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

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 that the XPS can also be embedded inside an LDAP server, and can intercept the LDAP operations before passing them to the LDAP 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 administrators' needs. The XPS will talk LDAPv3 protocol only.

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.

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 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 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 dn2wal() if the type of operation is add or 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, it will open the WAL. 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 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

[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.

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 its process id as the base for the filename.

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 would be ignored.

If EnableXPS is yes then the administrator should decide if the LDAP server is internal or remote to the XPS. For this purpose there is an ldapurl option. If LDAP server is internal then this field should be set to the NULL value. If not, then it should hold an actual URL of the LDAP server (with port number). Also, as was mentioned before, the XPS will talk LDAPv3 protocol only, so if the server whose URL is specified in this configuration file does not support the LDAPv3 protocol, XPS will return an error.

```
ldapurl             [NULL|<URL of LDAPv3 server>]
```

The default value of this option is NULL.

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.

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

The default value is yes.

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.

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

The default value is no.

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

```
RevokedRDNformat          [x509serialNumber+x509issuer | x509isssuerSerial |
                           x509serialNumber]
```

The default value is x509serialNumber.

There are several formats for the RDN of X.509 certificate entries defined in [Gietz] and [Chadwick], so the XPS administrator will have a choice of which format to use. In order to use one particular format (x509serialNumber+x509issuer) the administrator will need to know if the LDAP server supports multiple valued RDNs.

The following options of RDN are available for naming certificates (PKCs and ACs):

-x509serialNumber+x509issuer - multiple valued RDNs will be created from the x509serialNumber and x509issuer attribute values.

-x509serialNumber - this format of the RDN will be suitable for the cases when all certificates are issued by one CA.

-x509isssuerserial - administrators should use this format when the LDAP server does not support multiple valued RDNs.

```
CertRDNformat             [x509serialNumber+x509issuer | x509isssuerSerial |
                           x509serialNumber]
```

The default value is x509serialNumber+x509issuer.

Write Ahead Logs (WAL) are needed for internal purposes, so the administrator may specify a path for the WAL:

```
Walpath                   [path]
```

The default path if this parameter is missing, is /usr/local/var/xps/wals/

The Administrator may specify a path for the XPS error logs:

```
XPSerrorlog          [path/filename]
```

The default file is /usr/local/var/xps/error.log

The following option specifies the path to the file containing the matching table between the ASN.1 type references of fields in the X.509 attributes and their equivalent LDAP attribute type names defined in the PKIX schemas [Chadwick], [Gietz], [Sahalayev].

```
x509attrtypes  [path/filename]
```

The default file is /usr/local/var/xps/x509attrtypes.txt

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, and LDAPURL is NULL 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.

.

If EnableXPS is YES, and LDAPURL points to an external LDAP server, then the following operations will pass straight through the XPS server and not be affected by it: Bind, Unbind. The following operations will be intercepted by the XPS code and a referral to the external LDAP server will be sent in reply to the operations: 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 a referral will be returned to the LDAP client if the LDAP server is remote or the request will be processed as normal if the LDAP server is internal. If it does then a list of Modifications (in OpenLDAP internal opaque structure) from each X.509 attribute will be created. This list will be created using one of the x509***_2_mods() functions (see [Parsing] for a full explanation). Modifications will contain all the information about the certificate or CRL and they can be easily transformed into the Entry structure using slap_mods2entry() or into an LDAP ADD request. All parameters for new child entries (replication, access controls etc.) will be the same as for the new 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 dn2wal() and the list of child DNs will be extracted from the Modifications and written to the WAL. The WAL will then be flushed. Then how the entries will be added depends on whether the server is internal or remote.

Specifics of ADD operation for internal server

The function (be->be_add)() will be used to add the parent entry to the directory. All required information will be taken from the user's ADD request. If duplicateCertificate is false, the X.509 attributes will be removed from the parent entry.

Then an Entry structure will be created from the mods list using mods2entry() for the first child entry and then (*be->be_add)( ) will be used to add the child entry to the directory. These operations will be repeated for each child (and grandchild) in the list of Modifications.

