810.1.1.60
        NAME 'x509issuerSerial'
        DESC 'Used to hold the RDN of a certificate entry, formed by
              concatenating the AC serial number and issuer fields '
        EQUALITY distinguishedNameMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.12
        SINGLE-VALUE )
```

When encoding DNs that contain an x509issuer field, the string representation must be made according to [13]. These strings contain RFC2253 special characters and must therefore be escaped. For example, the issuer name in a certificate may be:

```
   x509issuer: OU=VeriSign Trust Network,OU=(c) 1998 VeriSign Inc. -
     For authorized use only,OU=Class 1 Public Primary Certification Au
     thority - G2,O=VeriSign Inc.,C=US
```

When used in the x509issuerSerial attribute of a DN, this may appear as:

```
   dn: x509issuerSerial=123456\,OU\=VeriSign Trust Network \,OU
    \=(c) 1998 VeriSign Inc. - For authorized use only\,OU\=Class 1 Public
    Primary Certification Authority - G2\,O\=VeriSign Inc.\2cC\3dUS
```

3. X.509 Schema Object Classes

The object classes have been designed to form a logical set and be extensible in an orderly way as new PKC/CRL/AC extensions are defined. The methodology is as follows. Every X.509 entry (for a PKC, CRL or AC) is of the x509base abstract object class. There is then an additional abstract object class for each, derived from x509base, which holds the attributes extracted from the basic PKC/AC/CRL ASN.1 structure (excluding all extensions). Further, there is an auxiliary object class for the extensions defined in X.509 [6], and an additional auxiliary object class (if needed) for extensions defined in existing Internet RFCs. As new extensions are defined, then new auxiliary object classes and attributes will need to be defined to cater for the attributes to be extracted from these. Finally there are several structural object classes for each, which allow the X.509 DER encoded attribute to be stored in the entry.

The X.509 base object class is defined in [8].

3.1 X509 AC object class

The X.509AC abstract object class is used to hold the attributes extracted from the basic fields of an attribute certificate. The X.509 AC object class is the abstract object class from which the structural object classes for attributeCertificate entries and AA certificate entries are derived.

```
(1.2.826.0.1.3344810.1.0.16
      NAME 'x509AC'
      SUP x509base
      ABSTRACT
```

```
    MUST ( x509version $
           x509serialNumber $
           x509signatureAlgorithm $
           x509issuer $
           x509validityNotBefore $
           x509validityNotAfter )

    MAY  ( x509ACHolderPKCSerialNumber $
           x509ACHolderPKCissuerDN $
           x509ACHolderRfc822Name $
           x509ACHolderDNSName $
           x509ACHolderDN $
           x509ACHolderURI $
           x509ACHolderIPAddress $
           x509ACHolderRegisteredID $
           x509authorityCertIssuer $
           x509authorityCertSerialNumber $
           x509authorityKeyIdentifier $
           x509ACObjectDigest $
           x509ACDigestAlgorithm $
           x509ACDigestedObjectType $
           x509issuerSerial ) )
```

The definition of x509base can be found in [7].

3.2 X.509 attribute certificate object class

This structural object class is for entries of attribute certificates belonging to holders.

```
  (1.2.826.0.1.3344810.1.0.17
   NAME 'x509attributeCertificate'
   SUP x509AC
   MUST attributeCertificateAttribute )
```

The attributeCertificateAttribute is defined in [10].

3.3 X.509 AA certificate object class

This structural object class is for entries of attribute certificates belonging to Attribute Authorities.

```
  (1.2.826.0.1.3344810.1.0.18
   NAME 'x509aACertificate'
   SUP x509AC
   MUST aACertificate )
```

The aACertificate attribute is defined in [10].

3.4 Embedded attributes

The x509AC object class does not contain the attributes embedded in the attribute certificate, since these can be attributes of any type. Therefore the LDAP entry created to hold the AC should also be of the auxiliary object classes appropriate for the attributes embedded in the AC. One pragmatic solution to this is to make the entry of object class extensibleObject [3].

3.5 X.509 AC extensions auxiliary object class

The x509ACext auxiliary object class is used to hold the attributes extracted from the AC extensions defined in the X.509 standard [6] and profiled in [5].

Note. If an AC holds additional extensions to these, then another auxiliary object class and supporting attributes will need to be defined.

```
(1.2.826.0.1.3344810.1.0.22
    NAME 'x509ACext'
    AUXILIARY
    MAY  ( x509issuerRfc822Name $
         x509issuerDNSName $
         x509issuerURI $
         x509issuerIPAddress $
         x509issuerRegisteredID $
         x509authorityCertIssuer $
         x509authorityCertSerialNumber $
         x509authorityKeyIdentifier $
         x509ACAuditID $
         x509ACTargetRfc822Name    $
         x509ACTargetDNSName    $
         x509ACTargetDN    $
         x509ACTargetURI    $
         x509ACTargetIPAddress    $
         x509ACTargetRegisteredID    $
         x509ACTargetGroupRfc822Name    $
         x509ACTargetGroupDNSName    $
         x509ACTargetGroupDN    $
         x509ACTargetGroupURI    $
         x509ACTargetGroupIPAddress    $
         x509ACTargetGroupRegisteredID    $
         x509DPRfc822Name  $
         x509DPDNSName  $
         x509DPDN  $
         x509DPURI  $
         x509DPIPAddress  $
         x509DPRegisteredID  $
         x509DPrelativeToIssuer $
         x509DPissuerRfc822Name  $
         x509DPissuerDNSName  $
         x509DPissuerDN  $
         x509DPissuerURI  $
         x509DPissuerIPAddress  $
         x509DPissuerRegisteredID  $
```

```
        x509DPReasonCodes  $
        x509ACNoRevocation ) )
```

4. Common X.509 Attribute Types

The following attribute types defined in [7] are used to hold the corresponding fields of ACs:

- x509serialNumber - used to hold the serial number of the AC
- x509version - used to hold the version of the AC
- x509signatureAlgorithm - used to hold the OID of the algorithm used to sign the CRL
- x509issuer - used to hold the DN of the AC issuer
- x509validityNotBefore - used to hold the not before validity time of the AC (note that only the Generalized Time format is permitted)
- x509validityNotAfter - used to hold the not after validity time of the AC (note that only the Generalized Time format is permitted)
- x509authorityCertIssuer - used in conjunction with x509authorityCertSerialNumber to identify the public key certificate of the AC issuer
- x509authorityCertSerialNumber - used in conjunction with x509authorityCertIssuer to identify the public key certificate of the AC issuer
- x509issuerRfc822Name - used to hold the email address of the AC issuer
- x509issuerDNSName - used to hold the DNS name of the AC issuer
- x509issuerURI - used to hold a URI for the AC issuer
- x509issuerIPAddress - used to hold the IP address of the AC issuer
- x509issuerRegisteredID - used to hold a registered OID of the AC issuer
- x509authorityKeyIdentifier - used to hold the identifier of the public key used to sign the AC, taken from the attribute cert issuer object digest field

5. Attribute Types for AC Specific Fields

The following attribute types may be used to store basic fields of an AC. The following basic fields are supported:

- x509ACHolderPKCSerialNumber and x509ACHolderPKCissuerDN - used to identify the holder via their public key certificate
- x509ACHolderRfc822Name - identifies the holder via their email address
- x509ACHolderDNSName - identifies the holder via their DNS name
- x509ACHolderDN - identifies the holder via their DN
- x509ACHolderURI - identifies the holder via their URI
- x509ACHolderIPAddress - identifies the holder via their IP address
- x509ACHolderRegisteredID - identifies the holder via a registered OID
- x509ACObjectDigest, x509ACDigestAlgorithm and x509ACDigestedObjectType - identifies the holder via a hash of information directly associated with the holder

5.1 AC holder PKC

The x509ACHolderPKCSerialNumber and x509ACHolderPKCissuerDN attributes are to hold the contents of the holder base certificate ID fields, in order to identify the holder via their public key certificate

5.1.1 AC holder PKC serial number

```
 (1.2.826.0.1.3344810.1.1.61
       NAME 'x509ACHolderPKCSerialNumber'
       DESC 'The serial number of the PKC of the AC holder,
       see RFC3281 4.2.2'
    EQUALITY integerMatch
    ORDERING integerOrderingMatch
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
    SINGLE-VALUE )
```

5.1.2 AC holder PKC issuer DN

```
 (1.2.826.0.1.3344810.1.1.62
       NAME 'x509ACHolderPKCissuerDN'
       DESC 'Distinguished name of the issuer of the PKC belonging to the
       AC holder, see  RFC3281 4.2.2'
       EQUALITY distinguishedNameMatch
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 )
```

5.2 AC Holder general names

The following attributes are used to hold the alternative forms of the general name of the holder. Separate attribute types are defined for all choices of the ASN.1 type "GeneralName" except for "otherName", "x400Address" and "ediPartyName".

5.2.1 Holder RFC 822 name

```
 (1.2.826.0.1.3344810.1.1.63
       NAME 'x509ACHolderRfc822Name'
       DESC 'Internet electronic mail address of the AC holder,
       see RFC3281 4.2.2'
       EQUALITY caseIgnoreIA5Match
       SUBSTR caseIgnoreIA5SubstringsMatch
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute must be encoded according to the syntax given in RFC 822 [11].

5.2.2 Holder DNS name

```
 (1.2.826.0.1.3344810.1.1.64
       NAME 'x509ACHolderDNSName'
       DESC 'Internet domain name of the AC Holder, see
           RFC3281 4.2.2'
       EQUALITY caseIgnoreIA5Match
       SUBSTR caseIgnoreIA5SubstringsMatch
```

```
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute must be encoded as Internet domain names in accordance with RFC1035 [12].

5.2.3 Holder directory name

```
  (1.2.826.0.1.3344810.1.1.65
        NAME 'x509ACHolderDN'
        DESC 'Distinguished name of the AC Holder, see
            RFC3281 4.2.2'
        EQUALITY distinguishedNameMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 )
```

Values of this attribute type must be encoded according to the syntax given in RFC2253 [13].

5.2.4 Holder uniform resource identifier

```
  (1.2.826.0.1.3344810.1.1.66
        NAME  'x509ACHolderURI'
        DESC 'Uniform Resource Identifier of the AC Holder,
        see RFC3281 4.2.2'
        EQUALITY caseExactIA5Match
        SUBSTR caseExactIA5SubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute must be encoded according to the syntax given in RFC2396 [14].

5.2.5 Holder IP address

```
  (1.2.826.0.1.3344810.1.1.67
        NAME 'x509ACHolderIPAddress'
        DESC 'Internet Protocol address of the AC Holder, see
            RFC3281 4.2.2'
        EQUALITY caseIgnoreIA5Match
        SUBSTR caseIgnoreIA5SubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute type must be stored in the syntax given in Appendix B of RFC2373 [16].

5.2.6 Holder registered ID

```
  (1.2.826.0.1.3344810.1.1.68
        NAME 'x509ACHolderRegisteredID'
        DESC 'Any registered OID of the AC holder, see
            RFC3281 4.2.2'
        EQUALITY objectIdentifierMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.38 )
```

registeredID is an identifier of any registered object assigned in accordance
with ITU-T Rec. X.660. [17]

5.3 AC object digest

x509ACObjectDigest, x509ACDigestAlgorithm and x509ACDigestedObjectType are used to hold the contents of the holder object digest info fields. They are used to identify the holder via a hash of information directly associated with the holder.

5.3.1 Object digest

```
( 1.2.826.0.1.3344810.1.1.69
    NAME 'x509ACObjectDigest'
    DESC 'Holds the hash value of the object identified by
        x509ACDigestedObjectType, see RFC 3281, section 7.3'
    EQUALITY bitStringMatch
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.6
    SINGLE-VALUE )
```

5.3.2 Object digest algorithm

```
( 1.2.826.0.1.3344810.1.1.70
    NAME 'x509ACDigestAlgorithm'
    DESC 'OID of the hashing algorithm used to create the
        Object digest, see RFC3281, section 7.3'
    EQUALITY objectIdentifierMatch
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.38
    SINGLE-VALUE )
```

5.3.3 Object type

```
(1.2.826.0.1.3344810.1.1.71
      NAME 'x509ACDigestedObjectType'
      DESC 'Type of object being digested, see RFC3281, section 7.3'
    EQUALITY integerMatch
    ORDERING integerOrderingMatch
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
    SINGLE-VALUE )
```

6. Attributes for Selected AC Extensions

In line with the AC profile RFC 3281 [5], the following AC extensions are supported:

- Audit Identity (defined here)
- AC targets (defined here)
- Authority Key Identifier (defined in [7])
- Authority Information Access (defined in [7])
- CRL distribution points (defined here)

- No revocation (defined here)

(Note. The CRL distribution point attributes defined in [7] were inadequate for our needs)

6.1 Audit identity

This attribute may be used to store the sequence number of the CRL.

```
  (1.2.826.0.1.3344810.1.1.72
        NAME 'x509ACAuditID'
        DESC 'Identity of holder used in audit trails, see RFC3281 4.3.1'
     EQUALITY octetStringMatch
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.40
     SINGLE-VALUE )
```

6.2 AC targets

ACs can be targeted at specific objects, or groups of objects. Objects and groups of objects are identified by their general names. Separate sets of attributes are specified for individual targets and groups of targets. Attribute types are defined for all choices of the ASN.1 type "GeneralName" except for "otherName", "x400Address" and "ediPartyName".

6.2.1 Target RFC 822 name

```
  (1.2.826.0.1.3344810.1.1.73
        NAME 'x509ACTargetRfc822Name'
        DESC 'Internet electronic mail address of the ACs Target,
        see RFC3281 4.3.2'
        EQUALITY caseIgnoreIA5Match
        SUBSTR caseIgnoreIA5SubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute must be encoded according to the syntax given in RFC 822 [11].

6.2.2 Target DNS name

```
  (1.2.826.0.1.3344810.1.1.74
        NAME 'x509ACTargetDNSName'
        DESC 'Internet domain name of the ACs Target, see
            RFC3281 4.3.2'
        EQUALITY caseIgnoreIA5Match
        SUBSTR caseIgnoreIA5SubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute must be encoded as Internet domain names in accordance with RFC1035 [12].

6.2.3 Target directory name

```
  (1.2.826.0.1.3344810.1.1.75
         NAME 'x509ACTargetDN'
         DESC 'Distinguished name of the ACs Target, see
             RFC3281 4.3.2'
         EQUALITY distinguishedNameMatch
         SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 )
```

Values of this attribute type must be encoded according to the syntax given in RFC2253 [13].

6.2.4 Target uniform resource identifier

```
  (1.2.826.0.1.3344810.1.1.76
         NAME  'x509ACTargetURI'
         DESC 'Uniform Resource Identifier of the ACs Target,
         see RFC3281 4.3.2'
         EQUALITY caseExactIA5Match
         SUBSTR caseExactIA5SubstringsMatch
         SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute must be encoded according to the syntax given in RFC2396 [14].

6.2.5 Target IP address

```
  (1.2.826.0.1.3344810.1.1.77
         NAME 'x509ACTargetIPAddress'
         DESC 'Internet Protocol address of the ACs Target, see
             RFC3281 4.3.2'
         EQUALITY caseIgnoreIA5Match
         SUBSTR caseIgnoreIA5SubstringsMatch
         SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute type must be stored in the syntax given in Appendix B of RFC2373 [16].

6.2.6 Target registered ID

```
  (1.2.826.0.1.3344810.1.1.78
         NAME 'x509ACTargetRegisteredID'
         DESC 'Any registered OID of the ACs Target, see
             RFC3281 4.3.2'
         EQUALITY objectIdentifierMatch
         SYNTAX 1.3.6.1.4.1.1466.115.121.1.38 )
```

registeredID is an identifier of any registered object assigned in accordance with ITU-T Rec. X.660. [17]

6.2.7 Target group RFC 822 name

```
  (1.2.826.0.1.3344810.1.1.79
        NAME 'x509ACTargetGroupRfc822Name'
        DESC 'Internet electronic mail address of the ACs Target group
        see RFC3281 4.3.2'
        EQUALITY caseIgnoreIA5Match
        SUBSTR caseIgnoreIA5SubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute must be encoded according to the syntax given in RFC 822 [11].

6.2.8 Target group DNS name

```
  (1.2.826.0.1.3344810.1.1.80
    NAME 'x509ACTargetGroupDNSName'
    DESC 'Internet domain name of the ACs Target group, see
        RFC3281 4.3.2'
    EQUALITY caseIgnoreIA5Match
    SUBSTR caseIgnoreIA5SubstringsMatch
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute must be encoded as Internet domain names in accordance with RFC1035 [12].

6.2.9 Target group directory name

```
  (1.2.826.0.1.3344810.1.1.81
        NAME 'x509ACTargetGroupDN'
        DESC 'Distinguished name of the AC's Target group, see
            RFC3281 4.3.2'
        EQUALITY distinguishedNameMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 )
```

Values of this attribute type must be encoded according to the syntax given in RFC2253 [13].

6.2.10 Target group uniform resource identifier

```
  (1.2.826.0.1.3344810.1.1.82
        NAME  'x509ACTargetGroupURI'
        DESC 'Uniform Resource Identifier of the AC's Target group
        see RFC3281 4.3.2'
        EQUALITY caseExactIA5Match
        SUBSTR caseExactIA5SubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute must be encoded according to the syntax given in RFC2396 [14].

