CHAPTER 3

Defining Directory Service Security Architecture

This chapter discusses client-server directory service architectures and describes what you can and cannot do to secure data transfers and authentication. The focus is on the Secured LDAP Client, which is a core and integral component of the Solaris 9 Operating Environment.

This chapter starts by discussing the Sun ONE Directory Server software security features such as access control and authentication mechanisms, in particular SASL DIGEST-MD5 and the Generic Security Services Application Programming Interface (GSSAPI) authentication mechanisms, followed by Transport Layer Security (TLS), and the Start TLS functionality. The server side is discussed from a system administration and developer point of view. The final part of this chapter describes the PAM components and modules.

This chapter is organized into the following sections:

Understanding Directory Server Security

The Sun ONE Directory Server 5.2 software provides the foundation for the new generation of e-business applications and services.

Based on a highly advanced, carrier-grade architecture, the Sun ONE Directory Server software delivers a high-performance, highly scalable, and highly secure infrastructure that provides organizations with a secure directory service implementation.

Sun ONE Directory Server 5.2 Software Security Features

This section describes the following:

Access Control

One of the primary reasons for using an access control mechanism is to control and restrict access to information and to control the operations that can be performed by users and administrators of the directory server. Operations to control access to the directory server include the ability to restrict permissions for adding, deleting, and modifying directory entries.

Accessing the directory service requires that the directory client authenticate itself to the directory service. This means that the directory client must inform the directory server who is going to be accessing the directory data so that the directory server can determine what the directory client is allowed to view and what operations can be performed. A directory client first authenticates itself and then performs operations. The server decides if the client is allowed to perform the operation or not. This process is known as *access control*.

Prior to the release of the Sun ONE Directory Server 5.2 software, when a directory (LDAP) client or directory (LDAP) application authenticated to the directory server, the directory server would determine whether or not the directory (LDAP) client or directory (LDAP) application was in fact allowed to perform such operations (such as add, delete, or modify a particular directory entry).

Additional Security Features

The Sun ONE Directory Server 5.2 software provides additional security functionality. The following is an introduction to this new functionality.

- `GetEffectiveRights` - In addition to the access control framework that is currently used within the Sun ONE Directory Server 5.2 software version, a new feature has been added called `GetEffectiveRights` that addresses various needs. The `GetEffectiveRights` mechanism is used by clients to evaluate existing access control instructions (ACIs) and to report the effective rights that they grant for a given user on a given entry. The `GetEffectiveRights` feature is useful for various reasons:
  - Provides ACI management functionality that is used to verify that the current ACI's are really offering the intended access rights.
  - Aids the administration of users, and retrieves their rights to directory entries and attributes. However, note that though it can be used to determine if an operation would succeed or fail, it cannot be used to determine if an operation was successful.
  - Enables verification of the access control policy. You can retrieve the permissions list for a user on a given entry and its attributes.
  - Enables administrators to debug access control issues.
  - Allows directory-enabled applications to easily determine whether a user has permission to perform a particular operation (for example, not give the user the option to delete an entry if they don't actually have permission to delete it).
- Encrypted Attributes – Data in any directory service, needs to be protected. The Sun ONE Directory Server 5.2 software has many different ways and mechanisms for protecting access to directory data. In the context of *attribute encryption*, this feature is designed to provide data privacy or protection of physical access to data such as LDIF files, backup files, and database files. Thus, attribute encryption allows you to specify that certain attributes will be stored in an encrypted form. This feature is configured at a database level, and once you decide you want to encrypt an attribute, that particular attribute will be encrypted for every entry in the database.
- Start TLS – Start Transport Layer Security (Start TLS) is a LDAPv3 extended operation plug-in in the Sun ONE Directory Server software. This operation provides for TLS establishment in an LDAP association, which allows the client to initiate an encrypted connection over an existing (or opened LDAP connection.

Note – Start TLS now available on many platforms including the Windows platform.

- Scoped Password Policy – A password policy is a set of rules that control how passwords are used in the Sun ONE Directory Server software. To improve the security and make it difficult for password-cracking programs to break into the

directory, it is desirable to enforce a set of rules on password usage. These rules are made to ensure that the users change their passwords periodically, the new password meets construction requirements, the reuse of old passwords is restricted, and to lock out users after a certain number of bad password attempts. In earlier versions of the directory server software, the password policy was limited in its functionality to one global policy for the entire directory. This limitation no longer exists in the Sun ONE Directory Server 5.2 software, which offers increased flexibility in that you can configure multiple password policies.

- Dynamically Loadable SASL Library – The Simple Authentication and Security Layer (SASL) is a generic mechanism for providing authentication and, optionally, integrity and privacy support to connection-based protocols. In previous releases of the directory server software, there were two ways of thinking of how you could add a SASL security mechanism. One mechanism was to write a server plug-in that implemented the SASL mechanism in terms of a pre-bind operation. The other mechanism was to write a SASL mechanism plug-in that would be loaded by the SASL library itself. With this in mind, a shared library, `libsasl`, and associated plug-ins (GSSAPI, DIGEST-MD5, and CRAM-MD5) have been developed for both the Sun ONE Directory Server 5.2 software and Solaris OE. However, at the present time, the dynamically loadable SASL library is private to the Sun ONE Directory Server 5.2 software. When the integrated version of `libsasl` is introduced on the Solaris OE, the Dynamically Loadable SASL mechanisms will be supported. However, due to U.S. government regulations, there will only be support for authentication, but not encryption.

