X.500 and LDAP Security: A Comparative Overview

Vesna Hassler
Technical University of Vienna

Abstract

In this article we give a comparative overview of the X.500 and LDAPv3 Directory security features. X.500 is a commonly used name for a series of joint ISO/IEC and ITU-T standards specifying a distributed directory service. It assumes the existence of an underlying OSI protocol stack. LDAP is an Internet alternative to the X.500 Directory Access Protocol (X.511 DAP). Since its first version LDAP has undergone significant changes, and many of them concern security. It was originally planned to use LDAP only to access the X.500 directory via an LDAP gateway. In the meantime, LDAP functionality was extended, which enables LDAPv3 to be used for both the server model and the client read and update access protocol.

X.500 is a commonly used name for a series of joint International Organization for Standardization/International Electrotechnical Commission (ISO/IEC) and International Telecommunication Union — Telecommunication Standardization Sector (ITU-T) standards specifying a distributed directory service. The work on X.500 was initiated in 1984 mainly by two communities of users: the International Consultative Committee for Telephone and Telegraph (now the ITU-T), which wanted to provide a white pages service for phone numbers or X.400 O/R addresses; and the ISO and ECMA, which wanted to provide a name server service for OSI applications [1].

A user wishing to obtain some information from the directory uses an application called a *directory client*. The directory client accesses a *directory server* by using a standardized protocol. This protocol is referred to in X.500 as the Directory Access Protocol (DAP). The X.500 directory service, however, assumes the existence of an underlying OSI protocol stack. Unfortunately, this is the main reason X.500 never became widely used, in addition to its high design complexity.

There are a number of X.500 products supporting DAP [2]. One of the most popular free products is QUIPU, containing an X.500 directory client from the public ISODE[1] releases. The ISODE OSI package can be layered on top of TCP/IP; it actually adds an additional transport sublayer that "hides" TCP and "pretends" to be an ISO transport protocol. The upper open system interconnection (OSI) layers (presentation, session, and application layer) can be layered on top of the transport sublayer. A similar concept was developed for X.25 (packet) networks. It was, however, insufficient to increase the widespread popularity of X.500, mainly due to the extremely complex server administration, which was especially true for the 1993 edition of the X.500 standard. The main application areas were Intranet-based and company-wide directory services.

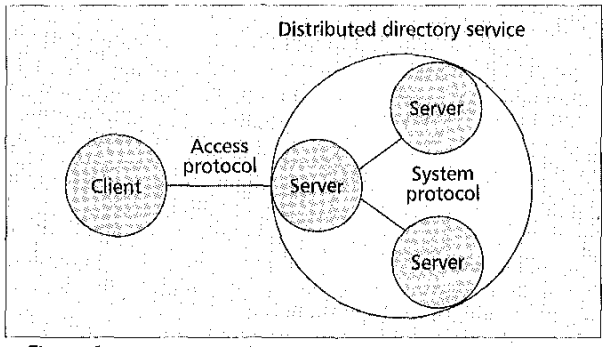
Lightweight Directory Access Protocol (LDAP) is an Internet alternative to the standard X.500 Directory Access Protocol (DAP) [3]. Since its first version LDAP has undergone significant changes, and many of them concern security. Originally it was planned to use LDAP only to access the X.500 directory via an LDAP gateway. In the meantime, LDAP functionality was extended, which enabled LDAPv3 to be used for both the server model and the client read and update access protocol. More information about X.500 and LDAPv3 can be found in [4].

LDAPv3 has adopted some of the X.500 security concepts and some of the widely accepted Internet security solutions, such as Transport Layer Security (TLS) [5]. However, it is very difficult to obtain a clear picture of which security services are available in X.500 and LDAPv3 and how they are related without having to study a series of X.500 standards and Internet Engineering Task Force (IETF) documents. In this article we give a comparative overview of X.500 and LDAPv3 security features. We analyze the security of the directory operations provided by X.500 [3] and LDAPv3 [6]. Our analysis is based on the 1995 editions of the ISO directory standards [3, 7, 8] and the latest (as of July 1999) IETF documents [6, 9–16].

The article is organized as follows. In the following section we give a brief tutorial on directory services. An overview of the authentication services for X.500 and LDAPv3 is given next. We then explain how to protect the directory operations.

[1] *ISODE has recently merged with Execmail into MessagingDirect.*

0890-8044/99/$10.00 © 1999 IEEE

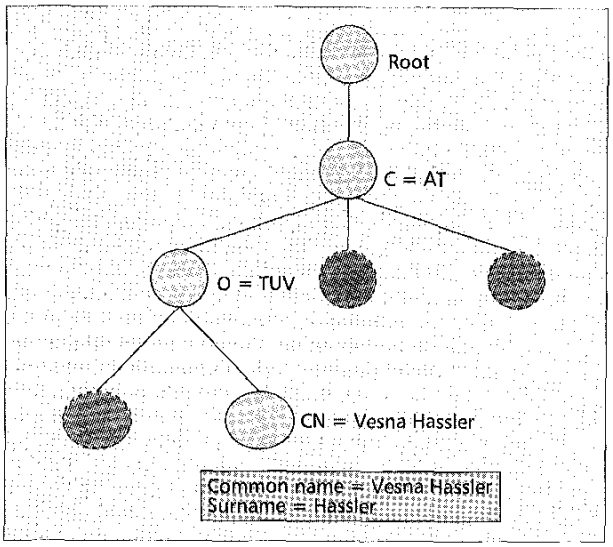


■ Figure 1. *A distributed directory service.*

The article gives an overview of the access control mechanisms. We list the directory attributes relevant to security services, and then conclude the article.

Directory Services

Many applications, such as browsing the Web, sending/receiving e-mail, or organizing enterprise information on the Intranet, need directory support. A *directory-enabled* application uses directory services to become more efficient, faster, more convenient, and easier to use [17]. Such applications are actually *clients* to directory *servers* that provide *directory services*. X.500 extended that concept to a *distributed* or *global* directory service, as illustrated by Fig. 1. A distributed directory service consists of a set of interconnected directory servers communicating by a *system protocol*. A client uses an *access protocol* to contact the nearest server and send a request (e.g., asking for a person's e-mail address). If the server cannot satisfy the request it may forward it to other servers in the group, process their responses, and build the final response to be sent to the client. In this way the servers provide a distributed directory service that is location-transparent to the client.