6.2.11 Target group IP address

```
  (1.2.826.0.1.3344810.1.1.83
         NAME 'x509ACTargetGroupIPAddress'
         DESC 'Internet Protocol address of the ACs Target group, see
             RFC3281 4.3.2'
         EQUALITY caseIgnoreIA5Match
         SUBSTR caseIgnoreIA5SubstringsMatch
         SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute type must be stored in the syntax given in Appendix B
of RFC2373 [16].

6.2.12 Target group registered ID

```
  (1.2.826.0.1.3344810.1.1.84
         NAME 'x509ACTargetGroupRegisteredID'
         DESC 'Any registered OID of the ACs Target group, see
             RFC3281 4.3.2'
         EQUALITY objectIdentifierMatch
         SYNTAX 1.3.6.1.4.1.1466.115.121.1.38 )
```

registeredID is an identifier of any registered object assigned in accordance
with ITU-T Rec. X.660. [17]

6.3 No revocation

```
  (1.2.826.0.1.3344810.1.1.85
         NAME 'x509ACNoRevocation'
         DESC 'If true, the AC will never be revoked, see
             RFC3281 section 4.3.6'
         EQUALITY booleanMatch
         SYNTAX 1.3.6.1.4.1.1466.115.121.1.7 )
```

6.4 CRL distribution points

The CRL distribution point extension indicates the locations where CRLs will be
published for this AC. It comprises the general name of the DP, plus optionally
the general name of the CRL issuer (if different from the AC issuer) plus the
reason codes that will be published at this DP. Separate attribute types are
defined for all choices of the ASN.1 type "GeneralName" except for "otherName",
"x400Address" and "ediPartyName". Note that because there can be multiple
distribution points, the multi-valued attributes defined here will not be able
to link each DP with its corresponding reasons and issuer.

If

6.4.1 Distribution point RFC 822 name

```
  (1.2.826.0.1.3344810.1.1.86
         NAME 'x509DPRfc822Name'
         DESC 'Internet electronic mail address of the
```

```
        distribution point, see RFC3280 section 4.2.1.14'
        EQUALITY caseIgnoreIA5Match
        SUBSTR caseIgnoreIA5SubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute must be encoded according to the syntax given in RFC 822 [11].

6.4.2 Distribution point DNS name

```
  (1.2.826.0.1.3344810.1.1.87
        NAME 'x509DPDNSName'
        DESC 'Internet domain name of the distribution point, see
            RFC3280 section 4.2.1.14'
        EQUALITY caseIgnoreIA5Match
        SUBSTR caseIgnoreIA5SubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute must be encoded as Internet domain names in accordance with RFC1035 [12].

6.4.3 Distribution point directory name

```
  (1.2.826.0.1.3344810.1.1.88
        NAME 'x509DPDN'
        DESC 'Distinguished name of the distribution point, see
            RFC3280 section 4.2.1.14'
        EQUALITY distinguishedNameMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 )
```

Values of this attribute type must be encoded according to the syntax given in RFC2253 [13].

6.4.4 Distribution point uniform resource identifier

```
  (1.2.826.0.1.3344810.1.1.89
        NAME  'x509DPURI'
        DESC 'Uniform Resource Identifier of the distribution
        point, see RFC3280 section 4.2.1.14'
        EQUALITY caseExactIA5Match
        SUBSTR caseExactIA5SubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute must be encoded according to the syntax given in RFC2396 [14].

6.4.5 Distribution point IP address

```
  (1.2.826.0.1.3344810.1.1.90
        NAME 'x509DPIPAddress'
        DESC 'Internet Protocol address of the distribution point, see
```

```
          RFC3280 section 4.2.1.14'
      EQUALITY caseIgnoreIA5Match
      SUBSTR caseIgnoreIA5SubstringsMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute type must be stored in the syntax given in Appendix B of RFC2373 [16].

6.4.6 Distribution point registered ID

```
 (1.2.826.0.1.3344810.1.1.91
      NAME 'x509DPRegisteredID'
      DESC 'Any registered OID of the distribution point, see
          RFC3280 section 4.2.1.14'
      EQUALITY objectIdentifierMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.38 )
```

registeredID is an identifier of any registered object assigned in accordance with ITU-T Rec. X.660. [17]

6.4.7 Distribution point name relative to CRL issuer

```
 (1.2.826.0.1.3344810.1.1.92
      NAME 'x509DPrelativeToIssuer'
      DESC 'RDN of the  distribution point, relative to the issuer, see
          RFC3280 section 4.2.1.14'
      EQUALITY distinguishedNameMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 )
```

Values of this attribute type must be encoded according to the syntax given in RFC2253 [13].

6.4.8 Distribution point CRL issuer RFC 822 name

```
 (1.2.826.0.1.3344810.1.1.93
      NAME 'x509DPissuerRfc822Name'
      DESC 'Internet electronic mail address of the
      distribution point CRL issuer, see RFC3280 section 4.2.1.14'
      EQUALITY caseIgnoreIA5Match
      SUBSTR caseIgnoreIA5SubstringsMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute must be encoded according to the syntax given in RFC 822 [11].

6.4.9 Distribution point CRL issuer DNS name

```
 (1.2.826.0.1.3344810.1.1.94
      NAME 'x509DPissuerDNSName'
      DESC 'Internet domain name of the distribution point CRL issuer,
      see RFC3280 section 4.2.1.14'
```

```
        EQUALITY caseIgnoreIA5Match
        SUBSTR caseIgnoreIA5SubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute must be encoded as Internet domain names in accordance with RFC1035 [12].

6.4.10 Distribution point CRL issuer directory name

```
  (1.2.826.0.1.3344810.1.1.95
        NAME 'x509DPissuerDN'
        DESC 'Distinguished name of the distribution point CRL issuer,
        see RFC3280 section 4.2.1.14'
        EQUALITY distinguishedNameMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 )
```

Values of this attribute type must be encoded according to the syntax given in RFC2253 [13].

6.4.11 Distribution point CRL issuer uniform resource identifier

```
  (1.2.826.0.1.3344810.1.1.96
        NAME  'x509DPissuerURI'
        DESC 'Uniform Resource Identifier of the distribution
        point CRL issuer, see RFC3280 section 4.2.1.14'
        EQUALITY caseExactIA5Match
        SUBSTR caseExactIA5SubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute must be encoded according to the syntax given in RFC2396 [14].

6.4.12 Distribution point CRL issuer IP address

```
  (1.2.826.0.1.3344810.1.1.97
        NAME 'x509DPissuerIPAddress'
        DESC 'Internet Protocol address of the distribution point CRL
        issuer, see RFC3280 section 4.2.1.14'
        EQUALITY caseIgnoreIA5Match
        SUBSTR caseIgnoreIA5SubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute type must be stored in the syntax given in Appendix B of RFC2373 [16].

6.4.13 Distribution point CRL issuer registered ID

```
  (1.2.826.0.1.3344810.1.1.98
        NAME 'x509DPissuerRegisteredID'
        DESC 'Any registered OID of the distribution point CRL issuer,
        see  RFC3280 section 4.2.1.14'
        EQUALITY objectIdentifierMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.38 )
```

registeredID is an identifier of any registered object assigned in accordance with ITU-T Rec. X.660. [17]

6.4.14 Distribution point reason codes

This attribute is used to indicate the reason codes associated with the various DPs.

```
  (1.2.826.0.1.3344810.1.1.99
        NAME 'x509DPReasonCodes'
        DESC 'The reason codes used by a DP, see
            RFC3280 section 4.2.1.14'
        EQUALITY bitStringMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.6 )
```

7. Security Considerations

This [Internet Draft/Standard] describes the subschema for the storage and matching of PKI attributes derived from CRLs. It does not address the protocol for the storage and retrieval of this information.

LDAP servers SHOULD use authentication and access control mechanisms to protect the information during its storage and retrieval.

8. IANA Considerations

This document uses the OID node 1.2.826.0.1.3344810 to identify the LDAP schema elements described here. This OID is assigned to TrueTrust Ltd, under its BSI assigned English/Welsh Registered Company number [18].

9. References

Normative

[1] Bradner, S. The Internet Standards Process -- Revision 3. RFC 2026 October 1996.

[2] S.Bradner. "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.

[3] Wahl, M., Coulbeck, A., Howes, T. and S. Kille, "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions", RFC 2252, December 1997.

[4] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[5] Farrell, S., Housley, R. "An Internet Attribute Certificate Profile for Authorization", RFC 3281, April 2002.

[6] ITU, "Information Technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks", ITU-T Recommendation X.509, March 2000.

[7] Gietz, P., Klasen, N. "Internet X.509 Public Key Infrastructure Lightweight Directory Access Protocol Schema for X.509 Certificates", <draft-ietf-pkix-ldap-pkc-schema-00.txt>, June 2004

[8] Chadwick, D.W., Sahalayev, M. V. "Internet X.509 Public Key Infrastructure LDAP Schema for X.509 CRLs", <draft-ietf-pkix-ldap-crl-schema-01.txt>, June 2003

[10] Chadwick, D.W., Legg, S. "Internet X.509 Public Key Infrastructure - LDAP Schema for PMIs" <draft-ietf-pkix-ldap-pmi-schema-00.txt>, July 2002

[11] Crocker, D., "Standard for the format of ARPA Internet text messages", STD 11, RFC 822, August 1982.

[12] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.

[13] Wahl, M., Kille, S. and T. Howes, "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names", RFC 2253, December 1997.

[14] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[15] Hodges, J. and RL. Morgan, "Lightweight Directory Access Protocol (v3): Technical Specification", RFC 3377, September 2002.

[16] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 2373, July 1998.

[17] CCITT Recommendation X.660 (1992) | ISO/IEC 9834-1:1993, Information technology - Open Systems Interconnection - Procedures for the operation of OSI Registration Authorities: General procedures.

[18] BRITISH STANDARD BS 7453 Part 1. Procedures for UK Registration for Open System Standards Part 1: Procedures for the UK Name Registration Authority.

[21] Bradner, S., "IETF Rights in Contributions", BCP 78, RFC3667, February 2004.

[22] Bradner, S., Ed., "Intellectual Property Rights in IETF Technology", BCP 79, RFC 3668, February 2004.

Informative

[9] Chadwick, D.W., Legg, S. "Internet X.509 Public Key Infrastructure - LDAP Schema for PKIs " <draft-ietf-pkix-ldap-pki-schema-00.txt>, July 2002

[19] S. Legg. "Lightweight Directory Access Protocol (LDAP) and X.500 Component Matching Rules" RFC 3687, February 2004

[20] J. Allen, M. Mealling. "The Architecture of the Common Indexing Protocol (CIP)". RFC 2651. August 1999.

10. Intellectual Property Notice

11. Acknowledgments

The authors would like to thank Peter Gietz for his help and co-operation, and in particular his willingness to align [7] with this document.

The authors would also like to thank TERENA, CESNET, SURFnet, UNINETT, RedIRIS and SWITCH, who jointly partially funded this work, and without whose support this work would not have been possible.

12. Copyright

13. Authors' Addresses

David Chadwick, Mikhail Sahalayev
IS Institute
University of Salford
Salford
England
M5 4WT

Email: d.w.chadwick@salford.ac.uk
M.Sahalayev@pgr.salford.ac.uk

14. Changes

Changes from <draft-ietf-pkix-ldap-ac-schema-00.txt>

1. Have added a section about attributes embedded in ACs.
2. Have aligned schema with <draft-klasen-ldap-x509certificate-schema-02.txt>
3. Have altered object class structure to introduce auxiliary object classes for certificate extensions
4. Have adjusted upper/lower case of components of attribute type names to be consistent
5. Have changed matching rules of x509ACTargetGroupDNSName to be IA5 matching rules
6. Minor editorial corrections
7. Changed from Standards Track to Informational after discussions with area and WG leaders.
8. Have added an IANA considerations section and Acknowledgment section

Appendix 7. LDAP Schema for X.509 CRLs

INTERNET-DRAFT D. W. Chadwick
PKIX WG M. V. Sahalayev
Intended Category: Informational University of Salford
Expires on 28 December 2004 28 June 2004

Internet X.509 Public Key Infrastructure
LDAP Schema for X.509 CRLs
<draft-ietf-pkix-ldap-crl-schema-02.txt>

STATUS OF THIS MEMO

Comments and suggestions on this document are encouraged. Comments on this document should be sent to the PKIX working group discussion list <ietf-pkix@imc.org> or directly to the authors.

ABSTRACT

This document describes an LDAP schema for X.509 CRLs. Each CRL is broken down into a set of attribute types. These attributes can then be stored in a CRL entry, or set of entries. Object classes are defined for these CRL entries. Each attribute type uses an existing LDAP syntax, so that no new matching rules need to be defined.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [2].

1. Introduction

It currently isn't possible to search LDAP servers for X.509 [6] attributes (public key certificates, CRLs etc.) as no matching rules have been defined for them. A couple of Internet Drafts [9,10] have been specified, but implementation of them is complex. Component matching [19] defines a mechanism for matching against complex syntaxes, by defining generic matching rules that can match against any user selected component parts in an attribute value of any arbitrarily complex attribute syntax. This might prove to be the proper way to solve LDAP search problems in the longer term, but it will take a long time until such ASN.1 based mechanisms are implemented in all LDAP servers and clients. Even when this has happened the mechanism proposed in this document will still be useful to some applications such as CIP [20].

A simple and easy to implement mechanism is needed today to search for X.509 attributes.

Rather than search for an X.509 attribute in an entry, it suggests the directory administrative user creates an entry (in the case of pubic key and attribute certificates) or a subtree (in the case of CRLs) from the X.509 attribute. The attributes of these new entries will be created from fields of the X.509 attribute (e.g. the issuer field), and if these new attributes are defined using existing LDAP syntaxes and matching rules, then it will be possible to use existing LDAP server technology to search for fields in X.509 attributes.

This document is one of a set comprising:
i)	the LDAP schema for X.509 public key certificates [7]
ii)	the LDAP schema for X.509 attribute certificates [8]
iii)	the LDAP schema for X.509 CRLs (this document)

Schema definitions are provided using LDAPv3 description formats from RFC2252 [3]. Definitions provided here are formatted (line wrapped) for readability. The specifications use the augmented Backus-Naur Form (ABNF) as described in RFC2234 [4].

2. DIT Structure and Naming

If the schema presented in this document is used to store information about CRLs in a directory, each CRL entry SHOULD be stored as a direct subordinate of the CRL issuer's entry, unless a crlScope Extension or issuingDistributionPoint Extension is present in the CRL. In the latter cases the CRL MUST be stored as indicated in those extensions. The CRL entry is named using the x509CRLThisUpdate attribute, using one of the following name forms:

```
 (1.2.826.0.1.3344810.1.3.0
     NAME 'x509CRLNameForm'
     OC x509CRL
     MUST x509CRLThisUpdate )

 (1.2.826.0.1.3344810.1.3.7
     NAME 'x509ARLNameForm'
     OC x509CRL
     MUST x509CRLThisUpdate )

 (1.2.826.0.1.3344810.1.3.8
     NAME 'x509deltaRLNameForm'
     OC x509CRL
     MUST x509CRLThisUpdate )
```

x509CRLNameForm is used to name CRL entries.
x509ARLNameForm is used to name Authority Revocation List entries.
x509deltaRLNameForm is used to name delta CRL entries.

Subordinate to the entry for the CRL, the user MAY create an entry for each revoked certificate. Each revoked certificate entry is named with the serial number of the revoked certificate, unless the CRL is an indirect CRL, in which case it is named with a combination of the serial number and issuer's DN i.e. a multi-valued RDN. For those LDAP servers that do not support multi-valued RDNs, an alternative name form is defined in which the serial number and issuer DN are concatenated together into one attribute value. This is expressed in the following name forms:

```
  (1.2.826.0.1.3344810.1.3.1
      NAME 'x509CRLentryNameForm'
      OC x509CRLentry
      MUST x509serial )

  (1.2.826.0.1.3344810.1.3.2
      NAME 'x509CRLindirectEntryNameForm'
      OC x509CRLentry
      MUST ( x509serialNumber $ x509issuer ) )

  (1.2.826.0.1.3344810.1.3.6
      NAME 'x509CRLindirectEntryAltNameForm'
      OC x509AC
      MUST ( x509issuerSerial ) )
```

When encoding DNs that contain an x509issuer field, the string representation must be made according to [13]. These strings contain RFC2253 special characters and must therefore be escaped. For example, the issuer name in a certificate may be:

```
  x509issuer: OU=VeriSign Trust Network,OU=(c) 1998 VeriSign Inc. -
    For authorized use only,OU=Class 1 Public Primary Certification Au
    thority - G2,O=VeriSign Inc.,C=US
```

When used in the x509issuerSerial attribute of a DN, this may appear as:

```
  dn: x509issuerSerial=123456\,OU\=VeriSign Trust Network \,OU
   \=(c) 1998 VeriSign Inc. - For authorized use only\,OU\=Class 1 Public
   Primary Certification Authority - G2\,O\=VeriSign Inc.\2cC\3dUS
```

3. X.509 Schema Object Classes

The object classes have been designed to form a logical set and be extensible in
an orderly way as new PKC/CRL/AC extensions are defined. The methodology is as
follows. Every X.509 entry (for a PKC, CRL or AC) is of the x509base abstract
object class. There is then an additional abstract object class for each,
derived from x509base, which holds the attributes extracted from the basic
PKC/AC/CRL ASN.1 structure (excluding all extensions). Further, there is an
auxiliary object class for the extensions defined in X.509 [6], and an
additional auxiliary object class (if needed) for extensions defined in existing
Internet RFCs. As new extensions are defined, then new auxiliary object classes
and attributes will need to be defined to cater for the attributes to be
extracted from these. Finally there are several structural object classes for
each, which allow the X.509 DER encoded attribute to be stored in the entry.

3.1 X509 base object class

The x509base object class is defined in [7].

3.2 X.509 CRL object class

The x509CRL object class is the abstract object class used for storing the
searchable attributes extracted from the basic fields of a CRL (excluding
extensions). It is the superior of the revocation list structural object
classes.