Authentication Mechanisms

This section discusses what authentication mechanisms are currently available, and how these authentication mechanisms can be used by directory (LDAP) clients.

The LDAPv3 standard which defines the LDAPv3 protocol was published in 1997, and originally proposed different mechanisms that could be used by directory (LDAP) clients to authenticate to directory (LDAP) servers (RFC 2251). RFC 3377 “The LDAPv3 Technical Specification” was published to list all RFC’s that comprise the full specifications of LDAPv3. That is, RFC 2251-2256, RFC 2829 (authentication methods) and RFC 2830 (Extension for TLS). The Sun ONE Directory Server 5.2 software conforms to the LDAPv3 Technical Specification.

There are several authentication methods that can be used to authenticate to a LDAPv3 directory server:

- None, no authentication, also known as anonymous – When using this method of authentication, a directory client will not be able to, or is not intended to, perform specific LDAP operations, such as modifications to directory entries or access to sensitive information. Using this method means that a directory client which has not authenticated or which has authenticated with its name but no password is anonymously authenticated. A client which failed to authenticate is not

authenticated as anonymous but the following operations will be considered as anonymous. In addition to a client being unauthenticated by default until a successful bind has been performed, an anonymous bind can be performed by using Simple authentication with no password (and typically no DN) as per RFC 2251 section 4.2.2.

Note – While it is true that all directory server (LDAPv3) implementations must support anonymous authentication because LDAPv3 does not require a bind as the first operation, it is perfectly legal for a directory server to be configured in such a way that it rejects any attempt to perform an operation without first authenticating to the server.

- Simple, password-based authentication – When this method of authentication is used, the DN (distinguished name) and password are sent over the network in clear text (not encrypted). It should be noted however, that even with the inherent security vulnerabilities, it is possible to use Simple authentication with transport-layer security (like TLSv1/SSL or IPSec) in a secure manner.
- SASL authentication mechanisms – The Simple Authentication and Security Layer (SASL) is a specification and method used by the LDAPv3 protocol to support what is known as pluggable authentication. This mechanism is used by the directory server (LDAPv3) and directory client (LDAPv3) to identify the user, authenticate this user to the directory server (LDAPv3), and finally to negotiate an optional security layer for subsequent protocol interactions. The SASL (RFC 2222) mechanism is covered in more detail later in this chapter.

Note – The LDAP v2 protocol does not support the Simple Authentication and Security Layer (SASL).

- Certificate-based authentication – Using this method, it is possible with the Sun ONE Directory Server 5.2 software to require that when the client connects to the directory server, the client provides a digital certificate to the directory server as identification. Authenticating a client using a client certificate really falls under the SASL category because a client certificate will only be used to authenticate the client if that client performs a bind operation using the SASL EXTERNAL mechanism.

Understanding the SASL Mechanism

This section explains what the Simple authentication security layer (SASL) is, how this is implemented in Sun ONE Directory Server 5.2 software, and what authentication methods it supports.

The Simple Authentication and Security Layer (RFC 2222) is an Internet Specification (DRAFT standard like LDAPv3) and method for adding authentication support with an optional security layer to connection-based protocols, such as LDAPv3. SASL also describes a structure for authentication mechanisms. The result is an abstraction layer between protocols and authentication mechanisms such that any SASL-compatible authentication mechanism can be used with any SASL-compatible protocol.

Before going into any more detail, let's take a brief look at why SASL is so important. Before SASL was introduced, what happened when a new protocol was written that required authentication? The answer is similar to that of the Solaris Operating Environment before the Pluggable Authentication Module (PAM) feature was introduced. Developers of the protocol had to explicitly allow and define the individual authentication mechanism. You ended up with a protocol that was developed in such a way that it had a particular way of handling how a CRAM-MD5 login was handled, a particular way of handling how a Kerberos v4 login was handled, and so on.

One of the biggest concerns of this model was when a new authentication method was developed and the protocol needed to be modified to support this particular authentication mechanism. This led to a lengthy process before the new authentication mechanisms could be released, and if your application used more than one protocol, for example, an email client, the developer was required to support CRAM MD5 for IMAP and CRAM-MD5 for POP, which would potentially create many different authentication mechanisms to implement and support. This process, of course, was not desirable for the protocol or application developer in an ever-changing environment where new authentication mechanisms are always being developed.

What was needed was a mechanism whereby developers could simply have one framework to write to. This is where the Simple Authentication Security Layer (SASL) which is described in RFC 2222 comes in to its own and addresses some, but not all, of the above issues. As an example, not all forms of SASL mechanisms can be handled by simply linking with some external library. DIGEST-MD5 authentication is a very good example of this because while you can use an external library to handle all the negotiation and the work of verifying the password, it is necessary to establish some mapping between the identity provided by the user and an account in the directory server. SASL EXTERNAL is an even better example because in many cases it has to be handled entirely by the server.

The Simple Authentication and Security Layer (SASL), is a generic mechanism and framework for protocols to accomplish authentication. Applications such as the Sun ONE Directory Server software and Solaris OE Secured LDAP Clients use the SASL library as a means of informing the application how to accomplish the SASL protocol exchange, and what the results are.

SASL is a framework whereby SASL authentication mechanisms control the exact protocol exchange. For example, if you have two protocols (such as IMAP and POP) and a number of different ways of authenticating, SASL attempts to make it so that only *n* plus *m* different specifications need be written instead of *n* times *m* different specifications. With the Sun SASL library, the mechanisms need only be written once, and they'll work with all servers that use them.

FIGURE 3-1 shows the SASL components.