■ Figure 2. *The directory information tree.*

The Directory Data Model

According to the X.500 data model adopted by LDAP, the directory information is organized in a tree-like hierarchy (Fig. 2). The hierarchy, referred to as the *directory information tree* (DIT), is only a logical one, since it can span the information physically stored in the repositories of several directory servers. A piece of information, or a group of related pieces, is assigned its own *entry*. Entries are represented by circles in Fig. 2. The data structure of an entry is specified by the entry's *object class*. For example, the object class of the {CN = Vesna Hassler} entry may be "person"; this object class must contain a person's common name and surname, and may contain some other data, such as the phone number or password [10]. The actual data are stored as *attribute type* and *value* pairs (e.g., surname = Hassler).

Each entry has a tag referred to as the *relative distinguished name* (RDN), for example {CN = Vesna Hassler}. The set of RDNs from the Root entry down to a specific entry form a *distinguished name* (DN) of the entry. The Root entry, however, is not listed in the DN. For example, the DN of Vesna Hassler's entry is {C = AT, O = TUV, CN = Vesna Hassler}, meaning that a person whose common name is Vesna Hassler works at the Technical University of Vienna in Austria. The DN defines the *path* to the entry, that is, it helps us find the entry in the DIT.

Operation	Description	X.500	LDAP
	Bind and unbind		
bind	Client establishes a connection to server	*	*
unbind	Client closes an existing connection to server	*	*
	Directory read		
read	Client reads data from a specified entry	*	
compare	Client compares some data with directory data	*	*
abandon	Client asks server to abandon an operation	*	*
	Directory search		
list	Client lists all entries under a specified entry	*	
search	Client searches for data in one or more entries	*	*
	Directory modify		
modify	Clients modifies directory data	*	*
add	Client adds one or more entries	*	*
remove (delete)	Client removes one or more entries	*	*
modifyDN	Client modifies DN of an entry	*	*

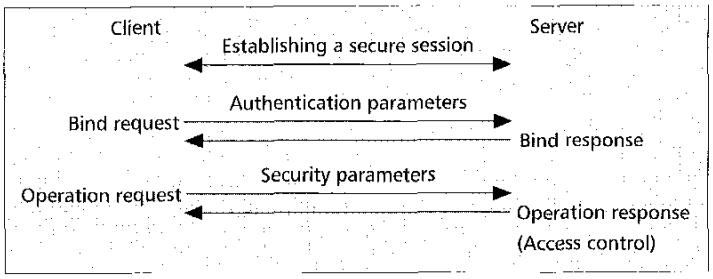
■ **Table 1.** *Directory operations.*

Security Issues

In order to determine what kind of protection, or *security services*, are necessary for directory clients and servers communicating over an insecure network, we must examine the following:

- Which types of operations are allowed by the directory access protocol
- For each protocol operation, which security attacks it should be protected against

In general, a protocol operation can be either a *request* (sent from the client to the server) or a *response* (sent from the server to the client). The directory operations provided by X.500 and LDAP are shown



■ Figure 3. *Accessing a directory server securely.*

in Table 1. The operations *read* and *list* are missing in LDAP since the equivalent functionality can be achieved with the *search* operation. In X.500 the *abandon* operation may be used to abandon an operation that interrogates the directory (*read*, *compare*, *list*, and *search*). LDAP is not specific about which operation may be abandoned.

The directory client is actually an *application* that communicates with the directory server on behalf of a specific user. We use the terms *client* and *user* interchangeably, but always with the same meaning: the client application with the identity of the user on whose behalf it communicates with the directory.

Bind and Unbind — The client can bind *anonymously* to the directory. In this case he has the default privileges of an anonymous user. In some cases only specific clients are allowed to establish a connection. They must prove their identity through an *authentication procedure*. This can, for example, be necessary if there is a risk that anonymous users have a good starting point to break into the system. In this case an *entity authentication* service is necessary. In the client-server world this type of authentication is referred to as *client-server authentication* (discussed later). We can distinguish the following three cases:

- The client must be authenticated by the server.
- The server must be authenticated by the client.
- Both the client and server must be authenticated to each other (i.e., mutually).

When disconnecting (or unbinding) from the directory, no security service is necessary since no harm can be done.

Directory Read — The *read* operation requires the client to have read privileges for a specific entry. Moreover, the client may have access to only specific data within the entry. For example, the client may be allowed to read the attribute types only, without being able to see their values. Or the client may be allowed to see an entry, but not to read the data stored in it. This can be further specified for the individual attribute types and values. What the client is allowed to see is decided by the directory *access control* mechanism (discussed later). The access control mechanism requires the client's identity to be checked in the authentication process. In addition, it is sometimes necessary that the strength of the authentication procedure be known as well (*authentication level*). For example, for sensitive information it may be required that the user be authenticated based on his public key certificate. In this case it is insufficient that the user's identity was checked with his password only, even if the password was correct. The rationale behind it is that it is much easier for an intruder to guess a password than to break a public key algorithm.

The *compare* operation is slightly different from the security perspective. First, the client must be granted the read permission for the entry in which the information to be compared is stored. Second, the client must be granted the compare permission for the attribute storing the information being compared. Note that the compare permission is weaker than the read permission. If the client has the compare permission for an attribute only, he can only try to guess the attribute value and compare the guesses with the actual attribute value. If the client has the read permission, he can simply read the attribute he is interested in.

The *abandon* operation does not involve any serious security issues.

Directory Search — The *list* operation may, similar to the *read* operation, require the client to be authenticated first. It may, however, be required that the permissions to browse the DIT and to return the DN of an entry be explicitly granted. This may be necessary if the structure of the directory would reveal some sensitive information about, for example, an enterprise directory.

The *search* operation may also require client authentication. In addition to the read permission, it may require that the client be explicitly granted the permission to perform matching against a search criterion (*filter*) for specific attribute types or values. The *search* operation can also be used in a similar way to the *read* or *list* operation. In such cases it requires similar protection to the corresponding operations, as described above.

Directory Modify — Virtually every directory requires the client to be authenticated in a proper way to perform the *modify*, *add*, *remove*, or *modifyDN* operation because these operations change the directory information. The modify operation deals with attributes, the other three with entries.

Protecting Requests and Responses — The server may need to be sure that a particular request is coming from a specific client. This situation is possible, for example, in LDAP, since the *bind* operation need not necessarily be used (and therefore the entity authentication procedure).

The client retrieves information from the directory. Naturally, he wants to be sure that the information is coming from the server he has chosen (*data authentication*), and that the server responses have not been modified in transit (*data integrity*). Sometimes it is also necessary to protect the privacy (or *confidentiality*) of the response contents.

Generally, there are two ways to protect client requests and server responses:

- To establish a secure session protecting a series of request-response pairs
- To protect each request or response individually

When establishing a secure session, an entity authentication procedure is performed, as described earlier. The secure session adds an *underlying security layer* to protect data authentication, data integrity, and, if necessary, data confidentiality. This protection is transparent to all messages exchanged during the lifetime of the session. In other words, the directory operations are unaware of the protection.