```
 (1.2.826.0.1.3344810.1.0.14
    NAME 'x509CRL'
    SUP x509base
    ABSTRACT
    MUST ( x509CRLThisUpdate $
          x509signatureAlgorithm $
          x509issuer )
    MAY ( x509CRLNextUpdate ))
```

3.3 X509 CRL extensions object class

The x509CRLext auxiliary object class is used to hold the attributes extracted
from the extensions defined in the X.509 standard [6] and profiled in [5]. Note
that the freshest CRL extension is not included since this was originally
forbidden from appearing in CRLs by X.509.

Note. If a CRL holds additional extensions to these, then another auxiliary
object class and supporting attributes will need to be defined.

```
(1.2.826.0.1.3344810.1.0.23
    NAME 'x509CRLext'
    AUXILIARY
```

```
 MAY ( x509authorityKeyIdentifier $
       x509authorityCertIssuer $
       x509authorityCertSerialNumber $
       x509issuerRfc822Name $
       x509issuerDNSName $
       x509issuerDirectoryName $
       x509issuerUniformResourceIdentifier $
       x509issuerIPAddress $
       x509issuerRegisteredID $
       x509CRLNumber $
       x509CRLDPRfc822Name $
       x509CRLDPDNSName $
       x509CRLDPDN $
       x509CRLDPURI $
       x509CRLDPIPAddress $
       x509CRLDPRegisteredID $
       x509CRLDPOnlyUserCerts $
       x509CRLDPOnlyCACerts $
       x509CRLDPOnlySomeReasons $
       x509CRLDPOnlyAttCerts $
       x509CRLDPindirect $
       x509CRLDeltaIndicator ) )
```

3.4 X.509 certificate revocation list object class

This object class is the structural object class used to hold certificate revocation list entries.

```
 (1.2.826.0.1.3344810.1.0.19
   NAME 'x509certificateRevocationList'
   SUP x509CRL
   MUST certificateRevocationList )
```

3.4 X.509 authority revocation list object class

This object class is the structural object class used to hold authority revocation list entries.

```
 (1.2.826.0.1.3344810.1.0.20
   NAME 'x509authorityRevocationList'
   SUP x509CRL
   MUST authorityRevocationList )
```

3.5 X.509 delta revocation list object class

This object class is the structural object class used to hold delta revocation list entries.

```
 (1.2.826.0.1.3344810.1.0.21
    NAME 'x509deltaRevocationList'
    SUP x509CRL
    MUST deltaRevocationList )
```

3.6 X.509 revoked certificate object class

The x509CRLentry object class is the structural object class used for storing revoked certificate entries.

```
  (1.2.826.0.1.3344810.1.0.15
        NAME 'x509CRLentry'
        SUP x509base
        MUST ( x509serialNumber $
         x509CRLCertRevocationDate )
        MAY ( x509CRLCertInvalidityDate $
        x509CRLCertReasonCode $
        x509CRLCertHoldInstructionCode $
        x509CRLCertIssuerRfc822Name $
        x509CRLCertIssuerDnsName $
        x509CRLCertIssuerDN $
        x509CRLCertIssuerURI $
        x509CRLCertIssuerIpAddress $
        x509CRLCertIssuerRegisteredID )  )
```

4. Common X.509 Attribute Types

The following attribute types defined in [7] are used to hold the corresponding fields of CRLs:

- x509serialNumber - used to hold the serial number(s) of the revoked certifictate(s)
- x509version - used to hold the version of the CRL
- x509signatureAlgorithm - used to hold the OID of the algorithm used to sign the CRL
- x509issuer - used to hold the DN of the CRL issuer
- x509issuerRfc822Name - used to hold the email address of the CRL issuer
- x509issuerDnsName - used to hold the DNS name of the CRL issuer
- x509issuerDirectoryName - used to hold an alternative DN for the CRL issuer
- x509issuerUniformResourceIdentifier - used to hold a URI for the CRL issuer
- x509issuerIpAddress - used to hold the IP address of the CRL issuer
- x509issuerRegisteredID - used to hold a registered OID of the CRL issuer
- x509authorityKeyIdentifier - used to hold the identifier of the key used to sign the CRL
- x509authorityCertIssuer - used in conjunction with x509authorityCertSerialNumber to identify the certificate of the issuer
- x509authorityCertSerialNumber - used in conjunction with x509authorityCertIssuer to identify the certificate of the issuer

5. Attribute Types for CRL Specific Fields

The following attribute types may be used to store basic fields of a CRL. The following basic fields are supported:

- this update
- next update

5.1 This update

This attribute may be used to hold the thisUpdate field of the CRL.

```
   (1.2.826.0.1.3344810.1.1.37
           NAME 'x509CRLThisUpdate'
           DESC 'Date at which this revocation list was issued - see RFC3280 5.1.2.4'
           EQUALITY generalizedTimeMatch
           ORDERING generalizedTimeOrderingMatch
           SYNTAX 1.3.6.1.4.1.1466.115.121.1.24
           SINGLE-VALUE )
```

Note that the field in the CRL may be in UTC or GeneralizedTime format. If in UTC format, the creator of this attribute MUST convert the UTC time into GeneralisedTime format when creating the attribute value.

5.2 Next update

This attribute may be used to hold the nextUpdate field of the CRL.

```
   (1.2.826.0.1.3344810.1.1.38
           NAME 'x509CRLNextUpdate'
           DESC 'Date by which the next revocation list in this series
                    will be issued, see - RFC3280 5.1.2.5'
           EQUALITY generalizedTimeMatch
           ORDERING generalizedTimeOrderingMatch
           SYNTAX 1.3.6.1.4.1.1466.115.121.1.24
           SINGLE-VALUE )
```

Note that the field in the CRL may be in UTC or GeneralizedTime format. If in UTC format, the creator of this attribute MUST convert the UTC time into GeneralisedTime format when creating the attribute value.

6. Attributes for Selected CRL Extensions

In line with the CRL profile RFC 3280 [5], the following CRL extensions are supported:

- CRL Number (defined here)
- Issuing Distribution Point (defined here)
- Authority Key Identifier (defined in [7])
- Issuer Alternative Name (defined in [7])
- Delta CRL Indicator (defined here)

The following extension is not included:

- freshest CRL (see 5.2.6 of RFC 3280 [5])

6.1 CRL number extension

This attribute may be used to store the sequence number of the CRL.

```
  (1.2.826.0.1.3344810.1.1.102
      NAME 'x509CRLNumber'
      DESC 'sequence number of issued CRL - see RFC3280 5.2.3'
      EQUALITY integerMatch
      ORDERING integerOrderingMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
      SINGLE-VALUE )
```

6.2 Issuing distribution point

The issuing distribution point comprises the general name of the issuing CA, plus some codes that indicate the contents of this CRL. Separate attribute types are defined for all choices of the ASN.1 type "GeneralName" except for "otherName", "x400Address" and "ediPartyName".

6.2.1 Issuing distribution point RFC 822 name

```
  (1.2.826.0.1.3344810.1.1.48
        NAME 'x509CRLDPRfc822Name'
        DESC 'Internet electronic mail address of the issuing
        distribution point, see RFC3280 5.2.5'
        EQUALITY caseIgnoreIA5Match
        SUBSTR caseIgnoreIA5SubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute must be encoded according to the syntax given in RFC 822 [11].

6.2.2 Issuing distribution point DNS name

```
  (1.2.826.0.1.3344810.1.1.49
        NAME 'x509CRLDPDnsName'
        DESC 'Internet domain name of the issuing distribution point, see
            RFC3280 5.2.5'
        EQUALITY caseIgnoreIA5Match
        SUBSTR caseIgnoreIA5SubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute must be encoded as Internet domain names in accordance with RFC1035 [12].

6.2.3 Issuing distribution point directory name

```
  (1.2.826.0.1.3344810.1.1.50
        NAME 'x509CRLDPDN'
        DESC 'Distinguished name of the issuing distribution point, see
```

```
          RFC3280 5.2.5'
      EQUALITY distinguishedNameMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 )
```

Values of this attribute type must be encoded according to the syntax given in RFC2253 [13].

6.2.4 Issuing distribution point uniform resource identifier

```
 (1.2.826.0.1.3344810.1.1.51
      NAME  'x509CRLDPURI'
      DESC 'Uniform Resource Identifier of the issuing distribution
      point, see RFC3280 5.2.5'
      EQUALITY caseExactIA5Match
      SUBSTR caseExactIA5SubstringsMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute must be encoded according to the syntax given in RFC2396 [14].

6.2.5 Issuing distribution point IP address

```
 (1.2.826.0.1.3344810.1.1.52
      NAME 'x509CRLDPIpAddress'
      DESC 'Internet Protocol address, of the issuing distribution point, see
          RFC3280 5.2.5'
      EQUALITY caseIgnoreIA5Match
      SUBSTR caseIgnoreIA5SubstringsMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute type must be stored in the syntax given in Appendix B of RFC2373 [16].

6.2.6 Issuing distribution point registered ID

```
 (1.2.826.0.1.3344810.1.1.53
      NAME 'x509CRLDPRegisteredID'
      DESC 'Any registered OID of the certificate issuer, see
          RFC3280 5.2.5'
      EQUALITY objectIdentifierMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.38 )
```

registeredID is an identifier of any registered object assigned in accordance with ITU-T Rec. X.660. [17]

6.2.7 Issuing distribution point only contains user certs

This attribute may be used to indicate if the CRL only contains revocations for end-entity certificates.

```
 (1.2.826.0.1.3344810.1.1.54
        NAME 'x509CRLDPOnlyUserCerts'
        DESC 'If true, the CRL only contains revocations for end-entity certs, see
            RFC3280 5.2.5'
        EQUALITY booleanMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.7 )
```

6.2.8 Issuing distribution point only contains CA certs

This attribute may be used to indicate if the CRL only contains revocations for CA certificates.

```
  (1.2.826.0.1.3344810.1.1.55
        NAME 'x509CRLDPOnlyCACerts'
        DESC 'If true, the CRL only contains revocations for CA certs, see
            RFC3280 5.2.5'
        EQUALITY booleanMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.7 )
```

6.2.9 Issuing distribution point only contains some reasons

This attribute may be used to indicate if the CRL only contains some revocation reason codes.

```
  (1.2.826.0.1.3344810.1.1.56
        NAME 'x509CRLDPOnlySomeReasons'
        DESC 'If true, the CRL only contains some revocation reason codes, see
            RFC3280 5.2.5'
        EQUALITY bitStringMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.6 )
```

6.2.10 Issuing distribution point only contains attribute certs

This attribute may be used to indicate if the CRL only contains revocations for attribute certificates.

```
  (1.2.826.0.1.3344810.1.1.57
        NAME 'x509CRLDPOnlyAttCerts'
        DESC 'If true, the CRL only contains revocations for attribute certs, see
            RFC3280 5.2.5'
        EQUALITY booleanMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.7 )
```

6.2.11 Issuing distribution point is indirect

This attribute may be used to indicate if the CRL is an indirect CRL and holds revocations of certificates issued by authorities other than the CRL issuer.

```
  (1.2.826.0.1.3344810.1.1.58
        NAME 'x509CRLDPindirect'
        DESC 'If true, the CRL is an indirect CRL, see
            RFC3280 5.2.5'
        EQUALITY booleanMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.7 )
```

6.3 Delta CRL Indicator

This attribute may be used to indicate if the CRL is a delta CRL.

```
  (1.2.826.0.1.3344810.1.1.59
        NAME 'x509CRLDeltaIndicator'
        DESC 'Indicates this is a delta CRL, and the value points to the
        sequence number of the issued base CRL to which this is a delta
        - see RFC3280 5.2.4'
     EQUALITY integerMatch
     ORDERING integerOrderingMatch
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
     SINGLE-VALUE )
```

7. Attributes for CRL Revoked Certificate Fields

The serial number attribute is as defined in [7].

7.1 Revocation date

This attribute may be used to hold the revocationDate field of a CRL entry.

```
  (1.2.826.0.1.3344810.1.1.39
        NAME 'x509CRLCertRevocationDate'
        DESC 'Date/time the CA actually revoked the certificate,
        see - RFC3280 5.1.2.6'
        EQUALITY generalizedTimeMatch
        ORDERING generalizedTimeOrderingMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.24
        SINGLE-VALUE )
```

Note that the field in the CRL may be in UTC or GeneralizedTime format. If in UTC format, the creator of this attribute MUST convert the UTC time into GeneralisedTime format when creating the attribute value.

8. Attributes for Selected CRL Entry Extensions

In line with the CRL profile RFC 3280 [5], the following CRL entry extensions are supported:

- Invalidity date (defined here)
- Certificate issuer (defined here)
- Reason code (defined here)
- Hold instruction code (defined here)

8.1 Invalidity date extension

This attribute may be used to hold the invalidity date of a certificate.

```
  (1.2.826.0.1.3344810.1.1.40
        NAME 'x509CRLCertInvalidityDate'
        DESC 'date at which it is known or suspected that the private
            key was compromised, see RFC3280 5.3.3'
        EQUALITY generalizedTimeMatch
        ORDERING generalizedTimeOrderingMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.24
        SINGLE-VALUE )
```

8.2 Certificate issuer extension

The certificate issuer extension is used in indirect CRLs to identify the issuer of the certificate that is revoked. Separate attribute types are defined for all choices of the ASN.1 type "GeneralName" except for "otherName", "x400Address" and "ediPartyName".

8.2.1 Certificate issuer RFC 822 name

```
  (1.2.826.0.1.3344810.1.1.41
        NAME 'x509CRLCertIssuerRfc822Name'
        DESC 'Internet electronic mail address of the certificate issuer, see
            RFC3280 5.3.4'
        EQUALITY caseIgnoreIA5Match
        SUBSTR caseIgnoreIA5SubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute must be encoded according to the syntax given in RFC 822 [11].

8.2.2 Certificate issuer DNS name

```
  (1.2.826.0.1.3344810.1.1.42
        NAME 'x509CRLCertIssuerDnsName'
        DESC 'Internet domain name of the certificate issuer, see
            RFC3280 5.3.4'
        EQUALITY caseIgnoreIA5Match
        SUBSTR caseIgnoreIA5SubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute must be encoded as Internet domain names in accordance with RFC1035 [12].

8.2.3 Certificate issuer directory name

```
  (1.2.826.0.1.3344810.1.1.43
        NAME 'x509CRLCertIssuerDN'
        DESC 'Distinguished name of the certificate issuer, see
            RFC3280 5.3.4'
        EQUALITY distinguishedNameMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 )
```

Values of this attribute type must be encoded according to the syntax given in RFC2253 [13].

8.2.4 Certificate issuer uniform resource identifier

```
  (1.2.826.0.1.3344810.1.1.44
        NAME  'x509CRLCertIssuerURI'
        DESC 'Uniform Resource Identifier of the certificate issuer, see
            RFC3280 5.3.4'
        EQUALITY caseExactIA5Match
        SUBSTR caseExactIA5SubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute must be encoded according to the syntax given in RFC2396 [14].