If no error occurs the WAL will be cleaned and the client will receive 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. Using the DNs from the WAL all the entries will be deleted. After each entry is successfully deleted, its DN 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.

Specifics of the ADD operation for remote server

If the LDAP server is remote an ADD request for the original entry will be created. It will basically contain the same information as the original ADD request (except possibly for the X.509 attribute). It will be sent to the LDAP server defined in the configuration file. To do that, an LDAP_ADD_REQUEST will be created from the mods list (We will use the same code as the ldapmodify client code) and the DN of the parent entry will be saved in the WAL along with the names of all the children (and grandchildren if revokedCertificateEntries is TRUE). The newly created LDAP request will be sent using ldap_add_ext_s().
If no errors occurred the DN of the first child entry will be created from the mods list. An LDAP_ADD_REQUEST will be created from the mods list. The newly created LDAP request will be sent using ldap_add_ext_s(). If no errors occur, this will be repeated for each child and grandchild entry in the mods list.
The WAL will be cleaned and the user will receive an LDAP_SUCCESS return code as the result of his ADD operation.
If there were any errors an LDAP_DELETE request will be created for each entry whose DN is found in the WAL. If any errors occur during the removal of the DNs, the DNs will be stored in the WAL, and error messages will be stored in the OpenLDAP log (if logging is switched on). After that the administrator will be able to remove the new entries manually.

DELETE operation

When XPS receives a DELETE operation XPS will perform an internal or external full subtree SEARCH operation from the original entry as the base entry, searching for object class *present*. If no entries are returned, either a referral will be returned to the LDAP client if the LDAP server is remote, or the request will be processed as normal if the LDAP server is internal. If all of the returned entries are of the *x509base* object class [Sahalaev], the DELETE will be processed. If any entries are not of the *x509base* object class, then the DELETE operation will be rejected. The way that the actual deletion will be accomplished depends on whether the server is internal or remote.

Specifics of the DELETE operation for internal server.

If the LDAP server is internal full subtree searching will be done with (*be->be_search)( ). All returned entries will initially be checked to ensure that they are of object class *x509base*. If any is not the DELETE will be rejected.The set of returned entries will be processed one by one. If an entry has the *hasSubordinates* operational attribute = FALSE, the entry will be saved in the WAL using Entry2wal() and then deleted from the directory using (be->be_delete)(). If the *hasSubordinates* operational attribute = TRUE, the entry will be passed over until the complete set of entries has been processed. The remaining set (all with *hasSubordinates* operational attribute = TRUE) will then be processed again by saving in the WAL and then deleting from the directory. None of them should now have subordinate entries.
Once all entries from the search result are deleted the parent entry will be deleted from the directory.
If no error occurred the WAL will be cleaned and client will receive the LDAP_SUCCESS result code. If there were any errors LDAP should be rolled back to it original state. All deleted enties are stored in WAL. They will be added back to the directory using (*be->be_add)( ). When an entry is added 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 them manually.

Specifics of the DELETE operation for remote server.

If the LDAP server is remote, a full subtree SEARCH request will be created and sent to the remote server using ldap_search_ext( ). All returned entries will be checked to ensure that they are of object class *x509base*. If any is not the DELETE will be rejected.
The set of returned entries will be processed one by one. If an entry has the *hasSubordinates* operational attribute = FALSE, the entry will be saved in the WAL,, then an LDAP DELETE request for this Entry will be created and sent to the LDAP server using ldap_delete_ext_s().If the *hasSubordinates* operational attribute = TRUE, the entry will be passed over until the complete set of entries has been processed. The remaining set (all with *hasSubordinates* operational attribute = TRUE) will then be processed again by saving in the WAL and then deleting from the directory. None of them should now have subordinate entries.
Once all entries are deleted successfully, the parent entry will be deleted.
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 its original state. All deleted enties are stored in the WAL. They will be added back to the directory.(care will need to be taken with CRLs that CRL entries are added before revoked certificate entries.) An ADD request will be created and sent using ldap_add_ext_s(). After each entry is added 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 them will be strored in the OpenLDAP log. The administrator will have to add them manually.