If no secure session is established, the operations needing protection must transfer security information within the corresponding messages. For example, if the integrity of a request must be protected, the client can append the integrity check information (e.g., MAC, discussed later).

In addition, it may be necessary that the server guarantee to have sent specific data. In other words, it may be necessary to prove the origin of a specific server response to a third party or in court at a later time. The security service that enables this is called *nonrepudiation* of origin and may only be applied to individual messages. The service is implemented with digital signatures and public key certificates. With public key cryptography there are two keys, one public and one private, whereby the public key can (and should) be made public. The public key is used to encrypt messages intended for the owner of the

key pair, because only he knows the corresponding private key necessary to decrypt the message. Some public key algorithms, such as RSA, can be used to compute *digital signatures*. Digital signatures are generated by encrypting a digital document with the signatory's private key. Since the private key is known to the signatory only, the signature is unique. An additional property of digital signatures compared to hand-written signatures is that they also depend on the document: if the document is changed, the signature changes as well.

Disclosure on Error — There is one more issue important to all operations. It may be that the client is not allowed to read, for example, an attribute, but is allowed to learn about the *existence* of the attribute. There may be, however, a different case, where the client is not allowed to learn about the existence of a data item. In such a case the directory must be careful with the error messages that may be returned when the user is trying to, for example, perform a compare operation on a data item. If, for example, the *insufficientAccessRights* error message is returned, the user will know about the existence of the data item, even if he has no permissions for it at all. For example, in X.500 in such cases either the *noSuchObject* or *noInformation* error message must be returned, which says nothing about the data item.

Client-Server Authentication

Figure 3 provides an overview of the three main possibilities for secure client-server communication. Each of them provides a different combination of security services. In this section we will describe client-server authentication taking place *before* invoking a directory operation. This type of authentication is referred to as *entity authentication*. In general, there are two possible scenarios:

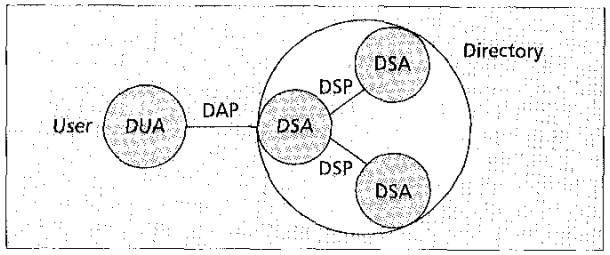
- To use an *external* security layer to establish a secure session before binding to the directory, represented by the Establishing a secure session arrow in Fig. 3
- To use the protocol structures to authenticate when binding to the directory, represented by the Authentication parameters arrows in Fig. 3

An additional scenario is to protect individual directory operations, that is, to send security parameters within operations (represented by the Security parameters arrows in Fig. 3). This type of authentication is referred to as *data* or *message authentication*, and will be described later. After the authentication procedure, the server-side access control determines whether the client is authorized to perform a particular operation on the directory data. Access control is explained later.

X.500 Authentication Parameters

Figure 4 illustrates a distributed directory service from Fig. 1 using the X.500 terminology. DAP is a protocol for read or update access to the X.500 directory. The client (directory user agent, DUA) uses DAP to access a directory server (directory system agent, DSA). The servers are interconnected and can interoperate to provide a global directory service to the user transparently. The servers communicate using the Directory System Protocol (DSP). The security services discussed in this article can in general be applied to both DAP and DSP, but we will in general refer to DAP. There are two more server operational protocols for communication between DSAs: Directory Operational Binding Management Protocol (DOP) and Directory Information Shadowing Protocol (DISP), but they are outside the scope of this article.

Bind request is a mandatory operation. Its arguments are protocol version, required authentication level, and the corresponding credentials. Depending on the type of access and the server requirements, the user may be authenticated using one of the following mechanisms:



■ Figure 4. *X.500 distributed directory service.*

- No authentication
- Simple authentication
- Strong authentication
- External authentication

The desired type of authentication is indicated through the choice of credentials in the corresponding optional argument of the *bind* operation:

```
Credentials ::= CHOICE {
simple                  [0] SimpleCredentials,
strong                  [1] StrongCredentials,
externalProcedure       [2] EXTERNAL }
```

Simple authentication is based on user passwords. In the simplest case user A's password is sent in the clear, which is of course not recommended. User A's password can be protected using a one-way function $f_{1()}$ in the following way:

Client $\rightarrow$ Server: t_1, r_1, A, Protected$_1$ = $f_1(t_1, r_1$, A, password$_A)$

t_1 is a timestamp, and r_1 a random number. They help detect replay attacks (i.e., ensure the freshness of messages). Timestamps should generally be used only if the client and the server clock are (at least logically) synchronized. A higher level of protection (i.e., lower chance of a successful birthday attack), can be achieved when using two one-way functions, f_1 and f_2, in the following way:

Client $\rightarrow$ Server: t_1, t_2, r_1, r_2, A, Protected$_2$ = f_2(t2, r2, Protected$_1$)

To perform this type of authentication, there is a corresponding data structure in the protocol, called `SimpleCredentials`, which is an optional part of the bind request or bind response message (the name field carries the user's identity, i.e., in our example it would contain A):

```
SimpleCredentials       :=    SEQUENCE {
   name                 [0]DistinguishedName ,
   validity             [1]SET {
      time1                [0]   UTCTime OPTIONAL,
      time2                [1]   UTCTime OPTIONAL,
      random1              [2]   BIT STRING OPTIONAL,
      random2              [3]   BIT STRING OPTIONAL} OPTIONAL,
   password             [2]CHOICE {
      unprotected          OCTET STRING,
      protected            SIGNATURE{OCTET STRING}} OPTIONAL}}
```

For strong authentication, X.509 recommends three protocols based on public key cryptography: one-, two-, and three-way authentication. In the one-way authentication protocol, only one party is authenticated and the freshness is achieved using timestamps and random numbers. $D_x(.)$ denotes encryption with x's private key (i.e., x's digital signature). S and C denote the server's and client's identifiers, respectively.

All necessary public keys for signature verification are obtained from X.509 certificates. [S→C] denotes a *certification path* from S to C. The certification path contains a *chain* of certificates. The first certificate in the chain, CA(S)<S>, cer-

tifies S's public key and is issued by a certification authority (CA) trusted by S. The last certificate in the chain, CA(C)<C>, certifies C's public key and is issued by a CA trusted by C. We can write as follows:

CA(S)<S>→CA(CA(S))<CA(S)>
→ CA(CA(CA(S))<CA (CA(S))>→ ...→ CA(C)<C>

Such a chain is often referred to as the *chain of trust*:

S→CA(S)→CA(CA(S))→ ... (CA(CA(...(S)))→CA(C)

We can read it as follows: S trusts CA(S), CA(S) trusts CA(CA(S)), ..., and some CA later in the chain finally trusts CA(C) which issued C's public key certificate.