8.2.5 Certificate issuer IP address

```
  (1.2.826.0.1.3344810.1.1.45
        NAME 'x509CRLCertIssuerIpAddress'
        DESC 'Internet Protocol address, of the certificate issuer, see
            RFC3280 5.3.4'
        EQUALITY caseIgnoreIA5Match
        SUBSTR caseIgnoreIA5SubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Values of this attribute type must be stored in the syntax given in Appendix B of RFC2373 [16].

8.2.6 Certificate issuer registered ID

```
  (1.2.826.0.1.3344810.1.1.46
        NAME 'x509CRLCertIssuerRegisteredID'
        DESC 'Any registered OID of the certificate issuer, see
            RFC3280 5.3.4'
        EQUALITY objectIdentifierMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.38 )
```

registeredID is an identifier of any registered object assigned in accordance with ITU-T Rec. X.660. [17]

8.3 Revocation reason code

This field may be used to hold the coded reason for the revocation

```
  (1.2.826.0.1.3344810.1.1.47
    NAME 'x509CRLCertReasonCode'
    DESC 'An integer code indicating the reason for the revocation, see
            RFC3280 5.3.1'
    EQUALITY integerMatch
    ORDERING integerOrderingMatch
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 )
```

8.4 Hold instruction code

This attribute may be used to store the hold instruction code for the certificate on the CRL.

```
  (1.2.826.0.1.3344810.1.1.103
        NAME 'x509CRLCertHoldInstructionCode'
        DESC 'Any registered OID indicating a hold instruction, see
            RFC3280 5.3.2'
        EQUALITY objectIdentifierMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.38 )
```

9. Security Considerations

This [Internet Draft/Standard] describes the subschema for the storage and matching of PKI attributes derived from CRLs. It does not address the protocol for the storage and retrieval of this information.

LDAP servers SHOULD use authentication and access control mechanisms to protect the information during its storage and retrieval.

10. IANA Considerations

This document uses the OID branch 1.2.826.0.1.3344810 to identify new LDAP attribute types and object classes. A register of all OIDs allocated under this branch is kept by the registered holder. This branch has been assigned to TrueTrust Ltd, under its BSI assigned English/Welsh Registered Company number [18].

11. References

Normative

[1] Bradner, S. The Internet Standards Process -- Revision 3. RFC 2026 October 1996.

[2] S.Bradner. "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.

[3] Wahl, M., Coulbeck, A., Howes, T. and S. Kille, "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions", RFC 2252, December 1997.

[4] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[5] Housley, R., Polk, T., Ford, W. and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and CRL Profile", RFC 3280, April 2002.

[6] ITU, "Information Technology - Open Systems Interconnection - The

Directory: Public-key and attribute certificate frameworks", ITU-T Recommendation X.509, March 2000.

[7] Gietz, P., Klasen, N. "Internet X.509 Public Key Infrastructure Lightweight Directory Access Protocol Schema for X.509 Certificates", <draft-ietf-pkix-ldap-pkc-schema-00.txt>, June 2004

[11] Crocker, D., "Standard for the format of ARPA Internet text messages", STD 11, RFC 822, August 1982.

[12] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.

[13] Wahl, M., Kille, S. and T. Howes, "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names", RFC 2253, December 1997.

[14] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[15] Hodges, J. and RL. Morgan, "Lightweight Directory Access Protocol (v3): Technical Specification", RFC 3377, September 2002.

[16] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 2373, July 1998.

[17] CCITT Recommendation X.660 (1992) | ISO/IEC 9834-1:1993, Information technology - Open Systems Interconnection - Procedures for the operation of OSI Registration Authorities: General procedures.

[18] BRITISH STANDARD BS 7453 Part 1. Procedures for UK Registration for Open System Standards Part 1: Procedures for the UK Name Registration Authority.

Informative

[8] Chadwick, D.W., Sahalayev, M. V. "Internet X.509 Public Key Infrastructure LDAP Schema for X.509 Attribute Certificates", <draft-ietf-pkix-ldap-ac-schema-02.txt>, June 2004

[9] Chadwick, D.W., Legg, S. "Internet X.509 Public Key Infrastructure - LDAP Schema for PKIs " <draft-ietf-pkix-ldap-pki-schema-00.txt>, July 2002

[10] Chadwick, D.W., Legg, S. "Internet X.509 Public Key Infrastructure - LDAP Schema for PMIs" <draft-ietf-pkix-ldap-pmi-schema-00.txt>, July 2002
[19] S. Legg. "Lightweight Directory Access Protocol (LDAP) and X.500 Component Matching Rules" RFC 3687, February 2004

[20] J. Allen, M. Mealling. "The Architecture of the Common Indexing Protocol (CIP)". RFC 2651. August 1999.

12. Intellectual Property Notice

13. Acknowledgments

The authors would like to thank Peter Gietz for his help and co-operation, and
in particular his willingness to align [7] with this document.

The authors would also like to thank TERENA, CESNET, SURFnet, UNINETT, RedIRIS
and SWITCH, who jointly partially funded this work, and without whose support
this work would not have been possible.

14. Copyright

15. Authors' Addresses

David Chadwick, Mikhail Sahalayev
IS Institute
University of Salford
Salford
England
M5 4WT

Email: d.w.chadwick@salford.ac.uk
M.Sahalayev@pgr.salford.ac.uk

16. Changes

Changes since version 00.

1. An alternative name form for revoked certificate entries has been added for those LDAP servers that don't support multiple valued RDNs.
2. Name forms for delta CRL entries and authority revocation list entries have been added
3. x509CRL object class has been changed to ABSTRACT and three new STRUCTURAL object classes have been added for CRLs, ARLs and delta CRLs.
4. Corrected some OID assignments
5. Minor editorial corrections

Changes since version 01

1. The object class structure has been re-vamped and x509base has been transferred to the certificate schema ID [7]
2. Minor editorial corrections
3. Changed from Standards Track to Informational after discussions with area and WP leaders.
4. Inserted IANA considerations section and Acknowledgement section

Appendix 8. Final Detailed Design of an LDAP X.509 Parsing Server

M. Sahalayev, D. W. Chadwick

Release Number	*Release Date*	*Comments*
Version 1.0	15 November 2002	For public comment
Version 1.1	28 November 2002	Added Critical extensions on Modifies causes rejection. Added if HasSubordinates is missing. Removed storing to-be-deleted parent entry in WAL. Minor editorials
Version 1.2	3 February 2003	Changed design for CRLs. Now a CRL creates a subtree of entries rather than a single one.
Version 1.3	12 February 2003	Correcting bugs in the Name Form section for revoked certificate entries.
Version 1.4	13 February 2003	Added a references for detailed design of the parsing functions. Misc cleanup
Version 1.5	20 February 2003	Added new cofig option x509attrtypespath
Version 1.6	20 March 2003	Added new config option for attribute types
Version 1.7	12 May 2003	Added new option for ASN.1 ->X.509 attribute types mapping Renamed several internal functions Removed "Specifics of the *** operation for external server. Its functionality will be provided by back-ldap backend Removed LDAP API functions (they will not be used, back-ldap will be used instead)
Version 1.8	8 August 2003	Redesign of Delete operation to do 1 level search instead of full subtree search
Version 1.9	20 April 2004	Changed matching rule for Modify Delete value to exact matching rule from binary match

Purpose

The X.509 Certificate Parsing Server (XPS) is an intermediary server that sits between an LDAP administrative client and LDAP server. It does not sit between an LDAP retrieval client and LDAP server. Its purpose is to extract certificate and CRL attributes from Add and Modify commands and to create subordinate certificate and CRL entries beneath the target entry that was destined to hold the certificate or CRL. The certificates and CRLs are broken up into their component data fields and attributes are created from each of them and placed in the certificate entry or CRL subtree of entries. For a CRL, a subtree of entries is created, consisting of a root CRL entry (mandatory - holding the CRL fields, CRL extensions and set of revoked certificate serial numbers) and a set of subordinate CRL entry entries (optional - holding the serial number, revocation date and CRL entry extensions). In this way LDAP clients can Search for certificates, CRLs and CRL entries that contain certain fields, without LDAP servers needing to be modified. *Note. This implementation of the XPS will be embedded inside an OpenLDAP server, so that XPS intercepts the LDAP operations before passing them to the OpenLDAP server*.

Limitations

There are a number of limitations with this approach, as follows.
Administrative clients and retrieval clients will have different views of the DIT. An administrative client thinks that a certificate or CRL is held in the entry that it was destined for, whereas in fact it is actually held in (a) subordinate(s) of the entry. (A configuration parameter will say if the attribute is held in the entry as well, but this will increase the storage requirements even more - see next point) LDAP clients will only find the subordinate entries in a Search operation and wont be able to retrieve the parent entry (where all the other attributes of the object are held) along with the certificate, CRL or CRL entry.
The storage space in the LDAP server is considerably increased as the subordinate entries will hold the original certificate or CRL as well as the extracted attributes (the storage will most likely be more than doubled, although actual values will have to be determined after implementation).

LDAP clients will not be able to perform Search operations that are looking for an entry containing more than one certificate, or an entry containing a certificate and other user attributes, or a CRL containing a particular CRL entry extension and CRL extension (although they cannot do this today anyway, except perhaps for Searches that contain a certificate exact match).

Detailed Design

In this design we differentiate between LDAP X.509 administrative operations and all other LDAP operations. The former are defined as:
An AddRequest for an entry holding an X.509 attribute in its AttributeList
A ModifyRequest with an X.509 attribute in any of its modification parameters
Any Delete Request
For the purpose of this design, X.509 attributes will be assumed to be any attribute in the X.509 specification [X509].

XPS will be based on OpenLDAP and implemented as a part of it. The configuration file (slapd.conf) will be extended in order to support XPS and configure it according to the administrators' needs.
OpenLDAP has the ability to support an internal database or a back-end LDAP server. This feature remains unaltered with the XPS modifications, so that administrators can use OpenLDAP-XPS to front end their existing LDAP server.

The WAL

The XPS server is quite complex, as it has to be able treat a set of LDAP operations emanating from a single administrative operation as a single transaction. This is made more difficult in that LDAP does not support transactions. Clearly one could build a simpler XPS implementation that only kept state information in memory, but then if the XPS server crashed the LDAP server would be left in an indeterminate state with respect to its X.509 attributes. This is clearly very undesirable given the importance of CRLs and certificates.
To avoid these problems we will use a Write Ahead Log (WAL). The information about all entries which are about to be added or deleted will be kept on the disk in the WAL and if there are any errors during the operations LDAP will be rolled back to its original state. The format of the WAL is a standard LDIF file [LDIF]
If it is an ADD request only the DN of the entries[1] will be required for rolling back. For adding a DN into the WAL we will create the xps_dn2wal() function. It will take a pointer to a dn as an input parameter and will return an error/success code.
In case of a DELETE request whole entries[2] should be saved in the WAL. For this purpose the function xps_entry2wal() will be created. It will require a pointer to the Entry structure as an input parameter and will return an error/success code.
MODIFY requests will require more information for rolling back. We will store all of the children of the entry to be modified. XPS will call xps_dn2wal() if the type of operation is add or xps_entry2wal() if it is delete. We will not need a separate code if the type of operation is replace, because in this case XPS will first delete all the child entries, then add the new ones. This will be logged as two separate WAL operations.
In the event that XPS or even the operating system crashes the WAL should contain all the information required for rolling back the LDAP directory to its previous state. After every operation with the WAL, fflush() will be called to be sure the information is written to the disk.
When the administrator starts the XPS, the WAL will be opened. It is possible that the WAL could be not empty. This means that XPS had crashed before some administrative operation was finished. In this case XPS will enter the recovery phase and will automatically roll back the directory using the data from the WAL. A recovery log file will be opened (XPSrecovery.log). (The WAL will be marked Recovery In Process before this starts. If XPS crashes again during the recovery process there is probably a bug in the implementation and the administrator will have to empty the WAL and tidy up the LDAP directory by hand.)
In any case all information about recovery actions taken will be stored in the recovery log file XPSrecovery.log.   It will contain data about the recovery i.e. successful recovery actions and failed recovery

[1] The DN of the entry to be added, plus the DNs of the subordinate X.509 entries to be added.
[2] We will need to take a copy of the entry to be deleted plus all the subordinate X.509 entries.

actions along with the cause of the failure. The purpose of this file is just to inform the administrator about XPS recovery and giving as much information about probable directory corruption as possible.
The format of the messages sent to the XPSrecovery.log can be found in Appendix 1. Further information about the WAL design can be found in [WAL].

Multi-tasking

OpenLDAP is capable of multi-tasking, in which several threads can be updating X.509 attributes simultaneously. Therefore we will use multiple WAL files. Each thread will open its own WAL file using the original entry's DN as the base for the file name. The file name will consist of the "wal" prefix, a 32-character string representing the DN's hash, and ".log" file extention.

Configuration options

XPS configuration options will be held in a separate section of the OpenLDAP main configuration file (slapd.conf). It will be placed between the global configuration directives and the backend definitions.
Note that all XPS configuration key words are case insensitive, as are their values

Note that all the default values have been chosen, so that the modified OpenLDAP code can run as a standard LDAP server using the existing configuration file, and it wont perform XPS functionality

If OpenLDAP is to be used as an XPS then the administrator should define the following option and put yes as the value.

```
EnableXPS		[yes|no]
```

The default value is no.
If EnableXPS is off then all other XPS related options will be ignored.

```
pkcTypes	[userCertificate userCertificate;binary cACertificate cACertificate;binary]
```

pkcTypes holds a space separated list of public key certificate LDAP attribute types that the XPS server should trap in order to parse and split up into their component attributes

```
acTypes			[attributeCertificate Attribute;binary attributeCertificateAttribute]
```

acTypes holds a space separated list of attribute certificate LDAP attribute types that the XPS server should trap in order to parse and split up into their component attributes

```
	crlTypes	[certificateRevocationList;binary authorityRevocationList;binary
			certificateRevocationList authorityRevocationList]
```

crlTypes holds a space separated list of certificate revocation list LDAP attribute types that the XPS server should trap in order to parse and split up into their component attributes

```
DuplicateAttribute	[yes|no]
```

DuplicateAttribute indicates if the certificate or CRL should be stored in the original entry as well as in the child entry. Note that if it is decided to store the original attributes in the entry as well as in the child, then it will increase the database size.
The default value is yes.