MODIFY operation

If XPS receives an LDAP_MODIFY request it will first check if there are any X.509 attributes in the modifications. If there are none of then a referral will be returned to the LDAP client if the LDAP server is remote or the request will be processed as normal if the LDAP server is internal.
Otherwise, depending on the operation type (mod_op), the following actions will be taken.

Operation type is add.

A list of Modifications (OpenLDAP internal opaque structure) from each X.509 attribute using x509***_2_mods() will be created. All parameters for the new child entries (replication, access controls etc.) will be the same as for the parent entry being modified.
If the parsing of the X.509 attribute value(s) finished successfully the list of child (and grandchild, if revokedCertificateEntries is TRUE) DNs will be extracted from the Modifications and written to the WAL. If XPS is configured to store original attributes as well as X.509 entries (duplicateCertificates = TRUE), then attribute types and values from the MODIFY request will be stored in the WAL. The WAL will then be flushed.

Internal Server

An Entry structure will be created from the mods list using mods2entry() for the first child entry and then (*be->be_add)( ) will be used to add the child entry to the directory. These operations will be repeated for each child (and grandchild) in the list of Modifications.
If there was any error the internal server should be rolled back to its original state. If entryAlreadyExists is returned, the corresponding DN should be removed from the WAL. At this point the WAL will contain a list of entries to be added, some of which may have been added and some of which have not. Using the DNs from the WAL all the entries will be deleted. After each entry is successfully deleted, its DN will be removed from the WAL. If the rollback is successful the original error code will be returned to the user.
If any errors (apart from noSuchObject) will occur during rollback the LDAP directory may

contain entries that are not supposed to be there. Their DNs will be stored in the WAL and OpenLDAP log. It will enable the administrator to remove them manually.
If (duplicateCertificates = TRUE), (be->be_modify)() will be used to add the original Attribute to the directory. All required information will be taken form the user's MODIFY request.

Remote Server

If the LDAP server is remote an LDAP_ADD_REQUEST will be created from the mods list for the first child entry. The newly created LDAP request will be sent using ldap_add_ext_s(). If no errors occur, this will be repeated for each child entry in the mods list, followed by each grandchild entry.
If there were any errors the LDAP server will need to be rolled back. If entryAlreadyExists was returned, then the corresponding DN will be removed from the WAL. An LDAP_DELETE request will be created for each entry whose DN is found in the WAL. If rollback is successful, the original error will be returned to the user.
If any errors occur during the removal the DNs will be stored in the WAL and in OpenLDAP log. After that the administrator will be able to remove them manually.
If duplicateCertificates = TRUE, A MODIFY request for the original X.509 Attribute will be created. It will basically contain the same information as the original MODIFY request. It will be sent to the LDAP server defined in the configuration file. To do that, an LDAP_MODIFY_REQUEST (add values) will be created from the mods list (We will use the same code as the ldapmodify client code) and the original X.509Attribute will be saved in the WAL along with the DNs of all the children to be added. The newly created LDAP request will be sent using ldap_modify_ext_s().

Operation type is delete.

If no values are provided (i.e. Delete the attribute), XPS will perform a one level SEARCH operation from the original entry as the base entry, searching for the attribute type. If the attribute type is attributeCertificateRevocationList, attributeAuthorityRevocationList, certificateRevocationList or authorityRevocationList, and revokedCertificateEntries is TRUE, a further one level search will be performed from the entry or entries returned from the first one level Search. All these child (and grandchild) entries will be added to the WAL and then deleted.

If values are provided, XPS will search for these values to check that they exist, by performing an equality match on the entire value. If they do exist they will be stored in the WAL. If any of them don't exist, the user will be sent an error message and rollback will be started. If the attribute type is attributeCertificateRevocationList, attributeAuthorityRevocationList, certificateRevocationList or authorityRevocationList, and revokedCertificateEntries is TRUE, a further one level search will be performed from the entry or entries returned from the first one level Search. These entries will be stored in the WAL and then deleted. When all have been deleted, the first set of returned entries will then be deleted.