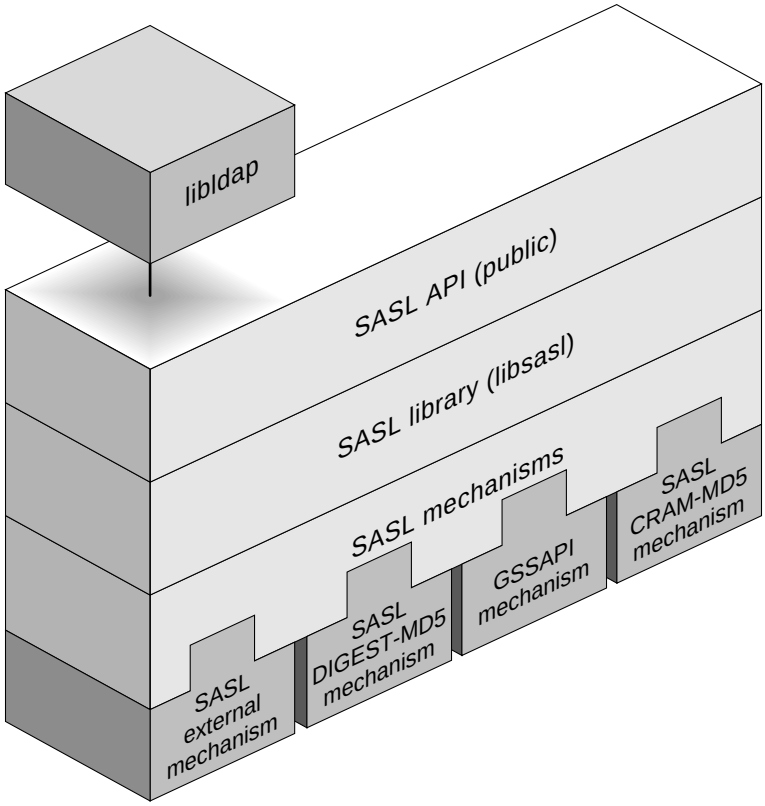

FIGURE 3-1 SASL Components

SASL DIGEST-MD5

The basic idea of the SASL DIGEST-MD5 mechanism is that both the client and server have a shared secret which is namely the user's password. Each side uses this secret together with `nonces` (defined on page 46) to prove to the other side that they do indeed know this shared secret without revealing the secret to the other side. If the client uses the wrong password, the server detects it. Similarly, if the server doesn't know the secret, the client detects that.

The Sun ONE Directory Server version 5.2 software integrates a new SASL library (`libsasl`), which is based on the Cyrus CMU implementation. Through this SASL framework, the directory server supports DIGEST-MD5 and the GSSAPI (which implements Kerberos v5).

Note – Currently the GSSAPI is only supported on the Solaris OE.

The SASL security feature is configurable through LDAP and is accessible through the entry `cn=sasl`, `cn=security`, `cn=config`. Using this entry, you can enable authentication mechanisms and also update the path where the SASL authentication mechanism is loaded by `libsasl`.

To solve the issue of mapping protocol identities to LDAP identities, there is a feature called *identity mapping* in the Sun ONE Directory Server 5.2 software. This feature maps from specific authentication protocols, such as DIGEST-MD5, GSSAPI, and HTTP, to an LDAP identity by applying mapping rules that are entirely customizable through LDAP. By default, there is an identity mapping for DIGEST-MD5 in the case where the client passes a dn as `authid`. It is possible to define as many mappings as you want. The identity mapping feature is described in the Sun ONE Directory Server 5.2 software documentation.

One of the important things to remember is that there is an authentication ID and an authorization (or proxy) ID. SASL also refers to the authorization ID as the user name. In the case when there is no proxy, the two identities are the same. Currently, the Sun ONE Directory Server 5.2 software does not support proxies through SASL.

The LDAPv3 Technical Specification (particularly RFC 2829), mandates the support of SASL DIGEST-MD5. Authenticating clients using the Digest authentication mechanism does not provide a strong authentication mechanism when compared to public key based mechanisms, but does prevent the much weaker and even more dangerous use of plain text passwords. In addition, the DIGEST-MD5 offers no confidentiality protection beyond protecting the actual password during the authentication phase. This means that the rest of the challenge and response, including the user's name and authentication realm, are available to an eavesdropper. However, the DIGEST-MD5 method can be used to provide integrity and confidentiality on the connection after the authentication process has been completed.

The MD5 message-digest algorithm is primarily used in three areas of the Solaris 9 Operating Environment. The Secured LDAP Client, the kernel (TCP and IPsec), and the User (SLP and PPP). Ronald Rivest, who was at the time working at the Laboratory for Computer Science at the Massachusetts Institute of Technology, published MD5 as an RFC (RFC 1321) in April 1992.

When you send data over the wire, you are concerned with three general issues: security, authenticity, and integrity. The *security* of your data ensures that no one else can read your data. This issue is important in many organizations that have information that cannot be exposed to external sources. *Authenticity* guarantees knowledge of the originator of the data; in other words, where the data source is from. This issue is important in areas such as the legal world where authentication issues (like digital signatures) are of great importance. Lastly, *integrity* guarantees that the data has not been tampered with in any way when it was transmitted, thus determining whether the data you received was the same data that was actually sent.

It is also very important to understand that MD5 hashing only guarantees the integrity of the data if it is possible to guarantee the integrity and/or authenticity of the MD5 digest itself. That is, if you use an MD5 digest to guarantee the integrity of a file, you should not store the digest with the file because if the file data is compromised it is very easy for a new MD5 digest to be generated. Alternately (and this is the way that DIGEST-MD5 works), all or part of the digest can include some shared secret that is known only by the originator and valid recipients, but not by untrusted third parties.