```
RevokedCertificateEntries	[yes|no]
```

RevokedCertificateEntries indicates if separate entries should be created subordinate to the CRL entry for each revoked certificate in the CRL. Each of these entries, if created, will hold the serial number of the revoked certificate, its revocation date and any crl entry extensions present in the CRL (such as reason code

and invalidity date). If these entries are not created, then the only information held about the revoked certificates will in the serialNumbers attribute in the CRL entry.
The default value is no.

RevokedRDNformat [x509serialNumber+x509issuer | x509isssuerSerial | x509serialNumber]

The following RDN formats are available for RevokedCertificateEntries, as described in [Chadwick]. Note that this parameter is ignored if RevokedCertificateEntries is No.
The default value is x509serialNumber.

x509serialNumber+x509issuer – multi-valued RDNs will be created from the certificate serial number and issuer fields.
x509serialNumber – the RDN is created from the certificate serial number. This format of the RDN will be suitable for the cases when all certificates | CRLs are issued by one CA.
x509issuerserial – a single valued RDN is created from the certificate serial number and issuer fields.
Administrators should use this format when the LDAP server does not support multi-valued RDNs.

CertRDNformat [x509serialNumber+x509issuer | x509isssuerSerial | x509serialNumber]

This defines the format of the RDNs for attribute certificate and public key certificate entries, as defined in [Gietz] and [Chadwick].
The default value is x509serialNumber+x509issuer.

Walpath [path]

Write Ahead Logs (WAL) are needed for internal purposes, so the administrator may specify a path for the WAL:
The default path if this parameter is missing, is /usr/local/var/xps/wals/

XPSerrorlog [path/filename]

The Administrator may specify a path for the XPS error logs:
The default file is /usr/local/var/xps/error.log

define_attr [ASN.1 attribute] [LDAP attribute]

This specifies the format for the matching between the ASN.1 type references of the fields in the X.509 attributes (as defined in the ASN.1 input file, see [Parsing]) and their equivalent LDAP attribute type names defined in the PKIX schemas [Chadwick], [Gietz], [Sahalayev]. Each line of this file should be in the above format, as described in [Parsing]:

For example:
define_attr certificate.tbsCertificate.signature.algorithm x509signatureAlgorithm

All these definitions will be held in x509attrtypes.txt file (although the administrator can specify his own file or even put the definitions into slapd.conf if he wishes)

Operations

If EnableXPS is NO, then all operations will pass straight through the XPS code and not be affected by it.

If EnableXPS is YES, then the following operations will pass straight through the XPS server and not be affected by it: Bind, Unbind, Compare, ModDN, Abandon, Search. The following administrative operations will be intercepted by XPS and modified as described below: ADD, DELETE, and MODIFY, unless they have a critical control set, in which case they will be rejected with Unsupported critical extension.
.

ADD operation

When XPS receives an ADD request the following actions will be taken. First it will check if the request contains any X.509 attributes. If it does not the request will be processed as normal by OpenLDAP and ignored by XPS. If it does then two lists of entries will be created (in OpenLDAP internal opaque structure) from each X.509 attribute. One list will contain CRL, AC and PKC entries (held in the "xps_e_array" global variable), the other one will contain revokedEntries from CRLs (if they are exist and if the "RevokedCertificateEntries" configuration parameter is set to "yes"). These lists will be created using the xps_create_entries() function (see [Parsing] for a full explanation). If duplicateAttribute is false, the X.509 attributes will be removed from the parent entry.

If parsing of the X.509 attribute values finishes successfully the DN of the parent entry will be added into the WAL using xps_dn2wal(). The parent entry will then be added to the directory using the function (be->be_add)().

If the parent entry is added and there are no errors, XPS will call xps_add_entries(). This function will take the first element from the "xps_e_array" list, add its DN to the WAL using xps_dn2wal() and then add the entry to the directory using (be->be_add)(). The same operations will be performed for each element of the "xps_e_array". After adding all of the entries from the "xps_e_array" the entries from the "xps_revoked" will be added using the same method.

If no error occurs the WAL will be cleaned and the client will receive the LDAP_SUCCESS return code. If there was any error LDAP should be rolled back to its original state. At this point the WAL will contain a list of already added entries (actually as LDIF delete entry commands). Using the LDIF commands from the WAL all the entries will be deleted. After each entry is successfully deleted, its DN (LDIF command) will be removed from the WAL. If any errors occur at this point the LDAP directory will contain entries that are not supposed to be there. Their DNs will be stored in the WAL and in the OpenLDAP log if logging in switched on. It will enable the administrator to remove them manually.

DELETE operation

When XPS receives a DELETE operation it will perform a one level SEARCH operation from the original entry as the base entry, searching for entries of the *x509base* object class [Sahalaev]. Searching will be done with (*be->be_search)( )
Note. If the DELETE was for a non-leaf node near the root, then a full subtree SEARCH could return millions of entries.

In OpenLDAP the SEARCH operation is perfomed by one of the backends. Normally, all the entries that are found are sent straight to the client. We will use the OpenLDAP callback mechanism to change the sending routine to our own routine – xps_sort_search_entries(). The routine will take each returned "Entry" as an input parameter and place it into one of the arrays:

xps_level2 if one of x509AC, x509PKC or x509CRL object classes is present.
xps_level3 if x509CRLentry object classes is present.
xps_level1 if non of the above OCs are present.

Other callback routines have also been modified so that

i) if a referral is returned, or
ii) if an error is returned, or

iii) if other callback routines other than “Entry” are called
then the DELETE will be rejected with UnwillingToPerform.

XPS will check that only Level 2 entries are returned. If there are some Level 3 entries this means that the x509 entries are messed up and the DELETE operation will be rejected with UnwillingToPerform. If there are no entries returned, then the DELETE was a non-XPS DELETE and XPS will pass the operation to OpenLDAP for normal processing.

If there are one or more Level 2 entries returned, then XPS will check the object classes of the returned entries, and for each x509CRL object class found, it will perform a 1-level SEARCH with the CRL entry as base, searching for object class *present*. XPS will check that only level 3 entries are returned by these 1-level Searches. If anything other than Level 3 entries are returned the DELETE will be rejected with UnwillingToPerform.

XPS has now collected all the XPS generated entries beneath the entry to be deleted. XPS will now perform a base object SEARCH, searching for object class *present* (i.e. read the entry to be deleted). XPS will check that the entry has been inserted into the Level 1 array by the callback routine xps_sort_search_entries(), otherwise the DELETE will be rejected with UnwillingToPerform.

XPS will now call xps_delete_entries(). This function will take the first entry from the xps_level3 array, if it's not empty, and will add the whole entry into the WAL using xps_entry2wal(), then delete it from the directory using (be->be_delete)(). The same will be done with all the entries from the xps_level3 array. After all entries from the xps_level3 array are deleted, xps_delete_entries() will do the same with the xps_level2 and xps_level1 arrays (i.e. add to WAL and delete from the directory).

If no error occurred the WAL will be cleaned and the client will receive the LDAP_SUCCESS result code. If there were any errors LDAP should be rolled back to it original state. All deleted entries are stored in the WAL. They will be added back to the directory using (*be->be_add)( ). When an entry is added to the directory it will be removed from the WAL. If any errors occur at this point some entries will still be missing from the directory. All of the failures will be stored in the XPS/recovery.log. The administrator will have to add the missing entries manually.

MODIFY operation

If XPS receives an MODIFY request it will first check if there are any X.509 attributes in the modifications. If there are none then the request will be processed as normal by OpenLDAP.
Otherwise, depending on the operation type (mod_op), the following actions will be taken for every X.509 attribute type found in the MODIFY operation.

Operation type is add.

In this case the sequence of actions is identical to the LDAP ADD operation. XPS will create 2 lists of entries to be added (xps_e_array and xps_revoked) from the values provided with the MODIFY operation. It will be done using the xps_create_entries() routine.

At this point in time XPS has a list of all the child and grandchild entries that need to be added to the directory. XPS will then move to the next modification operation.

Operation type is delete.

If no attribute values are provided (i.e. Delete the attribute), XPS will perform a one level SEARCH from the entry to be modified, searching for the attribute type to be deleted. The search for the attribute values will be done with (*be->be_search)(). The callback function xps_sort_search_entries() will put all the entries that are found into the xps_level2 array. If any entries are inserted into the level 1 or level 3 arrays then the MODIFY will be rejected with UnwillingToPerform.

If an attribute value is provided (i.e. Delete an attribute value), XPS will perform a one level SEARCH for this attribute value to check that it exists, by performing a certificateExactMatch for ACs and PKCs and a certificateListExactMatch for CRLs. If it does exist, then the callback function will put the returned child entry into the xps_level2 array. (If it is put into the level 1 or level 3 arrays then the MODIFY will be rejected with UnwillingToPerform.

NOTE that the xps_level2 array should have just one more entry in it after the SEARCH was performed than before the SEARCH was performed. (It cannot assume that there were zero entries in there at the start.)

If the attribute value does not exist, the user will be sent the error message noSuchAttribute.

If the attribute type of the value(s) to be deleted is one of the configured crlTypes, a further set of one level searches will be performed from the entry or entries returned from the first one level Search. XPS will perform a 1-level SEARCH with the CRL entry as base, searching for object class *present*. The callback function xps_sort_search_entries() should put all entries found into the xps_level3 array. XPS will check that only level 3 entries are returned by these 1-level Searches. If anything other than Level 3 entries are returned the MODIFY will be rejected with UnwillingToPerform.

At this point in time we have a list of all the child and grandchild entries that need to be deleted. XPS will then move to the next modification operation.

Operation type is replace.

In this case XPS will act as it were two separate operations. The sequence of actions is the same as in the preceding paragraphs for delete the attribute (with no attribute values provided), and add new attribute value(s).

Finalising the Modify Operation

After going through all of the modifications in the MODIFY operation there will be 2 arrays of entries to be deleted (xps_level2, xps_level3) and two arrays of entries to be added (xps_e_array and xps_revoked). XPS will add these entries to the WAL then add or remove them from the directory, via calls to xps_add_entries() and xps_delete_entries(). Note that whilst the new entries need to be added and old ones deleted, the WAL must not be emptied yet.

If both sets of adds and deletes are finished successfully, the remaining attributes in the MODIFY operation need to be modified in the parent entry, and duplicate attributes may also need adding or removing from the parent entry. If duplicateAttribute = False, the X.509 attributes are removed from the original MODIFY operation.

The MODIFY operation is then passed to OpenLDAP (unless it is empty i.e. it only contained X.509 attributes and duplicateAttribute = False). Note that the MODIFY operation is atomic, therefore it will either all work, or all fail. For this reason we do not need to add it to the WAL.

If the Modify operation succeeds, the WAL will be cleaned and the user will receive an LDAP_SUCCESS return code as the result of his MODIFY operation.

If the MODIFY operation fails and duplicateAttribute = True, remove the X.509 attributes from the original MODIFY operation and try again. (This is because the administrator could have altered the value of the duplicateAttribute parameter to True after the entry was created with duplicateAttribute = False). If the Modify operation still fails, then XPS will have to go into rollback mode using the WAL. XPS must delete all the added child (and grandchild) entries, and re-add back all the deleted child (and grandchild) entries.

NOTE. There will be a problem is the administrator changes the value of duplicateCertificates after adding some X.509 attributes but before running the Modify operation. If the initial value was False and the new value is True, then if the Modify attempts to Delete the attribute it will fail because XPS will try to delete the duplicate attribute (which does not exist). It is for this reason that we remove the duplicate attributes and try again. If the initial value was True and the new value is False, then the Modify operation will delete the child entry, but the duplicate attribute will still remain in the parent entry until the administrator removes it by hand.

Appendix A. XPSrecovery.log Format

When XPS finds a DN in the WAL the following message will be logged.

```
Undeleted entry found:
dn: [DN of the entry]
. . . removed. / . . . unable to remove
```

It will say ". . . removed" if the entry is successfully deleted, or failed to delete the entry because it did not exist. If the entry exists but cannot be deleted the ". . . unable to remove" message will be logged.

When XPS finds an undeleted Attribute in the WAL the following message will be logged.

```
Undeleted attribute value found:
dn: [DN of the entry]
[attribute type]: [attribute value]
. . . removed. / . . . unable to remove
```

It will say “. . . removed” if the entry is successfully deleted, or failed to delete the entry because it did not exist. If the entry exists but cannot be deleted the “ . . . unable to remove” message will be logged.

When XPS finds an Entry in the WAL the following message will be logged.

```
Unrestored entry found:
dn: [DN of the entry]
[attribute type]: [attribute value]
. . . . . . . . . . . .
[attribute type]: [attribute value]
. . . restored/. . . unable to restore
```

It will say “. . . restored” if the entry is successfully restored, or failed to restore the entry because it hadn't been deleted. If the entry does not exist but cannot be added to the Directory the “ . . . unable to restore” message will be logged.

When XPS finds an unrestored Attribute in the WAL the following message will be logged.

```
Unrestored attribute value found:
dn: [DN of the entry]
[attribute type]: attribute value
. . . restored/. . . unable to restore
```

it will say “. . . restored” if the Attribute is successfully restored, or failed to restore the Attribute because it hadn't been deleted. If the Attribute does not exist but cannot be added the “ . . . unable to restore” message will be logged.

Appendix B. List of Functions

XPS Functions

xps_create_entries() - this function creates 2 lists (xps_e_array and xps_revoked) of entries from provided x.509 attibute values.

xps_delete_entries() - this function deletes entries held in the global arrays xps_level3, xps_level2 and xps_level1 from the Directory.

xps_add_entries() - this function adds entries held in the global arrays xps_e_array and xps_revoked to the Directory. This function first adds the DN of the entry-to-be-added to the WAL using xps_dn2wal() and then adds the entry to the directory using (be->be_add)(). The same operations will be performed for each element of the “xps_e_array”. After adding all of the entries from the “xps_e_array” the entries from the “xps_revoked” array will be added using the same methods.

OpenLDAP internal functions for updating the backend database

(*be->be_add)( ) – adds an entry to the backend directory
(*be->be_delete)( ) – deletes an entry from the backend directory
(*be->be_modify)( ) – modifies an entry from the backend directory

These are internal OpenLDAP functions for adding, deleting and modifying entries and attributes. As OpenLDAP has a number of different backends each of them has its own function for the operations. For example, when the backend is defined the add function can be called using the pointer be->be_add. The same is true for deleting and modifying.

WAL Functions

xps_dn2wal() - this function will save the DN of an added entry in the WAL. It is used for rolling back added entries, if necessary. The WAL will contain an LDIF command to delete the named entry.
xps_entry2wal() - this function will save an entry-to-be-deleted in the WAL. It is used for rolling back deleted entries. The WAL will contain an LDIF command to add the entry to the DIT.

References

[API] M. Smith, A. Herron, M. Wahl, A. Anantha <draft-ietf-ldapext-ldap-c-api-xx.txt>
[Auth] Wahl, M., Alverstrand, H., Hodges, J., Morgan, R. "Authentication Methods for LDAP", RFC 2829, May 2000
[Chadwick] D.W.Chadwick, M.V.Sahalayev. “Internet X.509 Public Key Infrastructure - LDAP Schema for X.509 CRLs”. <draft-ietf-pkix-ldap-crl-schema-00.txt>, Feb 2003
[Gietz] N. Klasen, P. dsaGietz. "An LDAPv3 Schema for X.509 Certificates"<draft-klasen-ldap-x509certificate-schema-00.txt>, Feb 2002.
[Sahalayev] D.W.Chadwick, M.V.Sahalayev. “Internet X.509 Public Key Infrastructure - LDAP Schema for X.509 Attribute Certificates”. <draft-ietf-pkix-ldap-ac-schema-00.txt>, Feb 2003
[WAL] Mikhail Sahalayev, David Chadwick “Write Ahead Log (WAL) Design”, Feb 2003
[X509] ISO/ITU-T Rec. X.509(2000) The Directory: Authentication Framework
[Parsing] Mikhail Sahalayev, Ed Ball, David Chadwick “Parsing X.509 Attributes”, Feb 2003
[LDIF] G. Good “The LDAP Data Interchange Format (LDIF) - Technical Specification”, RFC 2829, June 2000

Appendix 9. Write Ahead Log (WAL) Design

M. Sahalayev, D. W. Chadwick

Release Number	***Release Date***	***Comments***
Version 1.0	3 February 2003	Initial release for public comment

The purpose of the WAL is to store information about LDAP DNs or entries while performing an LDAP operation such as ADD, DELETE or MODIFY. This will allow the X.509 Parsing Server (XPS) to restore the initial state of the database in case of a failure.

When a user initiates an operation, the WAL file will be opened. It will stay opened until the operation is finished and it's no longer required,and then this file will be deleted.

The content of the WAL file is either the DN of an entry (for ADD requests) or a whole entry (for DELETE requests). This information will be stored in LDIF format and multiple entries will be separated by blank lines. All LDIF operations will be performed using the existing OpenLDAP mechanisms. To be sure that information is actually written on the disk after every operation with the WAL system function fflush() will be called.

Example:

If a user sends an ADD request which contains the entry with 2 X.509 certificates, then XPS will create the entry and 2 additional X.509 entries. After storing the initial entry's DN in the WAL this entry will be added to the Directory. Then the first X.509 entry's DN will be stored in the WAL and the entry stored in the Directory. The same will be done for the second X.509 entry. At the end the WAL file will be similar to the following, assuming that the serial numbers for the two X.509 certificates are 1111111 and 222222, and the DN of the CA is "EMAILADDRESS=certify@pca.dfn.de, CN=DFN Toplevel Certification Authority, OU=DFN-PCA, OU=DFN-CERT GmbH, O=Deutsches Forschungsnetz,C=DE".

```
dn: =Roman Gebhart,ou=Payroll, o=Salford,c=gb
<CR LF>
dn: x509serialNumber=1111111, "EMAILADDRESS=certify@pca.dfn.de,CN=DFN
    Toplevel Certification Authority,OU=DFN-PCA,OU=DFN-CERT GmbH,O=Deutsches
    Forschungsnetz,C=DE", cn=Roman Gebhart,ou=Payroll, o=Salford,c=gb
<CR LF>
dn: x509serialNumber=2222222, "EMAILADDRESS=certify@pca.dfn.de,CN=DFN
    Toplevel Certification Authority,OU=DFN-PCA,OU=DFN-CERT GmbH,O=Deutsches
    Forschungsnetz,C=DE", cn=Roman Gebhart,ou=Payroll, o=Salford,c=gb
```

If all operations were successful the WAL file will be deleted. Otherwise the first blank line from the end of file will be found. If the WAL contains the DN of an entry, this entry will be deleted from the directory. If it's a whole entry, it will be added back. Then this piece of information will be deleted from the WAL.

In the same way all the WAL file will be processed until it is empty. When all recovery operations are finished, the WAL file will be deleted. For more information see [detailed design]. Note that when OpenLDAP/XPS first initialises, it must search for any WAL files left from the previous invocations, and perform the recovery operations such as those above before the XPS is ready to receive new requests.

The OpenLDAP server is multitasking and it is capable of handling multiple requests at once. So XPS may be running in different threads simultaneously. To avoid collisions different WALs will be required for each thread. Since the format of the WAL is a text file, each operation will create its own file. In order to avoid name collisions each file name will be constructed using the following information:

“wal” – the same part for every WAL file, distinguishing it from other OpenLDAP and XPS files.

32 ASCII symbols representing a 128-bit hash of the DN of the entry to be modified. This will avoid collisions and also will keep the XPS from modifying the same entry concurrently, since only one process can work with the particular WAL file. The hashing will be done using OpenLDAP lutil_MD5Init(), lutil_MD5Update(), and lutil_MD5Final() functions. For the converting 128-bit hash into ASCII string the function int hash2string(char *hash, char *dn) will be created. The input parameters are
dn - input string, containing the DN of the entry.
hash - 32-character string, representing the created hash. It will be 32 hexadecimal numbers (0-9, a-f)

hash2string () will return 0 if hashing was successful, or 1 - if not. .log – file extension, serves for information only purposes.

Example:

If there are three operations beingperformed by OpenLDAP/XPS at the present time there will be three log files in the directory, defined in slapd.conf. If walpath is set to /usr/local/var/xps/wals/ the result of *ls /usr/local/var/xps/wals/* will be similar to the following:

```
walKHWIHfdsafJG420ghdlT4YG.log
walerq1DRF2gagLVPE8efabNVT.log
wal1tr12LH9KhjtIO96lqR1MxQv.log
```

A collision may appear when some of the old WAL files are left from the previous running of the OpenLDAP/XPS server (for example if the system crashed due to some reason). In this case during the OpenLDAP startup XPS will try to restore the information from every WAL file (described above) and, regardless of the result, delete the WAL file (if the restoration wasn’t successful if will be logged in xpsrecovery.log).

Appendix 10. Parsing X.509 Attributes

E.Ball, M.Sahalayev

Version	Date	Comment
1.0	28 February 2003	Initial release
1.1	13 May 2003	Update of first release with change to x509attrfile.txt

When the XPS finds x.509 attributes in the entry to be added it calls one of the x509***_2_mods() functions.

There are three such functions:

x509AC_2_mods() - converts Attribute Certificate into the list of Modifications
x509CRL_2_mods() - converts Certificate Revocation List into the list of Modifications
x509PKC_2_mods() - converts Public Key Certificate into the list of Modifications

All these functions are generated by an ASN.1 compiler, written by E.Ball.

ASN1 compiler description

The compiler is based on a sub-set of the ASN.1 defined by X.680 (12/97) with some restrictions and additions. The additions are support for the ANY DEFINED BY type, which has been included for backwards support, and also a new EXTENSION type. This is for support of the extensions mechanism used in X509 certificates, whereby a value is passed in binary as OCTET STRING but actually also has an ASN.1 definition for the contents of the octet string.

The compiler is built using the compiler generator tools *flex* and *bison*. Flex is based on the well known UNIX lex and builds a lexical analyzer which is used to process the text input to the complier. The lexical analyzer recognize keywords in the ASN1, identifiers, numbers etc. and removes white space in the traditional way. The definition used by flex to build the lexical analyzer is contained in the file asn1.lex (see Appendix 1). The output from the lexical analyzer is actually a sequence of tokens and corresponding values that are used by the compiler itself. The compiler is built using the tool bison, a compiler generator based on the well known UNIX tool yacc. The definition of the compiler is contained in the file asn1.y (see Appendix 1), and consists of Bachus-Naur type definitions that closely follow the ASN1 definition described above.

Although bison is extremely useful in automatically building a parser that will recognize input in its definition file, it does not automatically generate parse trees and output code. These must be added to the definition file, in the form of embedded actions, to build a parse tree, and also a code generator. After the input has been scanned and a parse tree built, the code generator walks this tree and translates this into output code.

The Parse Tree

The C structures used by the compiler are contained in a file called common.h (see Appendix 1).

The most important are the Definition and the Type, corresponding to the equivalent ASN.1. The compiler maintains an internal stack that holds Type values and as each Type is recognized it is put onto the stack. Where Types may be combined together, as defined by the ASN1, they are pulled from the stack and a new Type put back on the stack. Eventually, assuming that the input is good, an ASN.1 definition will be recognized and a Definition structure will be created and put on a linked list. The Definition will contain a linked list of Type structures that are part of the Definition.

Generating the Code

Starting with the first Definition in the parse tree it is analyzed and the text of a C subroutine written to the output file. This has the name decodeABCD, where ABCD is replaced by the name of the Definition. For each Type contained in the typelist of the Definition a C TypeDescription structure is defined in the written routine. These are combined together into an array and passed to a more basic decode function.

For example:-

```
ABCD  ::= SEQUENCE { x INTEGER, y BIT STRING }
```

would produce:-

```
void decodeABCD(char * name,int tag,int tagOption,byte **P,int
count,TypeDescription TD[],int optional,int Default,char *ObjectIdentifier)
{int t;
TypeDescription _TD0={"x",-1,2,0,(_CPF)decodeINTEGER,0,0,""};
TypeDescription _TD1={"y",-1,2,0,(_CPF)decodeINTEGER,0,0,""};
TypeDescription _TD[]={_TD0,_TD1};
printf("%s ",name);
decodeSEQUENCE("ABCD",tag,tagOption,P,2,_TD,optional,Default,"");
}
```

The Code Library

A skeleton library has been written called decodelib.c. This provides the basic support for all the ASN.1 built in types, and it takes the user to the value part of each TLV ber structure in the input. The user must decide what it is that he wants to do with this value, e.g put it into his own structure, throw it away or whatever.

The library also copes with ASN.1 OPTIONAL and DEFAULT situations, although it is left up to the user to supply any missing DEFAULT values.

The library implements basic error checking, i.e. that tags are of the required type and that lengths of sequences are not exceeded.

Using the Compiler

The compiler is called decoder and it takes two command line arguments, the first being the name of the file containing the ASN1 definition, the second being the file name into which the generated C will be written. At the moment the compiler first copies the file decoderlib.c to this file and then writes the routines that it generates. It could be easily modified to miss out copying the library and this could then be simply linked in with the user's application.

e.g. decoder q.asn1 q.c

This will compile the definition in q.asn1 and write the file q.c

ASN.1 Additions

In order to support the EXTENSION mechanism the ASN.1 TypeAssignment has been extended from:-

```
TypeAssignment ::= typereference "::=" Type
```

to:-

TypeAssignment ::= typereference "::=" Type | OID typereference "::=" Type

Where OID is an object identifier production in dotted integer notation e.g 1.2.3.4 This allows an OID to be associated with an ASN1 type.

The Type ExtensionType is defined:-

ExtensionType ::= "EXTENSION" lcID, where lcID is a lowercase prefixed identifier.

These two modifications combine as follows:-

```
X509 DEFINITIONS ::=
BEGIN
Test::= SEQUENCE{
      abc OBJECT IDENTIFIER,
  e EXTENSION abc}
0.1.2.3.4 A::= INTEGER
END
```

Here EXTENSION abc means that e will be encoded as an OCTET STRING but that its internal structure will be defined by the OID contained in the value abc. At run time the OID is extracted from abc and stored. When the decodeEXTENSION code is reached this OID is looked up from abc and compared with those of known definitions. In this example if it is 0.1.2.3.4 then the value of e will be decoded as an INTEGER. If the OID is not known an error will be generated.

Generating the parsing functions

To generate these functions the ASN.1 definitions of CRL, PKC and AC have been written (files pkc.asn1, ac.asn1, crl.asn1 - see Appendix 1)
The following commands

```
decoder pkc.asn1 pkc.c
decoder ac.asn1 ac.c
decoder crl.asn1 crl.c
```

produce C files that contain the required functions along with necessary library functions.

Since the functions, created be the ASN.1 compiler use the strings from ASN.1 type references as attribute types, they will need to be replaced with the actual LDAP attribute types, taken from the LDAP PKI schema IDs [CRL], [AC], [PKC]. For example the ASN.1 type serialNumber will need to be replaced with the x509serialNumber LDAP attribute type. This is achieved as follows: a configuration file x509attrtypes.txt comprises two columns of strings. The first column holds the ASN.1 type reference, the second holds the corresponding LDAP attribute type name. An example x509attrtypes.txt file is shown in Appendix 1. If the LDAP schema definitions change, or new ones are defined, then the file x509attrtypes.txt can be updated, the new LDAP schemas can be added, OpenLDAP can be restarted and the new schema definitions will take effect. If however, a new X.509 extension is defined, this can only be supported by: adding the new X.509 extension to the appropriate asn1 file (pkc, ac or crl), recompiling it using the ASN.1 compiler, updating the file x509attrtypes.txt and updating the OpenLDAP schema.

Parsing the extensions

If a CRL, PKC or AC has any extensions they will be parsed as and stored in the memory as an "Extension " attribute. The XPS supports a number of the selected attributes, so their values have to be decoded as well. For this purpose the XPS will have a list of OIDs of the supported attributes (held in file x509attrtypes.txt) and a corresponding parsing function from the library, described above. For example, the Subject Altenative Name extension for the PKC has the following ASN.1 definition:

```
 SubjectAltName ::= GeneralNames

    GeneralNames ::= SEQUENCE SIZE (1..MAX) OF GeneralName

    GeneralName ::= CHOICE {
        otherName                   [0]     OtherName,
        rfc822Name                  [1]     IA5String,
        dNSName                     [2]     IA5String,
        x400Address                 [3]     ORAddress,
        directoryName               [4]     Name,
        ediPartyName                [5]     EDIPartyName,
        uniformResourceIdentifier   [6]     IA5String,
        iPAddress                   [7]     OCTET STRING,
        registeredID                [8]     OBJECT IDENTIFIER}
```

If the OID of this extension is found while parsing the certificate the following function will be called:

void decodeSEQUENCE(char *name,int tag,int implicit,byte **P,int count, TypeDescription TD[],int optional,int Default,)

The attribute types will be replaced with the corresponding ones from the schema [CRL], [AC], [PKC].

References

[AC] Chadwick, D.W., Sahalayev, M. V. "Internet X.509 Public Key Infrastructure LDAP Schema for X.509 Attribute Certificates", <draft-ietf-sahalayev-pkix-ldap-ac-schema-00.txt>, February 2003

[CRL] Chadwick, D.W., Sahalayev, M. V. "Internet X.509 Public Key Infrastructure LDAP Schema for X.509 CRLs", <draft-ietf-sahalayev-pkix-ldap-crl-schema-00.txt>, February 2003

[PKC] Klasen, N., Gietz, P. "An LDAPv3 Schema for X.509 Certificates",<draft-klasen-ldap-x509certificate-schema-00.txt>, February, 2002

Appendix 1 Example input file structures

Example asn1.lex file

```
%{
```

```
#include "asn1.tab.h"
#include "printtoken.h"
//SCOPEVAL "("[0-9a-zA-Z]+".."[0-9a-zA-Z]+")"
//{SCOPEVAL} {printf("%s ",yytext);strcpy(yylval.str,yytext); return SCOPE;}
%}

ID   [A-Z][A-Za-z0-9]*
lcID [a-z][a-zA-Z0-9]*
TAG  "["[0-9]*"]"
NUMBER [0-9]+


%%

"IA5String" {printf("%s ",yytext); strcpy(yylval.str,yytext);return IA5STRING;}
"TeletexString" {printf("%s ",yytext);
            strcpy(yylval.str,yytext);
                  return TELETEXSTRING;
                  }
"PrintableString" {printf("%s ",yytext);
            strcpy(yylval.str,yytext);
                  return PRINTABLESTRING;
                  }
"BMPString" {printf("%s ",yytext);
            strcpy(yylval.str,yytext);
                  return BMPSTRING;
                  }
"UniversalString" {printf("%s ",yytext);
            strcpy(yylval.str,yytext);
                  return UNIVERSALSTRING;
                  }
".." {printf(".. dotdot");strcpy(yylval.str,yytext); return DOTDOT;}


EXTENSION {printf("EXTENSION ");yylval.val=EXTENSION;
        return EXTENSION;}

GeneralizedTime {printf("GeneralizedTime ");
            yylval.val=GeneralizedTime;
            return GeneralizedTime;}

UTCTime {printf("UTCTime ");yylval.val=UTCTime; return UTCTime;}

"("  {printf("( "); strcpy(yylval.str,yytext); return OPENBR;}
")" {printf(") ");strcpy(yylval.str,yytext);return CLOSEBR;}



MIN {printf("%s ",yytext); strcpy(yylval.str,yytext); return MIN;}
MAX {printf("%s ",yytext); strcpy(yylval.str,yytext); return MAX;}
```

```
NULL {printf("%s ",yytext);yylval.val=_NULL; return _NULL;}

REAL {printf("%s ",yytext);yylval.val=REAL; return REAL;}

SIZE {printf("SIZE ");yylval.val=SIZE;return SIZE; }
OF {printf("OF ");yylval.val=OF; return (OF);}
ANY    {printf("ANY "); yylval.val=ANY; return(ANY);}
DEFINED {printf("DEFINED "); yylval.val=DEFINED; return(DEFINED);}
BY      {printf("BY "); yylval.val=BY; return(BY);}

DEFAULT {printf("DEFAULT ");yylval.val=DEFAULT;return DEFAULT; }
IMPLICIT {printf("IMPLICIT ");yylval.val=IMPLICIT;return IMPLICIT;} /* consume this
keyword */
OPTIONAL {printf("OPTIONAL ");yylval.val=OPTIONAL;return OPTIONAL;}
EXPLICIT {printf("EXPLICIT ");} /* consume */
TRUE   {printf("TRUE ");yylval.val=TRUE;return TRUE;}
FALSE  {printf("**FALSE ");yylval.val=FALSE; return FALSE;}

{TAG} {int i,sum=0;
 char *pnt=yytext;// convert the tag digits to an int
       for(i=1;i<strlen(yytext)-1;i++)
               sum=sum*10+pnt[i]-'0';

       printf("%s ",yytext);yylval.val=sum; return(TAG);
       }
{NUMBER} {int i,sum=0; // (0)
        char *pnt=yytext;// convert the value digits to an int
//          printf("\n**NUMBER yytext=%s\n",yytext);
       for(i=0;i<strlen(yytext);i++)
               sum=sum*10+pnt[i]-'0';

        printf("%d",sum);yylval.val=sum; return(NUMBER);
       }

"," {printf(", ");return (COMMA);}
"{" {printf("\n{ ");yylval.val=_br;return(_br);}
"}" {printf("}\n");yylval.val= br_;return (br_);}

::= {printf("::= "); return(DFB);}

{lcID} {printf("%s ",yytext);
        strcpy(yylval.str,yytext);
         return(lcID);
        }

BOOLEAN {printf("%s ",yytext);yylval.val=BOOLEAN; return BOOLEAN;}

OCTET  {printf("OCTET ");
        yylval.val=OCTET;
            return (OCTET);}
```

```
BIT {printf("BIT ");yylval.val=BIT;return BIT;}


STRING {printf("STRING ");
        yylval.val=STRING; return STRING;}

OBJECT  {printf("%s ",yytext);
                yylval.val=OBJECT;
                        return(OBJECT);}
IDENTIFIER  {printf("%s ",yytext);
                yylval.val=IDENTIFIER;
                        return(IDENTIFIER);}

DEFINITIONS {printf("DEFINITIONS ");return(DEFINITIONS);}

BEGIN  {printf("BEGIN ");return(_BEGIN);}

INTEGER {printf("INTEGER "); yylval.val=INTEGER; return (INTEGER); }

END {printf("END "); return(_END);}

ENUMERATED {printf("ENUMERATED ");yylval.val=ENUMERATED; return
ENUMERATED;}


"..." {printf("... ");yylval.val=DOTDOTDOT; return(DOTDOTDOT);}


CHOICE {printf("CHOICE ");
 yylval.val=CHOICE;
             return(CHOICE);}

SEQUENCE {printf("SEQUENCE ");
          yylval.val=SEQUENCE;
             return(SEQUENCE);}

SET {printf("SET ");
     strcpy(yylval.str,yytext);
      return(SET);}

{ID} {printf("%s ",yytext);
      strcpy(yylval.str,yytext);
      return(ID);}

"--{" {printf("%s ",yytext); yylval.val=LEFTMATCH;return LEFTMATCH;}
"}--" {printf("%s ",yytext); yylval.val=RIGHTMATCH; return RIGHTMATCH;}

"--"  {register int c;
       while ( (c = input()) != '\n' &&  c != EOF ) ;
      } /* consume comments */
"."  {printf("%s",yytext); return DOT;}
```

```
[\n]+ {printf("\n");}
[ \t]+    /* whitespace */
```

Example asn1.y file

```
/* special purpose asn1 grammar  for component matching project*/

%{
#include <stdio.h>
#include "type.h"
#include "asn1type.h"
#include "definition.h"
#include "stack.h"
#include "oid.h"

#define TERMINAL 0  // used to denote a terminal token
     Stack ST;
    int i;
    int paircount=0;
    Definition *top=0;
    char buf[100];
    Type AnyType={"",{"ANY",ANY},0,{0,-1}};
    Type AnyDefinedByType={"",{"ANY DEFINED BY",ANY_DEFINED_BY},0,{0,-1}};
    Type BitStringType={"",{"BIT STRING",BIT_STRING},0,{0,-1}};
    Type RealType={"",{"REAL",REAL},0,{0,-1}};
    Type ObjectIdentifierType={"",{"OBJECT IDENTIFIER",OBJECT_IDENTIFIER},0,{0,-1}};
    Type ChoiceType={"",{"CHOICE",CHOICE},0,{0,-1}};
    Type NullType={"",{"NULL",_NULL},0,{0,-1}};
    Type IdType={"",{"",0},0,{0,-1}};

    Type EnumeratedType={"",{"ENUMERATED",ENUMERATED},0,{0,-1}};
    Type IntegerType={"",{"INTEGER",INTEGER},0,{0,-1}};

    Type OctetStringType={"",{"OCTET STRING",OCTET_STRING},0,{0,-1},0,0,0};
    Type ExtensionType={"",{"EXTENSION",EXTENSION},0,{0,-1},0,0,0};