Let D_x(message) denote message m with the digital signature by signatory x appended to it. The client sends to the server the certification path [S→C], and the signed message containing a timestamp t_c, a random number r_c, and the server's identity S:

Client → Server: [S→C], $D_c(t_c, r_c, S)$

The server uses the certification path to find the client's public key and verify its certificate. If the certificate is valid, and the public key can verify the signature, the client is successfully authenticated. If the server has to be authenticated as well, an additional message is sent (two-way authentication):

Server → Client: $D_s(t_s, r_s, C, r_c)$

Finally, if no timestamps are used, they are set to zero in the first two authentication messages, and one more message is sent (three-way authentication):

Client → Server: $D_c(r_s, S)$

`StrongCredentials` is the protocol data structure that can be used to perform strong authentication, that is, to convey the authentication parameters. It is an optional part of the bind request or bind response message:

```
StrongCredentials          :=      SET {
     certification-path    [0] CertificationPath OPTIONAL,
     bind-token            [1] Token,
     name                  [2] DistinguishedName OPTIONAL}

Token                      :=      SIGNED { SEQUENCE {
     algorithm             [0] AlgorithmIdentifier,
     name                  [1] DistinguishedName,
     time                  [2] UTCTime,
     random                [3] BIT STRING}}
```

The `CertificationPath` field carries the client's certificate. `name` is the distinguished name of the client. The fields of `Token` are used to carry the parameters from the authentication protocols described above. `AlgorithmIdentifier` denotes the public key algorithm used to digitally sign the message.

X.500 Establishing a Secure Session

The authentication messages can be used for exchanging the session key information (key transport or key agreement). The key information has to be encrypted with the receiving party's public key. In the data structure for strong credentials there is, however, no parameter that can be used to convey the key information. Entity authentication which does not result in a session key for the protection of the subsequent messages does not make much sense. This is not the case with either simple authentication, because it is not its purpose, or strong authentication, because the suggested procedures for exchanging the key information cannot be implemented in DAP in its present form. There is one more possibility for authentication called `externalProcedure`, that is, to use an external procedure to establish a secure session prior to bind request, as indicated by *Establishing a secure session* in Fig. 3. The choice of the external procedure is not imposed by the standard.

LDAPv3 Authentication Parameters

If you look at Fig. 4 you will get a general idea of LDAPv3 security. The bind request in LDAPv3 is not mandatory. The security parameters that can be sent along with the operations are not yet standardized (see later). The mandatory-to-implement authentication Internet drafts [12, 13, 18] had gone through last call and been sent to the Area Directors to be progressed as Proposed Standard RFCs as of July 1999.

The bind request is defined as follows:

```
BindRequest ::= [APPLICATION 0] SEQUENCE {
     version               INTEGER (1 .. 127),
     name                  LDAPDN,
     authentication        AuthenticationChoice }

AuthenticationChoice ::= CHOICE {
     simple                [0] OCTET STRING,
                                 -- 1 and 2 reserved
     sasl                  [3] SaslCredentials }

SaslCredentials ::= SEQUENCE {
     mechanism             LDAPString,
     credentials           OCTET STRING OPTIONAL }
```

The LDAPv3 bind request is similar to the DAP bind request. There are, however, some differences. LDAPv3 simple authentication is equivalent to DAP simple unprotected authentication. However, in the last version of the LDAPv3 authentication document [12] simple authentication is allowed to be used either in combination with the SASL DIGEST-MD5 mechanism or over a secure layer provided by TLS [5]. We describe how to use TLS with LDAPv3 later in the article.

Protected Password — Simple protected authentication can be used as the SASL DIGEST-MD5 mechanism [18]. SASL [19] is an authentication framework for connection-based protocols. DIGEST-MD5 provides password protection as well as integrity and limited confidentiality protection after an authentication exchange. Here we give a simplified description of the authentication exchange:

Server sends **`digest-challenge`** to client:
`random`$_S$, `quality_of_protection`, `supported_ ciphers`

Client sends **`digest-response`** to server:
`username`, `host`, `service`, `random`C, `count_randomC`, `response_value`

Server sends **`response-auth`** to client:
`response_value`

`quality_of_protection` indicates which authentication type is supported by the server. It can have the following values: "auth" for authentication, "auth-int" for authentication with integrity protection, and "auth-conf" for authentication with integrity protection and encryption. The value `supported_ciphers` represents a list of symmetric encryption algorithms the server supports (3-DES and DES are mandatory). `random`$_S$ and `random`$_C$ are random strings. A random string is usually referred to as a *nonce*, but we will use the same notation for easier readability. Note that the random values used here are not necessarily random numbers in the mathematical sense as were the random numbers used in an earlier section. Their freshness is crucial, however, for protection against replay attacks. `service` is in our case equal to LDAP.

response_value is computed by applying a cryptographic hash function h(.) (i.e., MD5) to the response value in the following way (simplified):

response_value = h(username, host, randomS, randomC, password)

whereby the hash function parameters are represented as strings and concatenated to produce a common hash function input. A cryptographic hash function has the property that it is easy to compute h(input), but that it is nearly impossible to determine the input given h(input). After the second step the server can validate the client's response. If it is correct — and it can be correct only if the client knows the correct password — the server sends response-auth to the client. Subsequent authentication, which is optional, is also supported: if the client wants to be authenticated again with the same server, it may send digest-response again, with the value $count\_random_C$ one greater than used in the last digest-response.

In this way the client is authenticated if it knows the secret password. The mechanism prevents chosen-plaintext attacks and is therefore stronger than CRAM-MD5, originally proposed for use with LDAP.

X.509-Like Strong Authentication — The proposal for strong LDAPv3 X.509-like authentication will most probably be rejected since the members of the LDAPEXT IETF Working Group are reluctant to adopt too much of the X.500 functionality. It defines an SASL authentication mechanism [11] based on X.509 strong authentication [7]. The credentials are defined based on the X.511 security parameters (discussed later) with some optional fields. Some of its advantages over the strong authentication mechanism within TLS (next section) are:

- No secure session: In many situations a secure session is not needed. In such situations the use of TLS provides unnecessary overhead and complexity. In this way the export problems due to the confidentiality functionality are avoided too.
- Proxy support: The proposal uses credentials based on which a proxy can authenticate the client and then pass them on to the server.