The MD5 algorithm guarantees the integrity of your data by taking a bit pattern of arbitrary but finite length and producing a 128-bit *fingerprint* or *message* digest of that pattern. This pattern is always 128-bit, regardless of the length of the bit pattern. It is extremely unlikely for two different files to produce the same fingerprint, but because an MD5 hash only consists of 128 bits, there are theoretically an infinite number of bit sequences that when hashed using MD5 will produce the same digest. The MD5 algorithm is not complex and does not require large substitution tables. Security experts estimate that the difficulty of finding two-bit patterns having the same digest is 2^{64} operations, and the difficulty of finding a bit pattern having predetermined digest is 2^{128} operations. It is computationally possible to determine a file based on its fingerprint, but it is not feasible based on current technology and techniques. This means that it is not possible for someone to figure out your data based on its MD5 fingerprint. Before we look at an example, we need to be aware that the Solaris 9 OE does not ship with the MD5 binary.

Take a look at an example of the output produced by MD5 on the binary file `/usr/bin/crypt`.

```
# md5-sparc /usr/bin/crypt
```

You should see output similar to the following:

```
MD5 (/usr/bin/crypt) = c54740de32d1903b78322a9be712a31d
```

In particular, the string `c54740de32d1903b78322a9be712a31d` is the fingerprint of `/usr/bin/crypt`:.

What happened in the above example is that the MD5 message-digest algorithm applied a mathematical algorithm to the binary `crypt` and produced the fingerprint. What you see is that you get the exact same fingerprint; if you do not, then you know that the binary has been altered in some way. Finally, since MD5 does not encrypt data, it is not restricted by any exportation laws, so you can distribute this tool freely anywhere in the world.

Note – In the Solaris 9 OE, there is no `md5` binary. What you will find is the `/platform/sun4u/kernel/misc/md5` and `/platform/sun4u/kernel/misc/sparcv9/md5` kernel modules, which export the standard MD5 calls to a user program. To obtain the `md5` binary, download the following file:
`http://sunsolve.Sun.COM/md5/md5.tar.Z`

Now that we have taken a look into what the MD5 message-digest is and how it works, it is time to apply this to how this actually translates and works in the Sun ONE Directory Server 5.2 software environment. This section introduces a new element that is discussed later in this chapter and called the *Secured LDAP Client*, which supports, and thus can be used to authenticate using DIGEST-MD5, to the Sun ONE Directory Server 5.2 software.

The following process describes how the Secured LDAP Client authenticates to the Sun ONE Directory Server 5.2 software using the SASL DIGEST-MD5 authentication mechanism.

1. Initial Authentication

   This process starts with the Secured LDAP Client sending a bind request with either the SIMPLE or SASL method. In this context, if it's SASL, the DIGEST-MD5 mechanism is specified. The DIGEST-MD5 authentication is a two-step bind operation.

   The Secured LDAP Client issues a SASL DIGEST bind request, as well as requesting a SASL DIGEST-MD5 bind. While it is possible to specify the bind DN in the initial request, the DN should not be sent, but if it is, if should be ignored. Rather, the authorization ID (which may be a DN, but may also be basically any other kind of ID provided that it can be uniquely resolved to a user entry using the directory server's identity mapping API) is provided by the client in the second stage of the bind request.

In this process, both sides can compute a shared secret, A1. A1 consists of a hash of the `username`, realm and password, which is concatenated with the directory server `nonce` and the directory client `nonce`. It is assumed that both the Secured LDAP client and directory server can obtain this hash, given the `username` and realm.

Note – Although the directory server does need to have access to the clear text password in order for the client to use DIGEST-MD5 authentication, the attribute encryption feature of the Sun ONE Directory Server 5.2 software can at least somewhat mitigate the risk of having clear text passwords in the server by ensuring that they are not stored on disk in clear text (and therefore would not show up clearly in most backups or LDIF exports).

---

2. Digest Challenge Stage

   The directory server starts by sending a challenge, whereby the data encoded in the challenge contains a string formatted according to the rules for a `digest-challenge` which is shown in the following example:

```
# From RFC 2831 "Digest SASL Mechanism”
digest-challenge  =
       1#( realm | nonce | qop-options | stale | maxbuf | charset
algorithm | cipher-opts | auth-param )

        realm             = "realm" "=" <"> realm-value <">
        realm-value       = qdstr-val
        nonce             = "nonce" "=" <"> nonce-value <">
        nonce-value       = qdstr-val
        qop-options       = "qop" "=" <"> qop-list <">
        qop-list          = 1#qop-value
        qop-value         = "auth" | "auth-int" | "auth-conf" |
                             token
        stale             = "stale" "=" "true"
        maxbuf            = "maxbuf" "=" maxbuf-value
        maxbuf-value      = 1*DIGIT
        charset           = "charset" "=" "utf-8"
        algorithm         = "algorithm" "=" "md5-sess"
        cipher-opts       = "cipher" "=" <"> 1#cipher-value <">
        cipher-value      = "3des" | "des" | "rc4-40" | "rc4" |
                            "rc4-56" | token
        auth-param        = token "=" ( token | quoted-string )
```

   As you can see in this example, there are various directives and values declared. The following is a short description of what each directive value means:

- `realm` – A string which enables users to know which user name and password to use, in the event that they have different ones for different servers. Conceptually, it is the name of a collection of accounts that might include the user's account. This string should contain at least the name of the host performing the authentication and might additionally indicate the collection of users who might have access. An example might be `registered_users@gotham.news.example.com`. This directive is optional; if not present, the client should solicit it from the user or be able to compute a default. A plausible default might be the realm supplied by the user when they logged in to the client system. Multiple realm directives are allowed, in which case the user or client must choose one as the realm for which to supply to `username` and password.