    Type SequenceOfType={"",{"SEQUENCE OF",SEQUENCE_OF},0,{0,-1}};
    Type SetType={"",{"SET",SET},0,{0,-1}};
    Type BooleanType={"",{"BOOLEAN",BOOLEAN},0,{0,-1}};
    Type SequenceType={"",{"SEQUENCE",SEQUENCE},0,{0,-1}};

    Type SetOfType={"",{"SET OF",SET_OF},0,{0,-1}};
    Type UTCTimeType={"",{"UTCTime",UTCTime},0,{0,-1}};
    Type IA5StringType={"",{"IA5String",IA5STRING},0,{0,-1}};
    Type TeletexStringType={"",{"TeletexString",TELETEXSTRING},0,{0,-1}};
    Type PrintableStringType={"",{"PrintableString",PRINTABLESTRING},0,{0,-1}};
    Type BMPStringType={"",{"BMPString",BMPSTRING},0,{0,-1}};
    Type UniversalStringType={"",{"UniversalString",UNIVERSALSTRING},0,{0,-1}};

    Type GeneralizedTimeType={"",{"GeneralisedTime",GeneralizedTime},0,{0,-1}};
```

```
   Type *TypePnt;
//    ASN1Type * apnt;
   char buf[100]; // for temp storage of lcIDs
   ASN1Type asn1Syntax={"Syntax",0};
   ASN1Type asn1OctetString={"OCTET STRING",OCTET_STRING};
%}

%union {int val; char str[100];}
%token <str> TRUE FALSE _NULL OCTET STRING
%token <val> VALUE  VALUE_OF
%token  <str>ANY DEFINED BY ANY_DEFINED_BY
%token <str> DEFAULT IMPLICIT
%token <str> DFB  _BEGIN _END DEFINITIONS  IDENTIFIER
%token <str> CHOICE
%token <str>IMPLICIT OPTIONAL
%token <str> ModuleDefinition   SET SET_OF SEQUENCE_OF
%token <str> COMMA
%token <val> TAG MINUS NUMBER
%token <str> lcID UTCTime GeneralizedTime
%token <val> INTEGER OCTET_STRING BIT_STRING BOOLEAN SEQUENCE OF
%token <str> ID OPENBR CLOSEBR
%token <str> _br br_ MIN MAX DOTDOT
%token <str> SIZE SCOPE OBJECT IDENTIFIER OBJECT_IDENTIFIER
%token <str> REAL BIT ENUMERATED DOTDOTDOT DOT
%token <val> NAMED_NUMBER
%token <str> LEFTMATCH, RIGHTMATCH
//%token <str> OIDVAL
%token <val> EXTENSION
%token <str> IA5STRING, TELETEXSTRING, PRINTABLESTRING,BMPSTRING,
UNIVERSALSTRING

%type <str> typereference TaggedType  DefinedType
%type <str>  ComponentTypeList ComponentType
%type <str>  ReferencedType ChoiceType ChoiceThing
%type <str> AlternativeTypeList AlternativeType
%type <str>  NullType OctetStringType SetType SetOfType SetThing
%type <str> ObjectIdentifierType RealType BitStringType EnumeratedType
%type <str> RootEnumeration Enumeration Enumerations AdditionalEnumeration
%type <str> XXX EnumerationItem
%type <val> NamedNumber SignedNumber
%type <val> IntegerType BooleanType SequenceType SequenceOfType SequenceThing
%type <val> BuiltInType  Type
%type <str> lcID NamedType
%type <str> SizeConstraint UTCTimeType
//%type <str> SyntaxDefinition
%type <str> Oid

%%

exp : ID DEFINITIONS DFB {resetStack(&ST);}  _BEGIN ModuleBody {} _END
        {//printf("\nASN1 parsed ok\n");
```

```
            }         ;

Oid : NUMBER {Type T=IdType; // e.g 0.1.2.3.4
            sprintf(T.ObjectIdentifier,"%d",$1);
               pushType(&ST,&T);
              } DOT OidList
              {int i;
                 Type A,T;
                 pullType(&ST,&A);
                    pullType(&ST,&T);
                    strcat(T.ObjectIdentifier,".");
                    strcat(T.ObjectIdentifier,A.ObjectIdentifier);
                    pushType(&ST,&T);

              };
OidList: NUMBER {Type T=IdType;
              sprintf(T.ObjectIdentifier,"%d",$1);
                    pushType(&ST,&T);
                    }
        | Oid {};

ModuleBody : AssignmentList | /* empty */ {};

AssignmentList : Assignment | AssignmentList Assignment {} ;

Assignment: TypeAssignment {};

// | ValueAssignment ;

/*
SyntaxDefinition: LEFTMATCH SyntaxList RIGHTMATCH
      {Type T;

         //printf("***syntax definition ");
//       printStack(&ST);
         }

         ;


SyntaxList: ID {Type T,*new;
         strcpy(T.name,$1);
            pushType(&ST,&T);
            //printStack(&ST);
         }
            | ID COMMA SyntaxList
            {//There is a type on the stack containing syntaxlist
            Type SL,*new;
            pullType(&ST,&SL);//get it
            new=newType($1,&asn1Syntax,0);
            SL.next=new;
```

```
                pushType(&ST,&SL);

                } ;
*/

TypeAssignment : typereference {}DFB Type

{    Type A;Type T;int i;
     Definition *D;

printStack(&ST);
     pullType(&ST,&A);// the Type e.g. SEQUENCE
     pullType(&ST,&T);// the typereference
//   printf("***Typeassignment tag=%d %d\n",T.tag.value,A.tag.value);
     addDefinition(top,&T);
     D=currentDefinition(top);
     D->aType=A.aType; // e.g. ("SEQUENCE" SEQUENCE) or ("AAA" 0)
     D->tag.value=A.tag.value;
     D->tag.flag=A.tag.flag;

     strcpy(D->ObjectIdentifier,T.ObjectIdentifier);

//     D->SyntaxDefinition= A.SyntaxDefinition;
      D->typelist=A.next;
     D->next=0;


     }
     ;


typereference : ID  {Type T=IdType;
                    // printf("\n**ID= %s\n",$1);
                    strcpy(T.aType.name,$1);
                     pushType(&ST,&T);
                    }
              | Oid ID
                       {Type T=IdType, A;
                 //   printf("\n**OID= %s ID= %s\n",$1,$2);
                          pullType(&ST,&A); //the oid
                           strcpy(T.ObjectIdentifier,A.ObjectIdentifier);
                     strcpy(T.aType.name,$2);
                     pushType(&ST,&T);
                       //     sprintType(&T);
                    }                ;


Type :  BuiltInType { } | ReferencedType {} | ConstrainedType {};


ConstrainedType: TypeWithConstraint {} ;
```

```
TypeWithConstraint: SequenceSizeConstraintOf {} ;

SequenceSizeConstraintOf: SEQUENCE SizeConstraint OF Type
            {Type T1=SequenceOfType;
                  Type T2,*new;

                  pullType(&ST,&T2);  // the value left by Type

                  new=newType(T2.name,&T2.aType,0);// make some new storage
            new->tag=T2.tag;
                  T1.next=new;
                  pushType(&ST,&T1);

            } ;

SizeConstraint: SIZE {} Constraint {} ;

Constraint: OPENBR {}LowerEnd {} DOTDOT {} UpperEnd {} CLOSEBR {} ;

LowerEnd: NUMBER {}| MIN {} ;

UpperEnd: NUMBER {}| MAX {} ;

ReferencedType : DefinedType {} ;

DefinedType : typereference
          {
             } ;

BuiltInType : IntegerType    {} | BooleanType {}| SequenceType{}|
           CharacterStringType {} |
           SetOfType      {}| SequenceOfType {}| OctetStringType {}|
           TaggedType {}|NullType {}|ChoiceType {} |
           SetType {} | ObjectIdentifierType{}|
              RealType {} | BitStringType{}|
           AnyType {}| AnyDefinedByType{}|
           UTCTimeType {} | GeneralizedTimeType {} |
              ExtensionType {}
            | EnumeratedType        {} ;

CharacterStringType : IA5StringType | TeletexStringType |
                 PrintableStringType | BMPStringType|
                       UniversalStringType;

IA5StringType: IA5STRING {pushType(&ST,&IA5StringType);
                   };
TeletexStringType: TELETEXSTRING {pushType(&ST,&TeletexStringType);
                   };
PrintableStringType: PRINTABLESTRING {pushType(&ST,&PrintableStringType);
                   };
```

```
BMPStringType: BMPSTRING {pushType(&ST,&BMPStringType);
                    };
UniversalStringType: UNIVERSALSTRING {pushType(&ST,&UniversalStringType);
                    };

ExtensionType : EXTENSION lcID
             {Type e;
                    e=ExtensionType;
                    strcpy(e.ObjectIdentifier,$2);
//                  printf("\n***extension ");
                    sprintType(&e);
                    pushType(&ST,&e);
                   } ;

AnyDefinedByType: AnyDefinedByThing {} ;


AnyDefinedByThing: ANY DEFINED BY lcID  {pushType(&ST,&AnyDefinedByType);
            } | ANY DEFINED BY ID
            {pushType(&ST,&AnyDefinedByType);
            };

AnyType: ANY {pushType(&ST,&AnyType);} ;

NamedNumber: lcID {} _br SignedNumber {} br_
          {Type T,A;
              T=IdType;
              strcpy(T.name,$1);
              pullType(&ST,&A); // the value
              T.value=A.value;
              pushType(&ST,&T);

             };

SignedNumber: NUMBER
            {Type T;
                T.value=$1;
                   pushType(&ST,&T);
               } | MINUS NUMBER
               { Type T;
                T.value= - $1;
                   pushType(&ST,&T);
               } ;

EnumeratedType: ENUMERATED _br Enumerations br_
            {Type T;
               T=EnumeratedType;
               pushType(&ST,&T);
               };

Enumerations: RootEnumeration ; //| RootEnumeration XXX ;
```

```
XXX: COMMA DOTDOTDOT | COMMA DOTDOTDOT COMMA AdditionalEnumeration;

AdditionalEnumeration: Enumeration ;

RootEnumeration: Enumeration ;

Enumeration: EnumerationItem | EnumerationItem COMMA Enumeration ; // this needs extending

EnumerationItem: NamedNumber {}; // | lcID {}  ; // for the moment dont allow lcID

RealType: REAL {pushType(&ST,&RealType);} ;


BitStringType: BIT STRING {pushType(&ST, &BitStringType);} ;

ObjectIdentifierType: OBJECT IDENTIFIER {pushType(&ST,&ObjectIdentifierType);};

SequenceOfType: SEQUENCE OF
            {
            }
            Type
             {Type T1=SequenceOfType;
                   Type T2,*new;

                   pullType(&ST,&T2);  // the value left by Type

                   new=newType(T2.name,&T2.aType,0);// make some new storage
                   //printTypeList(&T2);
                   T1.next=new;
                   pushType(&ST,&T1);

                   } ;

SetOfType: SET OF Type
              {Type T1=SetOfType;
                   Type T2,*new;

                   pullType(&ST,&T2);  // the value left by Type
                   //printTypeList(&T2);
                   new=newType(T2.name,&T2.aType,0);// make some new storage

                   T1.next=new;
                   pushType(&ST,&T1);


                   } ;

SetType: SetThing {} ;

SetThing: SET _br br_ {pushType(&ST,&SetType);}|
        SET _br AlternativeTypeList br_
            {Type T1,T2;
               //printf("\n**set  type");
               T2=SetType;
            //printStack(&ST);
               //stack contains  "componenttype list "
            pullType(&ST,&T1);  //a ABC etc in linked list

            T2.next=T1.next;//the move the chain across
            pushType(&ST,&T2);
            //printStack(&ST);
               }    ;

OctetStringType : OCTET STRING {pushType(&ST,&OctetStringType);} ;
/* |
              OCTET STRING SyntaxDefinition
                     {
                      Type *SyntaxDef;
                      Type *new,A;
                      pullType(&ST,&A);
                      new=newType("",&asn1OctetString,0);
                      SyntaxDef=newType("",&asn1Syntax,0);
                strcpy(SyntaxDef->name,A.name);
                      SyntaxDef->next=A.next;
                      new->SyntaxDefinition=SyntaxDef;
                      pushType(&ST,new);
                    // printf("***OCTET STRING");
                    // printStack(&ST);
                     };
*/
ChoiceType: ChoiceThing {} ;

ChoiceThing : CHOICE _br AlternativeTypeList br_
          {Type T1,T2;
               //printf("\n**choice  type");
               T2=ChoiceType;
            //printStack(&ST);
               //stack contains  "componenttype list "
            pullType(&ST,&T1);  //a ABC etc in linked list
            T2.tag=T1.tag;
               printf("****choice tag=%d %d\n",T2.tag.value,T2.tag.flag);
            T2.next=T1.next;//the move the chain across
            pushType(&ST,&T2);
            //printStack(&ST);
               };


AlternativeTypeList : AlternativeType {} | AlternativeTypeList COMMA  AlternativeType
{Type T1,T2,*pnt;
              // we now take two types off the stack & join them together
               // so the top of stack gets one added onto its linked list
```

```
                //printf("\nalternative type list");
                        // printStack(&ST);
                        pullType(&ST,&T1);pullType(&ST,&T2);
                        // sprintType(&T1);
                        // sprintType(&T2);
                        pnt=T2.next; // the first malloced one
                // find the end of the chain

                while(pnt->next!=0) pnt=pnt->next;
                //pnt is now the end
                pnt->next=T1.next;
                pushType(&ST,&T2);
                        //printStack(&ST);
               };


AlternativeType: NamedType  {Type T,*new;
            // have a named type or a type  on the stack
                // create a newTYpe (malloc)
            // add it it on to the next pointer of the one on the stack
            // printf("\n**alternative type");
            // printStack(&ST);
                pullType(&ST,&T);
            new=newType(T.name,&T.aType,0);//new Type
                new->tag=T.tag;
            T.next=new;
            pushType(&ST,&T);
                // printStack(&ST);
            } |
           Type
           {Type T,*new;
            // have a named type or a type  on the stack
                // create a newTYpe (malloc)
            // add it it on to the next pointer of the one on the stack
              //printf("\n**alternative type");
              //printStack(&ST);
                pullType(&ST,&T);
            new=newType(T.name,&T.aType,0);//new Type
                new->tag=T.tag;
            T.next=new;
            pushType(&ST,&T);
                // printStack(&ST);
            };


NullType : _NULL {pushType(&ST,&NullType);} ;

TaggedType: TAG {Type T;
              T.tag.value=$1 +0xA0;
                      T.tag.flag=0;
              strcpy(T.name,"tag");
```

```
                    // printf("**tag=%d\n",T.tag.value);
                     pushType(&ST,&T);
                     // printStack(&ST);
              }
          Type
              {Type TagT,T;
                     printf("\n***tag type ");
                     //printStack(&ST);
                      pullType(&ST,&T);
                      pullType(&ST,&TagT);

                      T.tag=TagT.tag;

                      printf("\n****tag=%d %d",T.tag.value,T.tag.flag);
                      pushType(&ST,&T);
                      printStack(&ST);
              }|

        TAG {Type T;
                T.tag.value=$1;
              strcpy(T.name,"tag");
                     pushType(&ST,&T);
                     // printStack(&ST);
              }
        IMPLICIT
          Type
              {Type TagT,T;
                     printf("\n***IMPLICIT tag type ");
                     //printStack(&ST);
                      pullType(&ST,&T);
                      pullType(&ST,&TagT);
                      //printf("\n***tag=%d",TagT.tag.value);

                      T.tag.value=TagT.tag.value;
               T.tag.flag=IMPLICIT_TAG;
                      pushType(&ST,&T);
                      // printStack(&ST);
                     };
IntegerType : INTEGER {pushType(&ST,&IntegerType);} |
            INTEGER  _br NamedNumberList  br_ {pushType(&ST,&IntegerType);} ;

NamedNumberList: NamedNumber {} | NamedNumberList COMMA NamedNumber {};

NamedNumber: lcID OPENBR NUMBER CLOSEBR {};

SequenceType : SEQUENCE _br br_ {pushType(&ST,&SequenceType);}  |
             SequenceThing {} ;

SequenceThing:  SEQUENCE _br {} ComponentTypeList br_
             {Type T1,T2;
                //printf("\n**sequence type");
```

```
                  T2=SequenceType;
               //printStack(&ST);
                  //stack contains  "componenttype list "
               pullType(&ST,&T1);  //a ABC etc in linked list

               T2.next=T1.next; // move the chain across
               pushType(&ST,&T2);
               //printStack(&ST);
                  } ;


ComponentTypeList : ComponentType {} | ComponentTypeList COMMA  ComponentType
                 {Type T1,T2,*pnt;
                 // we now take two types off the stack & join them together
                  // so the top of stack gets one added onto its linked list
                        // printf("\n***componenttypelist ");
                       //  printStack(&ST);
                         pullType(&ST,&T1);
//                       printf("match=%s!",T1.matchingRule);
                         pullType(&ST,&T2);
//                       printf("match=%s$",T2.matchingRule);
                         pnt=T2.next; // the first malloced one
                  // find the end of the chain
                         while(pnt->next!=0) pnt=pnt->next;
                  //pnt is now the end
                  pnt->next=T1.next;
                  pushType(&ST,&T2);