LDAPv3 Establishing A Secure Session

In this section we describe two mechanisms, TLS and DIGEST-MD5, for establishing a secure session. In this way a secure layer can be established so that the LDAPv3 protocol messages are exchanged over it, and are thus unaware of the protection.

TLS — A strong authentication mechanism mandatory for LDAPv3 is TLS [5]. TLS 1.0 is based on SSLv3 and provides strong entity authentication, data integrity, and data privacy. In the TLS protocol server authentication is mandatory, and client authentication is optional. If in LDAP client authentication is required, we actually have a mutual (bidirectional) entity authentication service. To use TLS, the name of the SASL mechanism in the LDAP BindRequest has to be EXTERNAL, and the credentials field has to be empty. The protocol scenario is the same as for external mechanisms in DAP, as shown in Fig. 3.

A TLS session must be established prior to sending a bind request. The client is authenticated by a strong mechanism if it supplies a certificate. The name under which the client binds to the server is the name from the client's certificate, so the name in the bind request, if present, must be identical to the name in the certificate. The TLS negotiation starts with an ExtendedRequest called Start TLS. The ExtendedRequest/Response mechanism is the inherent LDAPv3 extension mechanism which allows adding new operations to the protocol. It is also possible that the client does not provide a certificate during the TLS setup; in this case it is up to the server to decide whether the client may bind anonymously over TLS.

An LDAP *extended operation* allows clients to define additional operations with predefined syntaxes and semantics not available elsewhere in the protocol. In a later section we show the structure of an LDAP protocol message (LDAPMessage). In LDAPMessage the protocolOp field specifies the type of operation. If you want to extend the LDAP protocol with a new request or response type, you should set the value of this field to either extendedReq or extendedResp, respectively.

DIGEST-MD5 — If the client is authenticated with SASL DIGEST-MD5 (discussed above) it is possible to protect integrity and confidentiality of all messages exchanged during that session.

For session integrity protection, the subsequent messages between the client and the server are protected by a MAC appended to the message. The MAC is sometimes referred to as the *message digest*. To compute the MAC, the client and server must share a *common secret*, usually represented by the client's password. The MAC is computed by applying a cryptographic hash function h(.) in the following way (simplified):

MAC(message) = h(C, S, password, random, sequence number, message)

whereby the hash function parameters are represented as strings and concatenated to produce a common hash function input. A cryptographic hash function has the property that it is easy to compute the MAC, but nearly impossible to determine the hash function input given the MAC value. In other words, if an eavesdropper obtains the message and the MAC value, he cannot determine the password. Also, he cannot compute the valid MAC for a different message since he does not know the password. The MAC values change each time a message is sent due to the variable parameters (random and sequence number); in this way replay attacks are disabled.

For session integrity protection, the MAC is computed by applying the HMAC-MD5 [20] to the values of $random_S$, $random_C$, and other parameters from digest-response. For session confidentiality protection, the encryption key is computed as a function of similar parameters.

Summary

In Table 2 we give an overview of the security services that may be initiated through the *bind* operation in X.500 and LDAPv3. As you can see, they are mainly authentication services, but two additional security services may be initiated as well: session integrity and session confidentiality. They illustrate the main difference between the two entity authentication concepts:

- X.500 "limits" the effect of the entity authentication procedure to the *bind* operation (request and response). The reason is that X.500 provides protection which can be applied to each directory operation individually, as described later.
- LDAPv3 wants to protect not only the *bind* operation, but also the operations that may follow it within a session. There is a draft on protecting individual operations too (later section), but at the moment it is only considered an experimental concept and will not be standardized soon.

Protecting Directory Operations

In this section we describe the security services that can be applied to individual directory operations: data (or message) integrity, data authentication, and nonrepudiation of origin. The discussion about which directory services need which kind of protection can be found earlier.

X.500 Security Parameters

Another useful security feature in DAP is that the server can be required to sign the result of an operation (see *Operation request* in Fig. 3). The signing request can be sent as an optional part of the common arguments called SecurityParameters:

```
SecurityParameters    :=     SET {
    certification-path    [0] CertificationPath OPTIONAL,
    name                  [1] DistinguishedName OPTIONAL,
    time                  [2] UTCTime OPTIONAL,
    random                [3] BIT STRING OPTIONAL,
    target                [4] ProtectionRequest OPTIONAL}
ProtectionRequest     :=    INTEGER { none(0), signed(1) }
```

The CertificationPath field contains the signer's certificate. name is the distinguished name of the intended recipient. If the value of ProtectionRequest is set to signed, the server has to sign the response. An operation argument (part of a request) or operation result (part of a response) can be OPTIONAL. time is the timestamp specifying the intended expiry time for the validity of signed requests. random is a random number and can be used in both a signed request or signed response. The timestamp and random number enable the detection of reply attacks, similar to simple and strong authentication.

LDAPv3 Security Parameters

The status of the last signed LDAPv3 operations draft [19] will most probably be declared experimental. The draft describes a mechanism based on LDAP *controls*. They can be used to extend the existing LDAP operations. For example, an LDAP protocol message consists of the following fields:

```
LDAPMessage ::= SEQUENCE {
      messageID       MessageID,
      protocolOp      bindRequest,
      controls        [0] Controls OPTIONAL }
```

The messageID field must carry a unique value for the corresponding LDAP session. The protocolOp field specifies the type of operation (in our example it is bindRequest, but can be something else, e.g., searchRequest, searchResponse, etc.). The controls field may carry any type of extension information not specified in the LDAP protocol.

The signed operations draft defines three LDAPv3 controls named SignedOperation, DemandSignedResult, and SignedResult:

```
SignedOperation             ::= CHOICE {
      signbyServer          NULL,
      signatureIncluded           OCTET STRING }

DemandSignedResult   ::= LDAPSigType

LDAPSigType    ::= BOOLEAN

SignedResult                ::= CHOICE {
      signature                 OCTET STRING }
```

The first mechanism supported is *directory audit trail*. The submitter of an LDAP operation that changes the directory information can include the SignedOperation control including either a signature of the operation (signatureIncluded, an S/MIME multipart/signed message) or a request that the LDAP server sign the operation on behalf of the LDAP client. The server can accept the signed operation without verifying the signature since the client has no way of knowing what policies were followed in order to verify the signature. The audit trail is stored in the newly defined Changes attribute:

```
Changes              ::= CHOICE {
      sequenceNumber       [0]   INTEGER (0..maxInt),
      signedOperation      [1]   OCTET STRING }
```

signedOperation is a multipart/signed S/MIME message containing the directory operation and the signature. Subsequent sequence numbers indicate the sequence of changes