Note – In the Sun ONE Directory Server 5.2 software, the realm is always the FQDN host name of the server that appears in the value of the `nsslapd-localhost` attribute in the `cn=config` entry.

- `nonce` – The server's nonce is a random 32-byte (256-bit) random number, which is platform dependent. The server-specified data string which must be different each time a `digest-challenge` is sent as part of initial authentication. It is recommended that this string be base64 or hexadecimal data. Because the string is passed as a quoted string, the double-quote character is not allowed unless escaped. The contents of the `nonce` are implementation dependent. The security of the implementation depends on a good choice. It is recommended that it contain at least 64 bits of entropy. The `nonce` is opaque to the client. This directive is required and must appear exactly once; if not present, or if multiple instances are present, the client should abort the authentication exchange.
- `qop-options` – A quoted string of one or more tokens indicating the *quality of protection* values supported by the server. The value `auth` indicates authentication; the value `auth-int` indicates authentication with integrity protection; the value `auth-conf` indicates authentication with integrity protection and encryption. This directive is optional; if not present it defaults to `auth`. The client must ignore unrecognized options; if the client recognizes no option, it should abort the authentication exchange.

Note – At present, the Sun ONE Directory Server software only supports `auth` `qop`.

- `stale` – The `stale` directive is not used in initial authentication. This directive may appear at most once; if multiple instances are present, the client should abort the authentication exchange.

Note – At present, the Sun ONE Directory Server software does not support `reauth`, thus the server will not report a reauthentication stale because of a `nonce` timeout.

- `maxbuf` – A number indicating the size of the largest buffer the server is able to receive when using `auth-int` or `auth-conf`. If this directive is missing, the default value is `65536`. This directive may appear at most once; if multiple instances are present, the client should abort the authentication exchange.

Note – `maxbuf` is only used if confidentiality or integrity is specified. The Sun ONE Directory Server software sets this to `65535`.

- `charset` – This directive, if present, specifies that the server supports UTF-8 encoding for the `username` and password. If not present, the `username` and password must be encoded in ISO 8859-1 (of which US-ASCII is a subset). The directive is needed for backwards compatibility with HTTP Digest, which only supports ISO 8859-1. This directive may appear at most once; if multiple instances are present, the client should abort the authentication exchange.

Note – Only UTF-8 is supported. Any other charset will be rejected.

- `algorithm` – This directive is required for backwards compatibility with HTTP Digest, which supports other algorithms. This directive is required and must appear exactly once; if not present, or if multiple instances are present, the client should abort the authentication exchange.

Note – Only `md5-sess` is supported. Any other algorithm is rejected.

- `cipher-opts` – A list of ciphers that the server supports. This directive must be present exactly once if `auth-conf` is offered in the `qop-options` directive, in which case implementation of the `3des` and `des` modes is mandatory. The client *must* ignore unrecognized options; if the client recognizes no option, it should abort the authentication exchange.
  - `des` – Data Encryption Standard (DES) cipher [FIPS] in cipherblock chaining (CBC) mode with a 56-bit key.
  - `3des` – the *triple DES* cipher in CBC mode with EDE with the same key for each E stage (aka “two keys mode”) for a total key length of 112 bits.
  - `rc4`, `rc4-40`, `rc4-56` – the RC4 cipher with a 128-bit, 40-bit, and 5-bit key, respectively.

Note – No ciphers are currently offered by the Sun ONE Directory Server software because confidentiality is not yet supported.

- `auth-param` – This construct allows for future extensions; it may appear more than once. The client *must* ignore any unrecognized directives. The size of a `digest-challenge` *must* be less than 2048 bytes.

Note – No `auth-param` is sent by Sun ONE Directory Server software at this time.

3. Digest Response Stage

   The Secured LDAP Client makes note of the `digest-challenge` and responds with a string formatted and computed according to the rules for a `digest-response`. The Secured LDAP Client performs two MD5 hashes of the password with the challenge, and the realm. The challenge is the `nonce`; the realm is the realm containing the user's account. This directive is required if the server provided any realms in the `digest-challenge`, in which case it may appear exactly once and its value *should* be one of those realms. If the directive is missing, realm value is set to the empty string when computing. The way to understand this, is to think of A1 which is the shared secret, whereby A1 is not sent over the wire, but where just a one way hash of it is. Time is not used in the response, but it is possible to generate a random `nonce`. In the Solaris 9 OE, use `/dev/urandom`. For SASL, users are considered to be located in a realm. It is an organizational item. In this case, the server's specified realm (if there is one) is returned.

```
# From RFC 2831 "Digest SASL Mechanism”
digest-response  = 1#( username | realm | nonce | cnonce |
                       nonce-count | qop | digest-uri | response |
                          maxbuf | charset | cipher | authzid |
                          auth-param )

       username         = "username" "=" <"> username-value <">
       username-value   = qdstr-val
       cnonce           = "cnonce" "=" <"> cnonce-value <">
       cnonce-value     = qdstr-val
       nonce-count      = "nc" "=" nc-value
       nc-value         = 8LHEX
       qop              = "qop" "=" qop-value
        digest-uri      = "digest-uri" "=" <"> digest-uri-value <">
       digest-uri-value  = serv-type "/" host [ "/" serv-name ]
       serv-type        = 1*ALPHA
       host             = 1*( ALPHA | DIGIT | "-" | "." )
       serv-name        = host
       response         = "response" "=" response-value
       response-value   = 32LHEX
       LHEX             = "0" | "1" | "2" | "3" |
                          "4" | "5" | "6" | "7" |
                          "8" | "9" | "a" | "b" |
                          "c" | "d" | "e" | "f"
       cipher           = "cipher" "=" cipher-value
       authzid          = "authzid" "=" <"> authzid-value <">
       authzid-value    = qdstr-val
```