                 };

ComponentType: NamedType
              {Type T,*new;
                printf("\n**Component type");
               // have a named type on the stack
                  // create a newTYpe (malloc)
               // add it it on to the next pointer of the one on the stack
              //printStack(&ST);
                  pullType(&ST,&T);
  //            printf("***1\n");
               new=newType(T.name,&T.aType,0);//new Type tagged on to T
  //            printf("***2\n");
               new->tag.value=T.tag.value;
               new->tag.flag=T.tag.flag;
               strcpy(new->ObjectIdentifier,T.ObjectIdentifier);
               T.next=new;

               pushType(&ST,&T);
               //printStack(&ST);
              } |
              NamedType OPTIONAL
             {Type T,*new;
```

```
                printf("\nComponent type optional");
              // have a named type on the stack
                  // create a newTYpe (malloc)
              // add it it on to the next pointer of the one on the stack
                  pullType(&ST,&T);
              new=newType(T.name,&T.aType,0);//new Type tagged on to T
                  new->tag.value=T.tag.value;
              new->tag.flag=T.tag.flag;
              new->optional=1;
              strcpy(new->ObjectIdentifier,T.ObjectIdentifier);
              T.next=new;
              pushType(&ST,&T);
  //            printStack(&ST);
             } |
             NamedType DEFAULT Value
             {Type T,*new;
                //printf("\nComponent type");
              // have a named type on the stack
                  // create a newTYpe (malloc)
              // add it it on to the next pointer of the one on the stack
                  pullType(&ST,&T);
              new=newType(T.name,&T.aType,0);//new Type tagged on to T
                  new->tag.value=T.tag.value;
              new->tag.flag=T.tag.flag;
              new->Default=1;
              strcpy(new->ObjectIdentifier,T.ObjectIdentifier);
              T.next=new;
              pushType(&ST,&T);
              //printStack(&ST);
             };

Value :  NUMBER {} | TRUE {} | FALSE {} |lcID {};

NamedType: lcID {Type T;
                strcpy(T.name,$1);
                pushType(&ST,&T);}
           Type
            {Type T,A;
//              printf("\n**named type");
//              printStack(&ST);
                pullType(&ST,&T);
//              sprintType(&T);
             pullType(&ST,&A); //lcID
               //  printf(" %s ",A.name);
             strcpy(T.name,A.name);
             pushType(&ST,&T);
             // printStack(&ST);
               //  printf("\n** end named type tag %d %d ",T.tag.value,A.tag.value);

             };
```

```
BooleanType : BOOLEAN {pushType(&ST,&BooleanType);} ;

UTCTimeType : UTCTime {pushType(&ST,&UTCTimeType);};

GeneralizedTimeType : GeneralizedTime {pushType(&ST,&GeneralizedTimeType);} ;

%%
// extern FILE *yyin;
/*
main(int argc,char *argv[])
{  ++argv, --argc;  /* skip over program name */
/*
            if ( argc > 0 )
                  yyin = fopen( argv[0], "r" );
            else
                  yyin = stdin;
yyparse();
 //printf("top=%s\n",top.name);
}
*/

yyerror (char *s){printf("%s\n",s);exit(0);}

yywrap () {/*printf("paircount is %d done\n",paircount);*/}
```

Common.h file

```
#if ! defined _oidstuff
#define _oidstuff
typedef struct _oid{int length;char *oidbytes;} OID;

typedef struct _oidfunct {void (* function)();unsigned char *oidString;} OidFunction;

typedef struct _oidref {char *name;
                   OID  *oid;
                            struct _oidref *next;
                        }OIDREF;
//OID *newOID(OID oid);
#endif
#if ! defined _choice

#define _choice

typedef void( * _CPF)(char * , unsigned char **);
typedef void(* _CPFPLUS)(char *,int,int,unsigned char **,int, void *,int,int, char *);
typedef struct _typedesc
   {
         char * name;
         int tag;// just for user specified tags
```

```
          int primitive; // 30 for SEQUENCE etc
          int tagOption;// IMPLICIT_TAG or EXPLICIT_TAG
          //   void (* function)(char *,unsigned char **) ;
          _CPF function;
          int optional;
          int Default;
          char *ObjectIdentifier;
   }
TypeDescription;
#endif

#if ! defined  _tagg
#define _tagg
typedef struct _tagg{
   int flag; // implicit or explicit
   int value;// the actual tagoidbytes

#define IMPLICIT_TAG 0
#define EXPLICIT_TAG 1

}Tag ;

#endif


#if ! defined _asndef
#define _asndef
typedef struct _asn1type{
   char name[100];
   int token;
} ASN1Type;
#endif


#if !defined _typedef
#define _typedef
typedef struct _type{
   char name[100];

   ASN1Type aType;
   struct _type *next;
   Tag tag;
   int optional;// 1 if OPTIONAL 0 otherwise
   int Default;// 1 if this is a default type
//    struct _type *SyntaxDefinition;
   char ObjectIdentifier[100];
   int value; // for NUMBER types & enumerations
   }Type;
#endif
```

```
#if ! defined _stackdef
#define _stackdef
typedef struct _stack{
    Type array[50];
    int index;
}Stack;
#endif

#if ! defined s_def
#define s_def
typedef struct _definition {
    char  name[100];
    struct _definition *next;
    Type *typelist;
    Tag tag;
    ASN1Type aType;
   // Type * SyntaxDefinition;
    char ObjectIdentifier[100];
//    char matchingRule[100];
} Definition;

#endif
```

pkc.asn1

```
X509 DEFINITIONS ::=
BEGIN

            Certificate  ::=  SEQUENCE  {
                  tbsCertificate       TBSCertificate,
                  signatureAlgorithm   AlgorithmIdentifier,
                  signatureValue       BIT STRING  }
              TBSCertificate  ::=  SEQUENCE  {
                  version         [0]  EXPLICIT Version DEFAULT v1,
                  serialNumber         CertificateSerialNumber,
                  signature            AlgorithmIdentifier,
                  issuer               Name,
                  validity             Validity,
                  subject              Name,
                  subjectPublicKeyInfo SubjectPublicKeyInfo,
                  issuerUniqueID  [1]  IMPLICIT UniqueIdentifier OPTIONAL,
                                 -- If present, version shall be v2 or v3
                  subjectUniqueID [2]  IMPLICIT UniqueIdentifier OPTIONAL,
                                 -- If present, version shall be v2 or v3
                  extensions      [3]  EXPLICIT Extensions OPTIONAL
                                 -- If present, version shall be v3
                  }
              Version  ::=  INTEGER  {  v1(0), v2(1), v3(2)  }
              CertificateSerialNumber  ::=  INTEGER
```

```
          AlgorithmIdentifier  ::=  SEQUENCE  {
              algorithm               OBJECT IDENTIFIER,
              parameters              ANY DEFINED BY algorithm OPTIONAL  }
          Name ::= CHOICE {
          rdnSequence  RDNSequence }
          RDNSequence ::= SEQUENCE OF RelativeDistinguishedName
          RelativeDistinguishedName ::=
            SET OF AttributeTypeAndValue
          AttributeTypeAndValue ::= SEQUENCE {
            type     AttributeType,
            value    AttributeValue }

          AttributeType ::= OBJECT IDENTIFIER
          AttributeValue ::= ANY DEFINED BY AttributeType
--   MODIFIED by E Ball
          Validity ::= SEQUENCE {
              notBefore      Time,
              notAfter       Time }
          Time ::= CHOICE {
              utcTime        UTCTime,
              generalTime    GeneralizedTime }
          UniqueIdentifier  ::=  BIT STRING
          SubjectPublicKeyInfo  ::=  SEQUENCE  {
              algorithm            AlgorithmIdentifier,
              subjectPublicKey     BIT STRING  }
          Extensions  ::=  SEQUENCE SIZE (1..MAX) OF Extension
          Extension  ::=  SEQUENCE  {
              extnID      OBJECT IDENTIFIER,
              critical    BOOLEAN DEFAULT FALSE,
              extnValue   OCTET STRING  }
END
```

ac.asn1

```
X509ATTRIBUTECERTIFICATE DEFINITIONS ::=
BEGIN

AttributeCertificate ::= SEQUENCE {
             acinfo              AttributeCertificateInfo,
             signatureAlgorithm   AlgorithmIdentifier,
             signatureValue       BIT STRING
         }

         AttributeCertificateInfo ::= SEQUENCE {
             version             AttCertVersion DEFAULT 1,
             holder              Holder,
             issuer              AttCertIssuer,
             signature           AlgorithmIdentifier,
             serialNumber        CertificateSerialNumber,
          attrCertValidityPeriod   AttCertValidityPeriod,
          attributes          Attributes,
          issuerUniqueID      UniqueIdentifier OPTIONAL,
          extensions          Extensions OPTIONAL
        }
Attributes ::= SEQUENCE OF Attribute

        AttCertVersion ::= INTEGER { v2(1)  }
        Holder ::= SEQUENCE {
           baseCertificateID   [0] IssuerSerial OPTIONAL,
                    -- the issuer and serial number of
                    -- the holder's Public Key Certificate

           entityName          [1] GeneralNames OPTIONAL,
                    -- the name of the claimant or role
           objectDigestInfo    [2] ObjectDigestInfo OPTIONAL
                    -- used to directly authenticate the holder,
                    -- for example, an executable
        }

        ObjectDigestInfo ::= SEQUENCE {
          digestedObjectType  ENUMERATED {
                publicKey            (0),
                publicKeyCert        (1),
                otherObjectTypes     (2) },
                        -- otherObjectTypes MUST NOT
                        -- be used in this profile
          otherObjectTypeID   OBJECT IDENTIFIER OPTIONAL,
          digestAlgorithm     AlgorithmIdentifier,
          objectDigest        BIT STRING
        }

        AttCertIssuer ::= CHOICE {
          v1Form   GeneralNames,  -- MUST NOT be used in this
                                  -- profile
          v2Form   [0] V2Form     -- v2 only
        }

        V2Form ::= SEQUENCE {
          issuerName            GeneralNames  OPTIONAL,
          baseCertificateID     [0] IssuerSerial  OPTIONAL,
          objectDigestInfo      [1] ObjectDigestInfo  OPTIONAL
            -- issuerName MUST be present in this profile
            -- baseCertificateID and objectDigestInfo MUST NOT
            -- be present in this profile
        }

        IssuerSerial  ::=  SEQUENCE {
          issuer         GeneralNames,
          serial         CertificateSerialNumber,
          issuerUID      UniqueIdentifier OPTIONAL
        }

        AttCertValidityPeriod  ::= SEQUENCE {
            notBeforeTime  GeneralizedTime,
            notAfterTime   GeneralizedTime
        }

    AlgorithmIdentifier  ::=  SEQUENCE  {
            algorithm               OBJECT IDENTIFIER,
            parameters              ANY DEFINED BY algorithm OPTIONAL  }

 CertificateSerialNumber::= INTEGER

  GeneralNames ::= SEQUENCE SIZE (1..MAX) OF GeneralName

  GeneralName ::= CHOICE {
     otherName                 [0]     AnotherName,
     rfc822Name                [1]     IA5String,
     dNSName                   [2]     IA5String,
     x400Address               [3]     OCTET STRING,
     directoryName             [4]     Name,
     ediPartyName              [5]     EDIPartyName,
     uniformResourceIdentifier  [6]     IA5String,
     iPAddress                 [7]     OCTET STRING,
     registeredID              [8]     OBJECT IDENTIFIER
  }

   AnotherName ::= TYPEIDENTIFIER

   TYPEIDENTIFIER ::= SEQUENCE {
     typeid         OBJECT IDENTIFIER,
     value          ANY DEFINED BY typeid }


  EDIPartyName ::= SEQUENCE {
     nameAssigner    [0] DirectoryString  OPTIONAL,
     partyName       [1] DirectoryString
  }
 UniqueIdentifier::= BIT STRING

Name::= CHOICE {rdnSequence RDNSequence }
 RDNSequence ::= SEQUENCE OF RelativeDistinguishedName

 RelativeDistinguishedName::= SET OF AttributeTypeAndValue

          AttributeTypeAndValue ::= SEQUENCE {
            type     AttributeType,
            value    AttributeValue }
AttributeType::= OBJECT IDENTIFIER
 AttributeValue::= ANY DEFINED BY AttributeType
```

```
DirectoryString ::= CHOICE {
  teletexString    TeletexString,
  printableString  PrintableString,
  bmpString        BMPString,
  universalString  UniversalString
}

-- Attribute types


  Extensions  ::=  SEQUENCE SIZE (1..MAX) OF Extension
            Extension  ::=  SEQUENCE  {
                 extnID      OBJECT IDENTIFIER,
                 critical    BOOLEAN DEFAULT FALSE,
                 extnValue   EXTENSION  extnID }

  Attribute ::= SEQUENCE {
                type      AttributeType,
                values    AttributeValues
                  -- at least one value is required
           }
AttributeValues::= SET OF AttributeValue

END
```

crl.asn1

```
X509 DEFINITIONS ::=
BEGIN
CertificateList  ::=  SEQUENCE  {
        tbsCertList          TBSCertList,
        signatureAlgorithm   AlgorithmIdentifier,
        signatureValue       BIT STRING  }

   TBSCertList  ::=  SEQUENCE  {
        version               Version OPTIONAL,
                              -- if present, shall be v2
        signature             AlgorithmIdentifier,
        issuer                Name,
        thisUpdate            Time,
        nextUpdate            Time OPTIONAL,
        revokedCertificates   SEQUENCE OF SEQUENCE  {
             userCertificate        CertificateSerialNumber,
             revocationDate         Time,
             crlEntryExtensions     Extensions OPTIONAL
                                    -- if present, shall be v2
                              }  OPTIONAL,
        crlExtensions          [0]  EXPLICIT Extensions OPTIONAL
                                    -- if present, shall be v2
                              }
```

```
Version::= INTEGER

  Name::= CHOICE {rdnSequence RDNSequence }
   RDNSequence ::= SEQUENCE OF RelativeDistinguishedName

   RelativeDistinguishedName::= SET OF AttributeTypeAndValue

              AttributeTypeAndValue ::= SEQUENCE {
                type     AttributeType,
                value    AttributeValue }
  AttributeType::= OBJECT IDENTIFIER
   AttributeValue::= ANY DEFINED BY AttributeType

 Time ::= CHOICE {

                utcTime        UTCTime,
                generalTime    GeneralizedTime }
              UniqueIdentifier  ::=  BIT STRING

Extensions  ::=  SEQUENCE SIZE (1..MAX) OF Extension

Extension  ::=  SEQUENCE  {
     extnID      OBJECT IDENTIFIER,
     critical    BOOLEAN DEFAULT FALSE,
     extnValue   OCTET STRING  }

CertificateSerialNumber  ::=  INTEGER

AlgorithmIdentifier  ::=  SEQUENCE  {
        algorithm               OBJECT IDENTIFIER,
        parameters              ANY DEFINED BY algorithm OPTIONAL  }

END
```

Example x509attrtypes.txt file

Define_attr X509Certificate.tbsCertificate.version **x509version**
Define_attr X509Certificate.tbsCertificate.validity.notBefore **x509validityNotBefore**
Define_attr X509Certificate.tbsCertificate.validity.notAfter **x509validityNotAfter**
Define_attr X509Certificate.tbsCertificate.Extention.holderAltName.rfc822name
x509pkcHolderRfc822Name
Define_attr X509Certificate.tbsCertificate.Extention.holderAltName.HolderDnsName
x509pkcHolderDnsName
Define_attr X509Certificate.tbsCertificate.Extention.holderAltName.HolderDN **x509pkcHolderDN**
Define_attr X509Certificate.tbsCertificate.Extention.holderAltName.HolderURI **x509pkcHolderURI**
Define_attr X509Certificate.tbsCertificate.Extention.holderAltName.HolderIpAddress
x509pkcHolderIpAddress
Define_attr X509Certificate.tbsCertificate.Extention.holderAltName.HolderRegisteredID
x509acHolderRegisteredID

Define_attr X509Certificate.tbsCertificate.Extention.IssuerAltName.IssuerRfc822Name **x509IssuerRfc822Name**

Define_attr X509Certificate.tbsCertificate.Extention.IssuerAltName.IssuerRfc822Name **x509IssuerDnsName**

Define_attr X509Certificate.tbsCertificate.Extention.IssuerAltName.IssuerRfc822Name **x509IssuerURI**

Define_attr X509Certificate.tbsCertificate.Extention.IssuerAltName.IssuerRfc822Name **x509IssuerIpAddress**

Define_attr X509Certificate.tbsCertificate.Extention.IssuerAltName.IssuerRfc822Name **x509IssuerRegisteredID**

Define_attr X509Certificate.tbsCertificate.Extention.IssuerAltName.IssuerRfc822Name **x509authorityCertIssuer**