Security service	Security mechanism	X.500 bind mechanism	LDAPv3 bind mechanism
Anonymous access	No client authentication	No Credentials	No bind operation
	No client authentication Server authentication	–	SASL EXTERNAL (TLS ExtendedRequest) without client certificate
Simple client Authentication	Password	SimpleCredentials, password unprotected	SASL DIGEST-MD5 password protection
		SimpleCredentials, password protected	SASL EXTERNAL (TLS ExtendedRequest) password protection
Strong client/server authentication	Digital signature	StrongCredentials	[SASL X.509, draft]
			SASL EXTERNAL (TLS ExtendedRequest)
External authentication	Any	EXTERNAL	SASL EXTERNAL (TLS ExtendedRequest)
Session authenticity and integrity (implicit data authenticity and integrity)	MAC	EXTERNAL	SASL EXTERNAL (TLS ExtendedRequest)
			SASL DIGEST-MD5
Session confidentiality (implicit data confidentiality)	Encryption	EXTERNAL	SASL EXTERNAL (TLS ExtendedRequest)
			SASL DIGEST-MD5

■ **Table 2.** *Summary of X.500 and LDAPv3 bind security services.*

that have been made to the directory object.

The second mechanism allows the LDAP client to require that the *server sign the result* it returns. The client sends the `DemandSignedResult` control with `LDAPSigType` set to TRUE. If the server can and wishes to sign to operation, it returns a `SignedResult` control in addition to the normal result. The signature is formatted as an S/MIME pkcs-7/signature object specified in RFC2311.

Summary

Both X.500 and LDAPv3 provide a mechanism to protect individual directory operations. The LDAPv3 proposal is still considered experimental only. We hope it will soon be made a part of the standard.

Access Control

This section gives an overview of the access control model and mechanisms. The access control models in both X.500 and LDAPv3 are based on the *access control matrix*. In practice, the access control matrix is implemented in one of the following ways:

- The row-wise implementation is referred to as a *capability list*, where for each subject (`user`) there is a list of objects and the subject's access rights (or permissions) to each object.
- The column-wise implementation is referred to as an *access control list* (ACL) where for each object (`item`) there is a list of subjects that have access to it and the access rights granted to each individual subject.

X.500 Basic Access Control

If we take the example from Fig. 2, a client may be given specific *permissions* (or *rights*) for specified *items* (ACL) in the following ways:

- The entry CN = Vesna Hassler as a whole
- The attribute types stored in the entry (commonName, surname)
- The attribute types of specific attributes (e.g., commonName)
- The attribute type and value of specific attributes (e.g., commonName = Vesna Hassler)
- The attribute values stored in the entry (Vesna Hassler, Hassler)
- The attribute values of specific attributes (e.g., Vesna Hassler)

There are some other cases, including so-called operational attributes, but they are outside the scope of this tutorial. For more information see [8].

The client permissions refer to specific operations, as outlined earlier. A permission can be *granted* or *denied*. If a permission is granted, the client is allowed to perform a specified operation on a specified item (e.g., grantAdd, grantRead). If a permission is denied, the client is not allowed to do so (denyAdd, denyRead).

On the other hand, by using a capability list an item may be made accessible to:

- All users
- Only the users with the same DN (e.g., CN = Vesna Hassler only to Vesna Hassler)
- Only to specified users
- Only to a specified user group
- Only to a set of users whose DNs fall within the definition of a directory subtree

The mechanism specified in X.501 [8] called *basic access control* enables defining many different access control policies. The information about accessing directory information is specified as a data structure called ACIItem:

```
ACIItem              := SEQUENCE {
    identificationTag        DirectoryString {ub-tag},
    precedence               Precedence,
    authenticationLevel      AuthenticationLevel,
    itemOrUserFirst CHOICE {
        itemFirst        [0] SEQUENCE {
            protectedItems    ProtectedItems,
            itemPermissions SET OF ItemPermission},
        userFirst        [1] SEQUENCE {
            userClasses       UserClasses,
            userPermissions SET OF UserPermission}}}
```

`identificationTag` is used to identify a particular `ACIItem` for the purposes of protection and management. The value of `precedence`, an integer between 0 and 255, helps resolve conflicts if more than one `ACIItem` refer to the same piece of information. In such a case the highest precedence value will prevail over all other values.

All users allowed to gain specific access rights to `protectedItems` (e.g., entry, attribute type, attribute value) are listed in `itemPermissions`. This is an implementation of the ACL. All items that may be accessed in a specified way by the users listed in `userClasses` are listed in `userPermissions`. This is an implementation of the capability list. The permissions (or rights) for the basic access control mechanism are listed in Table 3.

The directory access protocol is not concerned with the enforcement of the access control rules. The information the access control decision relies on comes, however, from the authentication process. The value of `AuthenticationLevel` specifies the required level of authentication:

```
AuthenticationLevel     :=      CHOICE {
    basicLevels SEQUENCE {
        level   ENUMERATED {none(0) simple(1), strong(2)},
        localQualifier   INTEGER OPTIONAL},
    other       EXTERNAL}
```

The value of `level` corresponds to the authentication procedures described earlier in the article. `localQualifier` may be assigned to the client by the server according to the local policy. `other` is used for determining the authentication level if an external authentication mechanism is used.

LDAPv3 Access Control

Access control is still not a mandatory part of an LDAPv3 server. There are some proprietary solutions, such as for Netscape Directory Server. In [9] X.501 `ACIItem` is mentioned as one of the syntaxes defined for LDAP so far, but not a mandatory one. The LDAPv3 requirements for an ACL-based access control are specified in [14]. It seems that the X.501 access control model could satisfy those requirements, but a serious evaluation has not been done so far. The last proposal (as of July 1999) for an LDAPv3 access control scheme can be found in [15]. It describes the access control model and mechanisms for LDAP with the corresponding LDAP controls and one extended operation. An ACL contains the access control policy information controlling access to an item or collection of items. For each naming context (i.e., subtree) with a specific ACL mechanism, the subschema entry must contain the `aclMechanism` attribute defining the `aclMechanism` for the scope of that subschema entry.

LDAP controls provide a way to specify extension information [6]. It can be sent as part of a request, and apply only to that request without being saved. The LDAP ACI controls are:

- `getEffectiveAccess` (used with the *search* operation): The server looks up the rights for the returned directory operation for a subject based on the subject DN and returns that rights information.
- `specifyCredentials` (used with the *bind* operation): The client sends the credential he wants to be associated with the bind DN to determine his access permissions in subse-

quent LDAP operations.