In this example, there are various directives and values declared. Below is a short description of what each directive value means:

- `username` – The user’s name in the specified realm, encoded according to the value of the `charset` directive. This directive is required and must be present exactly once; otherwise, authentication fails.
- `realm` – The realm containing the user’s account. This directive is required if the server provided any realms in the `digest-challenge`, in which case it may appear exactly once and its value should be one of those realms. If the directive is missing, `realm-value` sets to the empty string when computing A1.

Note – If the same realm is not specified in the response as was in the challenge (the server’s FQDN), then the authentication will fail.

- nonce – The server-specified data string received in the preceding digest-challenge. This directive is required and *must* be present exactly once; otherwise, authentication fails.

The following is an example of what would be returned in the digest response:

```
# From RFC 2831 "Digest SASL Mechanism"

HEX( KD ( HEX(H(A1)),
{ nonce-value, ":" nc-value, ":",
cnonce-value, ":", qop-value, ":", HEX(H(A2)) }))

This is where SASL digest-md5 will support authzid. NOTE: If the
userid and authzid do not match, then the Sun ONE Directory Server
policy is to refuse the authentication.

A1 = { H( { username-value, ":", realm-value, ":", passwd } ),
":", nonce-value, ":", cnonce-value }
```

Here the username (which must be UTF-8) is in the form that the directory server will map. For example: dn: *ldap_dn* or u: *username* or any other form for which you have matching rules defined.

The realm *must* be the FQDN of the directory server specified, which is sent to the client in the initial challenge stage, and the passwd would be the user's password. H(x) is the DIGEST-MD5 hash of the string, and nonce value is supplied by the directory server. The cnonce is a nonce generated by the Secured LDAP Client.

```
# From RFC 2831 "Digest SASL Mechanism"

A2       = { "AUTHENTICATE:", digest-uri-value }
Let { a, b, ... } be the concatenation of the octet strings a, b,
...

Let H(s) be the 16 octet MD5 hash [RFC 1321] of the octet string s.

Let KD(k, s) be H({k, ":", s}), i.e., the 16 octet hash of the
string k, a colon and the string s.
```

Let HEX(*n*) be the representation of the 16-octet MD5 hash *n* as a string of 32 hex digits (with alphabetic characters always in lower case, because MD5 is case sensitive).

Example: (S==Directory Server and C==Secured LDAP Client

```
S: realm="example.com",nonce="OA6MG9tEQGm2hh",qop="auth",
algorithm=md5-sess,charset=utf-8
```

This is the directory server reply, indicating `realm`, `nonce`, quality of protection it offers, mechanisms, and a specified character set.

Note – The Secured LDAP Client implementation only supports UTF-8.

```
C: charset=utf8,username="michaelh",realm="example.com",
nonce="OA6MG9tEQGm2hh",nc=00000001,cnonce="OA6MHXh6VqTrRk",
digest-uri="ldap/example.com",
response=d388dad90d4bbd760a152321f2143af7,qop=auth
```

We reconfirm a number of elements, and add additional ones, such as `cnonce` and `digest-uri`. These are needed by the directory server to confirm the Secured LDAP Client's password, which the directory server must know. The directory server has all the elements to be able to recompute the response.

The following step can be done if the directory server authentication is desired. However, The Secured LDAP Client currently does not check this, but instead retrieves this information from the server. Future versions of the Secured LDAP Client will check this.

```
S: rspauth=ea40f60335c427b5527b84dbabcdfff
```

The directory server calculates a new digest based on the above algorithm, to prove to the Secured LDAP Client that it knows the Secured LDAP Client password.

Note – The size of a `digest-response` is 2048 bytes, which is the limit for all authentication exchanges for DIGEST-MD5.

---

4. The authentication exchange is completed once the directory server has sent the `reauth` packet.

   Additionally, the DIGEST-MD5 authentication can be used for integrity and confidentiality. Currently neither the Sun ONE Directory Server 5.x software or the Secured LDAP Client supports this functionality.

It is important to understand that in the final exchange from the directory server to the client, the server is indicating that it too can calculate the shared secret A1. This will prove that this is a *trusted* server because it knows how to compute our shared secret.

Note – The DIGEST-MD5 details are buried deep in the underlying `libldap.so.5` library, which is used by the Secured LDAP Client (`libsldap.so.1`) library. In `libsldap`, we just call the LDAP client C API (which is currently a private Solaris OE interface), to bind. The rest is handled by the `libldap` library.

In summary, it is important to emphasize and understand that digests are one-way functions that are relatively easy to compute, but are extremely difficult to determine the possible inverses.

Setting up the SASL DIGEST-MD5 Authentication Mechanism

This procedure in this section uses the `ldapsearch` utility to authenticate against the Sun ONE Directory Server version 5.2 software, using the SASL DIGEST-MD5 authentication mechanism.