Internal Server

If the LDAP server is internal the search for the attribute values will be done with (*be->be_search)().
If all the entries are found they will be saved in the WAL using Entry2wal(). If the attribute type is attributeCertificateRevocationList, attributeAuthorityRevocationList, certificateRevocationList or authorityRevocationList, and revokedCertificateEntries is TRUE, a further one level search will be performed from the entry or entries returned from the first one

level Search. These are stored in the WAL and then deleted from the directory using (be->be_delete)(). Finally the original set of entries are deleted.
If duplicateCertificate is TRUE and all entries from the search result are deleted the original X.509 Attribute will be saved in the WAL using attr2wal() and deleted from the parent entry.
If there was any error LDAP should be rolled back to it original state. All deleted enties were stored in the WAL. They will be added back to the directory using (*be->be_add)( ). When an entry is added it will be removed from the WAL. Finally if duplicateCertificate is TRUE the original X.509 Attribute will be restored.
If any errors occur during rollback some entries may be missing from the directory. All of them will be strored in the WAL and OpenLDAP log. The administrator will have to add them manually.

Remote Server

If the LDAP server is remote the SEARCH request will be created and sent to the remote server using ldap_search_ext( ).
If all the entries are found they will be saved in the WAL. If the attribute type is attributeCertificateRevocationList, attributeAuthorityRevocationList, certificateRevocationList or authorityRevocationList, and revokedCertificateEntries is TRUE, a further one level search will be performed from the entry or entries returned from the first one level Search. These are stored in the WAL, then an LDAP DELETE request for each Entry will be created and sent to the LDAP server using ldap_delete_ext_s(). Finally, the initial set of entries saved in the WAL will be deleted.
If duplicateCertificate is TRUE and all entries are deleted successfully the original X.509Attribute will be saved in the WAL and then also deleted from the parent entry.
If there was any error the remote server should be rolled back to it original state. All deleted enties have been stored in the WAL. They will be added back to the directory. An ADD request will be created and sent using ldap_add_ext_s(). When each entry is added it will be removed from the WAL. Finally if duplicateCertificate is TRUE the original X.509 Attribute will be restored to the parent entry.
If any errors occur during rollback some entries may be missing from the directory. All of them will be stored in the WAL and the OpenLDAP log. The administrator will have to add them manually.

Operation type is replace.

In this case XPS will delete the X.509 attribute from the parent entry (if duplicateCertificate = True), and all the corresponding child (and grandchild if revokedCertificateEntries is TRUE) entries and then add the new X.509 attribute to the parent (if duplicateCertificate = True) and create the child entries (and in the case of CLRs with revokedCertificateEntries is TRUE, grandchild entries). The sequence of actions is the same as in the preceding paragraphs (equivalent to delete the attribute with no attribute values provided, and add the new attribute values in the replace).

After all Modify Operations have completed successfully

The WAL will be cleaned and the user will receive an LDAP_SUCCESS return code as the result of his MODIFY operation.

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 Function

x509_2_mods(
        Modifications **mods,   // The output array of the modifications lists. Each list will be created from a single X.509 attribute value.

```
        Attribute **a,                  // Input pointer to the original x.509 attribute
        const char **text                     // Output error message, used for debugging
        )
```
This function will parse all of the X.509 attribute values in the input attribute, and will fill in the lists of modifications which will be stored in the array of mods variables. It will require a pointer to the X.509 attribute received from the user.

OpenLDAP internal functions for updating the backend database

(*be->be_add)( ) – adds an entry
(*be->be_delete)( ) – deletes an entry
(*be->be_modify)( ) – modifies an entry

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.

LDAP API Operation Functions

ldap_search_ext_s( ) - API function for sending a search request to an LDAP server.
ldap_delete_ext_s() - API function for sending a delete request to an LDAP server.
ldap_add_ext_s() - API function for sending an add request to an LDAP server.
ldap_modify_ext_s() - API function for sending a modify request to an LDAP server.
For further references see [API].

WAL Functions

Dn2wal() - this function will save the DN of an added entry in the WAL. It is used for rolling back added entries, if necessary.
Entry2wal() - this function will save an entry-to-be-deleted in the WAL. It is used for rolling back deleted entries.
attr2wal() - this function will save the attributes of entries-to-be-modified in the WAL. It is used in rolling back entry updates.

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