Analogous to the first ACL control listed above, an ACL extended operation is defined for transmission of access control information to help with the management of access control information independent of other directory operations (`ldapGetEffectiveRights`).

There are two ACL mechanisms, or *rights families* — LDAPv3 and X500 — but X500 has not been defined yet. Other vendor-specific mechanisms can additionally be defined. Access rights can apply to an entire object, or to the attributes of the object. Four operations are defined for granting and denying the rights of a subject, and for replacing and deleting the ACL of an entry or attribute.

As you can see in Table 3, LDAPv3 introduces a new group of access rights to protect objects referred to by directory entries:

- *Manage* is used to restrict access to operations that read or write very sensitive data.
- *Execute* is used to control access to the executable objects referred to by directory entries.
- *Get* retrieves the attribute values from such objects.
- *Set* writes the attribute values from such objects.

The rationale to introduce the new rights is, however, still unexplained in [15], as well as their practical value.

In the draft the subject identity is in the form of a DN. Security protocols being specified in the IETF Common Authentication Technology (CAT) Working Groups and other working groups use a more generic format (i.e., mechanism + mechanism-specific identity object). The approach from the LDAP ACL proposal is to have the clients ask the server to map its authentication mechanism-specific identity onto a DN; the client can then use this DN in the access control operations.

A common representation of access control information is needed for replication. Replication means that a server is keeping a copy of its data on one or more other servers. ACI representation can be defined in either the replication protocol being developed in LDAP Directory Update Protocol (LDUP) or LDAP Data Interchange Format (LDIF) files, which do not exchange operations but the entries themselves. The LDAP ACL draft specifies the LDIF syntax for ACI. The draft does not define how a server should interpret the ACI syntax. For example, deny on one server may be different from deny on another server. The ACI is stored in the newly defined directory attributes (discussed later).

Summary

Table 3 shows the LDAPv3 and X.500 access rights.

X.500 defines which access rights must be checked and granted for which operation (operational semantics [8]). Here are some examples:

- To replace a value of an attribute the *modify* operation can be used (Table 1). For this to succeed the client must be granted the rights to *remove* the old value and *add* a new one. For this purpose LDAPv3 can use the right to *write* a value.
- To search the directory, in X.500 the client must have the right to *browse* the entries and *match* them against a specified *filter*. These rights are covered in LDAPv3 by the *search* right.
- To change a DN, in X.500 the client must have the right to *rename* the entry. Additionally, if the entry's superior node changes, the client must be granted the right to *export* the entry and *import* it under the new superior entry. Obviously, in LDAPv3 there is only one corresponding right for this operation, the right to *edit* the DN.

LDAPv3 servers are free to implement any access control rule syntax and semantics they choose, as long as the operational semantics is compatible with the LDAP access control operations. The operational semantics is, however, work in progress.

Here we point out some additional differences between the LDAPv3 and X.500 access control models [21]:

- Precedence — In contrast to X.500, there are no precedences.
- Authentication — LDAPv3 does not indicate the type of authentication users must undergo before access can be granted to them.
- ACL protection — In LDAPv3 no mechanism is defined to control access to access control information. In X.500 access control information is held as operational attributes and therefore

Protected item		Access rights	
LDAPv3	X.500 basic	LDAPv3	X.500 basic
Attribute values	allUserAttributeTypes	Read	Read
	attributeType	Write	–
	allAttributeValues	Search	–
	allUserAttributeTypesAndValues	Compare	Compare
	attributeValue	–	Add
	selfValue	–	DiscloseOnError
		–	Remove
		–	FilterMatch
Object (Entry)	Entry	Add	Add
		Delete	Remove
		–	Read
		–	DiscloseOnError
		–	Browse
		–	Export
		–	Import
		–	Modify
		EditDN	Rename
		–	ReturnDN
Object pointed to by directory object	–	Manage	–
		Use	–
		Get	–
		Set	–

■ **Table 3.** *Access rights in X.500 basic access control and LDAPv3.*

can be used to protect themselves.

- ACL interaction — The LDAP access control administration mechanisms are neutral regardin policy inheritance mechanisms, explicit vs. implicit denial, and group nesting. In X.500 the ACLs that actually apply to an entry are derived from the ACLs that are stored within the EntryACI and PrescriptiveACI attributes from all the enclosing access control domains.
- Representation of ACI — LDAPv3 servers may store access control information in any way they choose. In X.500 ACLs are held in predefined operational attributes.

Security service	X.500 mechanism	LDAPv3 mechanism
Entity authentication	No, simple, strong, and external authentication	Protected password and X.509- like
Data authentication	Security parameters	X.509-like or signed operations
Data integrity	Security parameters	X.509-like or signed operations
Data confidentiality	No explicit protection	No explicit protection
Nonrepudiation of origin	Security parameters	X.509-like or signed operations
Secure session (implicit data integrity, authentication, and, if requested, confidentiality)	External procedure	TLS or DIGEST-MD5
Access control	Basic access control	LDAPv3

■ **Table 4.** *Security services with X.500 and LDAPv3.*

Directory Attributes

In this final section we give a list of all security-relevant *directory attributes*. They can be stored in the directory just like any other data. For example, if somebody receives a digitally signed document he will need the signatory's public key certificate. The certificate can be stored in the directory server as a special attribute type for which the client can look. The attributes storing certificates or the attributes needed for interoperability, such as `supportedAlgorithms` that specifies the cryptographic algorithms supported by the server, are made readable to anybody. Some attributes are confidential, such as `userPassword` that must not be stored in plaintext. Finally, some attributes (called *operational*) serve administrative purposes, such as administering the access control information.

X.500 Directory Attributes

The parameters required for strong authentication are stored in an X.500 server as user attributes of the following types [7]:

- `userPassword`
- `userCertificate` — User (client or server) public key certificate
- `cACertificate` — CA public-key certificate
- `certificateRevocationList` — Certificate revocation list of user certificates; it contains the certificates declared invalid although not expired, so the public keys they certify must not be used to verify digital signatures any longer
- `authorityRevocationList` — Certificate revocation list of CA certificates
- `deltaRevocationList` — Changes in the last CRL published; this attribute is convenient if the CRL becomes large
- `crossCertificatePair` — a pair of certificates with which two parties mutually certify each other
- `attributeCertificateAttribute` — user attribute certificate; similarly to public-keys, other data (e.g., procuration) may be certified as well
- `attributeCertificateRevocationList` — CRL of attribute certificates
- `supportedAlgorithms` — Defined to support the selection of an algorithm for use when communicating with a remote end entity using certificates (an overview of algorithms and their object identifiers can be found in [22])

Access control information is stored in operational attributes `entryACI`, `prescriptiveACI`, and `subentryACI` which are normally not visible to the client. They control access to a directory entry, a set of entries in a directory subtree, and a set of subentries (holding the administrative information) in an administrative area, respectively. There is an additional attribute, `accessControlScheme`; its value indicates the type of the access control scheme (e.g., basic access control [8]).