Before we get into the step-by-step instructions, it's worth discussing how SASL DIGEST-MD5 is configured because it can be a little tricky.

The DIGEST-MD5 authentication mechanism is now a loadable authentication plug-in in the Sun ONE Directory Server version 5.2 software. Now the LDAP tools (such as `ldapsearch`) rely on the LDAP C-SDK that relies on the `libsasl` to perform a SASL bind. This means that the `libsasl` has to be aware of where to load the DIGEST-MD5 plug-in. When you install the Sun ONE Directory Server version 5.2 software, the plug-ins are copied under *root_server*`/lib/sasl/libdigestmd5.so` (for 32-bit plug-ins), and *root_server*`/lib/sasl/64/libdigestmd5.so` (for 64-bit plug-ins). The directory server needs to be able to load these plug-ins, and know where to get the plug-ins. To achieve this, the directory server looks at the attribute value `dsSaslPluginsPath` under the config entry `cn=sasl,cn=security,cn=config` in the `dse.ldif`. By default, the `dsSaslPluginsPath` is set up to point to *root_server*`/lib/sasl`. You can update this multi-valued attribute if you want to load plug-ins that are stored in another location.

On the client side, it's a little bit different. First, a client might not be aware of where the directory server is installed. But the client needs to retrieve the SASL plug-ins anyway (at least the DIGEST-MD5 plug-in). In the Sun ONE Directory Server 5.2 software (on Solaris OE), the `sasl` library looks at `/usr/lib/mps/sasl2` (for 32-bit plug-ins) and `/usr/lib/mps/sparcv9` (for 64-bit plug-ins).

▼ To Set up the SASL DIGEST-MD5 Authentication Mechanism

Note – This procedure refers to the installation of the unbundled version of the Sun ONE Directory Server 5.2 software. On future versions of Solaris OE, the Sun ONE Directory Server 5.2 software might include a set of SVR4 packages that you install using the `pkgadd` command, and configure using the `directoryserver` utility. If you are installing the directory server from SVR4 packages, your installation steps are different than those listed here. Refer to the product documentation, and to "Differentiating Server and Client Versions" on page 191 for details.

---

1. **Download the Sun ONE Directory Server version 5.2 software product from the `http://www.sun.com` web site.**

2. **Uncompress and extract the Sun ONE Directory Server software.**

   Extract in a directory other than the directory where you intend to install the server (not `ServerRoot`).

3. **Run the `idsktune` utility**

   The `idsktune` utility is located in the root of the directory server distribution. Apply any necessary patches and modifications that are reported by `idsktune`. Once this is done, rerun the `idsktune` utility to confirm that all is as it should be before running the `setup` command.

4. **Install and Configure the Sun ONE Directory Server version 5.2 software.**

   Refer to the product documentation for details.

5. **Create user accounts in the directory server.**

   Use either the Sun ONE Directory Server Console or the `MakeLDIF` utility (this utility is available for download. See "Creating LDIF for Benchmarks" on page 479). This example uses the Java `MakeLDIF` utility.

```
# /usr/ccs/bin/make
Building...
Processed 1000 entries
1002 entries written to example-1k.ldif
Writing filters to example-1k-filter-file.ldif
Wrote 1000 equality filters for uid
Wrote 1000 equality filters for givenname
Wrote 1000 equality filters for sn
Wrote 2479 substring filters for cn
.
.
.
dn: employeeNumber=1000,ou=People,dc=example,dc=com
telephoneNumber: 1-510-315-4801
departmentNumber: 1000
sn: Ruble
employeeType: Employee
employeeNumber: 1000
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
objectclass: mailRecipient
objectclass: nsCalUser
givenName: Lucille
mailDeliveryOption: mailbox
cn: Lucille Ruble
initials: LR
uid: lucyr
mail: Lucy.Ruble@example.com
userPassword: secret123
l: Menlo Park, United States
st: CA
description: This is the description for Lucille Ruble
mailhost: mailhost.example.com
nsCalHost: calhost.example.com
```

6. **Add the newly created user accounts to the directory server.**

   Example using the `ldif2db` utility using the default database back end:

```
$ cd /var/Sun/mps/slapd-instance
$ ./stop-slapd
$ ./ldif2db -n userRoot -i LDIF_file
$ ./start-slapd
```

   Example output from `ldif2db`:

```
importing data ...
[08/Jan/2003:15:46:27 +0000] - import userRoot: Index buffering
enabled with bucket size 16
[08/Jan/2003:15:46:27 +0000] - import userRoot: Beginning import
job...
[08/Jan/2003:15:46:27 +0000] - import userRoot: Processing file
"$(LDIF_FILE)”
[08/Jan/2003:15:46:32 +0000] - import userRoot: Finished scanning
file "$(LDIF_FILE)” (1002 entries)
[08/Jan/2003:15:46:33 +0000] - import userRoot: Workers finished;
cleaning up...
[08/Jan/2003:15:46:36 +0000] - import userRoot: Workers cleaned
up.
[08/Jan/2003:15:46:36 +0000] - import userRoot: Cleaning up
producer thread...
[08/Jan/2003:15:46:36 +0000] - import userRoot: Indexing complete.
Postprocessing...
[08/Jan/2003:15:46:36 +0000] - import userRoot: Flushing caches...
[08/Jan/2003:15:46:36 +0000] - import userRoot: Closing files...
[08/Jan/2003:15:46:37 +0000] - import userRoot: Import complete.
Processed 9002 entries in 10 seconds. (900.20 entries/sec)
```

   Example using the `ldapmodify` utility:

   Make sure that you use the `ldapmodify` command from the `/var/Sun/mps/shared/bin` directory, or some of the options may not be recognized because there are other versions of the `ldapmodify` command in the Solaris OE.

```
$ cd /var/Sun/mps/shared/bin
$./ldapmodify -a -c -h directoryserver_hostname -p ldap_port
-D "cn=Directory Manager" -w "password" -f LDIF_file
-e /var/tmp/ldif.rejects 2> /var/tmp/ldapmodify.log
```

Note – If the `ldapmodify` command is not executed as `./ldapmodify` from the *server-root*`/shared/bin` directory, you must have `LD_LIBRARY_PATH` set to *server-root*`/lib` so that `ldapmodify` finds the appropriate dynamic libraries.