LDAPv3 Directory Attributes

If an LDAPv3 server stores the X.500 security attributes, they have to be stored as binary values because of the changes in the ASN.1 definition in various X.509 editions. An additional security attribute from the LDAPv3 user schema [10] is `supportedSASLmechanisms` and is stored in the DIT root (i.e., in the so-called DSA-specific entry, DSE). The `signedDirectoryOperationSupport` and `changes` attributes are required for signed operations (see earlier section). The `supportedExtension` attribute is important for the TLS authentication since it is defined as an extended operation (discussed earlier).

The LDAP ACL draft [15] defines a new attribute containing a list of access control mechanisms supported by the server (`supportedACIMechanisms`) to be stored in the DIT root. It also defines the aCIMechanism attribute that must be stored in each subschema entry containing the identifier of the access control mechanism in effect for the scope of that subschema entry. Finally, it defines three new attributes that can be added to any object class (i.e., type of entry, see above). Their purpose is to store the access control information (ACI) in the following ways:

- `aci` defines what is protected and who is granted/denied what kind of permissions
- `vendorAci` is used for vendors-specific information with a syntax different from `aci`
- `policyOwner` determines who controls administrative subdomains (e.g., subtree), and who can set or change ACI for implementations that have no ACI controlling access to it

Summary

Most of the security-related attributes in LDAPv3 are adopted from X.509. The LDAPv3 offers, however, only the storage for these attributes, and is not concerned with their semantics, whereas X.509 is a functional part of the X.500 global directory service.

Summary and Conclusions

In Table 4 we give a summary and roadmap of the security services and mechanisms described in the article. The LDAPv3 security functionality has not yet been completely specified. Entity authentication, including secure session, is completed, with pending Proposed Standard status. Other security services necessary for the operation of a directory service are work in progress. The basic data structures are the same as in X.509, such as the certificate format. However, the attempts to adopt some of the X.500 access protocol's security solutions for LDAPv3 have all failed. One reason is that in the Internet

some solutions have already become standards, such as TLS, recently been adopted as a Proposed Standard, and LDAPv3 wants to "outsource" security whenever possible. Another reason is that the X.500 standards do not provide sufficient information to ensure interoperability in some cases — see, for example, the authentication procedures discussed in this article — and therefore cannot be adopted without additions, and sometimes even modifications. LDAPv3 is missing an explicit message authentication and integrity mechanism (e.g., other than within TLS) that would allow the complex TLS setup to be avoided when necessary. This functionality could be provided by the signed operations concept described in [16] which is unfortunately not likely to be adopted as a standard soon. The access control model and operational semantics in LDAPv3 are work in progress.

References

The Internet drafts listed below represent work in progress. They expire in six months unless replaced with a new version. Their status can be checked online at http://www.ietf.org. The RFC documents can also be downloaded from this Web site.

[1] D. Chadwick, *Understanding X.500 — The Directory*, Chapman & Hall, 1994; available at http://www.eema.org/understanding_ldap.html
[2] C. Apple and K. Rossen, "X.500 Implementations Catalog-96," RFC 2116.
[3] ISO, "Information Technology — Open Systems Interconnection - The Directory: Abstract Service Definition," ISO/IEC 9594-3 (also ITU-T X.511), 1995.
[4] Critical Angle's LDAP World, http://www.critical-angle.com/ldapworld/
[5] T. Dierks and C. Allen, "The TLS Protocol Version 1.0," RFC 2246.
[6] M. Wahl, T. Howes, and S. Kille, "Lightweight Directory Access Protocol (v3)," RFC 2251.
[7] ISO, "Information Technology — Open Systems Interconnection — The Directory: Authentication Framework," ISO/IEC 9594-8 (also ITU-T X.509), 1995.
[8] ISO, "Information Technology — Open Systems Interconnection — The Directory: Models," ISO/IEC 9594-2 (also ITU-T X.501), 1995.
[9] M. Wahl *et al.*, "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions," RFC 2252.
[10] M. Wahl, "A Summary of the X.500(96) User Schema for Use with LDAPv3," RFC 2256.
[11] S. E. Kille, "X.509 Authentication SASL Mechanism," Interne draft draft-ietf-ldapext-x509-sasl-01.txt
[12] M. Wahl and H. Alvestrand, "Authentication Methods for LDAP," Internet draft draft-ietf-ldapext-authmeth-04.txt
[13] J. Hodges, R. L. Morgan, and M. Wahl, "Lightweight Directory Access Protocol (v3): Extension for Transport Layer Security," Internet draft draft-ietf-asid-ldapv3-tls-05.txt
[14] E. Stokes et al., "Access Control Requirements for LDAP," Internet draft draft-ietf-ldapext-acl-reqts-01.txt
[15] B. Blakley, D. Byrne, and E. Stokes, "Access Control Model for LDAP," Internet draft draft-ietf-ldapext-acl-model-03.txt.
[16] B. Greenblat and P. Richard, "Signed Directory Operations Using S/MIME," Internet draft draft-ietf-ldapext-sigops-03.txt.
[17] T. A. Howes and M.C. Smith, *LDAP. Programming Directory-Enabled Applications with Lightweight Directory Access Protocol*, Macmillan, 1997.
[18] P. J. Leach and C. Newman, "Using Digest Authentication as a SASL Mechanism," Internet draft draft-leach-digest-sasl-03.txt.
[19] J. Myers, "Simple Authentication and Security Layer (SASL)," RFC 2222.
[20] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," RFC 2104.
[21] B. Blakley, D. Byrne, and E. Stokes, "X.500 vs. LDAP ACL Model Proposal comparison," IETF-LDAPEX mailing list archive ftp://ftp.innosoft.com/anon/files/ietf-ldapext/, June 15, 1998.
[22] German National Research Center for Information Technology GmbH, "SECUDE-5.1 Hyper Link Documentation: Cryptographic Algorithms," http://www.darmstadt.gmd.de/secude/Doc/htm/algs.htm

Biography

VESNA HASSLER (hassler@infosys.tuwien.ac.at) received B.Sc. and M.Sc. degrees in electrical engineering and computing from Zagreb University, Croatia, in 1988 and 1991, respectively. In 1995 she received a Ph.D. in computer engineering and communications from Graz University of Technology, Austria. She is currently an assistant professor in the Information Systems Institute, Technical University of Vienna, Austria. Her current interests include network security, directory services, e-commerce, and smart cards.