Note – In this example, the `-e /var/tmp/ldif.rejects 2> /var/tmp/ldapmodify.log` string is redirecting messages to a log file.

Example output from `ldapmodify`:

```
adding new entry employeeNumber=8995,ou=People,dc=example,dc=com

adding new entry employeeNumber=8996,ou=People,dc=example,dc=com

adding new entry employeeNumber=8997,ou=People,dc=example,dc=com

adding new entry employeeNumber=8998,ou=People,dc=example,dc=com

adding new entry employeeNumber=8999,ou=People,dc=example,dc=com

adding new entry employeeNumber=9000,ou=People,dc=example,dc=com
.
.
.
```

Example output from the access log:

```
[08/Jan/2003:14:57:49 +0000] conn=24 op=997 msgId=998 - ADD dn=
"employeeNumber=8995,ou=People,dc=example,dc=com"
[08/Jan/2003:14:57:49 +0000] conn=24 op=997 msgId=998 - RESULT
err=0 tag=105 nentries=0 etime=0
[08/Jan/2003:14:57:49 +0000] conn=24 op=998 msgId=999 - ADD dn=
"employeeNumber=8996,ou=People,dc=example,dc=com"
[08/Jan/2003:14:57:49 +0000] conn=24 op=998 msgId=999 - RESULT
err=0 tag=105 nentries=0 etime=0
[08/Jan/2003:14:57:49 +0000] conn=24 op=999 msgId=1000 - ADD dn=
"employeeNumber=8997,ou=People,dc=example,dc=com"
[08/Jan/2003:14:57:49 +0000] conn=24 op=1000 msgId=1001 - ADD dn=
"employeeNumber=8998,ou=People,dc=example,dc=com"
[08/Jan/2003:14:57:49 +0000] conn=24 op=999 msgId=1000 - RESULT
err=0 tag=105 nentries=0 etime=0
[08/Jan/2003:14:57:49 +0000] conn=24 op=1000 msgId=1001 - RESULT
err=0 tag=105 nentries=0 etime=0
[08/Jan/2003:14:57:49 +0000] conn=24 op=1001 msgId=1002 - ADD dn=
"employeeNumber=8999,ou=People,dc=example,dc=com"
[08/Jan/2003:14:57:49 +0000] conn=24 op=1001 msgId=1002 - RESULT
err=0 tag=105 nentries=0 etime=0
[08/Jan/2003:14:57:49 +0000] conn=24 op=1002 msgId=1003 - ADD dn=
"employeeNumber=9000,ou=People,dc=example,dc=com"
[08/Jan/2003:14:57:49 +0000] conn=24 op=1002 msgId=1003 - RESULT
err=0 tag=105 nentries=0 etime=0
[08/Jan/2003:14:57:49 +0000] conn=24 op=1003 msgId=1004 - UNBIND
[08/Jan/2003:14:57:49 +0000] conn=24 op=1003 msgId=-1 - closing -
U1
[08/Jan/2003:14:57:50 +0000] conn=24 op=-1 msgId=-1 - closed
.
.
.
```

7. **Confirm the current value of the** `passwordStorageScheme` **attribute value.**

   The SASL DIGEST-MD5 authentication mechanism is a two-stage bind operation. In the first stage, the client issues a SASL DIGEST bind request (as previously mentioned, the bind DN is not typically included in the first request issued by the client when authenticating using DIGEST-MD5). The directory server returns a challenge. The client performs two DIGEST-MD5 hashes of the password with the Challenge, and the `Realm`. The client sends the result to the directory server. The directory server performs the same hashes and compares the results. Because the

directory server must perform the hashes, the server requires a clear text password. Before changing the `passwordStorageScheme` attribute value, you can confirm the current value using the `ldapsearch` command.

Example output from `ldapsearch`:

```
$ cd /usr/sunone/servers/shared/bin
$ ./ldapsearch -h directoryserver_hostname -p ldap_port
-D "cn=Directory Manager" -w password
-b “cn=Password Policy,cn=config” "(objectclass=*)"
passwordStorageScheme
cn=Password Policy,cn=config
passwordStorageScheme=SSHA
```

8. **Run `ldapsearch` to verify the user, and to verify a successful bind.**

   When you create directory user accounts, the passwords are stored in the default password storage mechanism (Salted Secure Hashing Algorithm (SSHA)). Using one of these accounts, you must change the userpassword to be in the clear. Before changing the `userpassword` of an entry, you should run `ldapsearch` to verify that the user exists in the directory server, (also note the format of the `userpassword` attribute value) and that you can bind successfully with the password.

Example running `ldapsearch` to verify a user entry:

```
$./ldapsearch -h directoryserver_hostname -p ldap_port
-b "dc=example,dc=com" uid=lucyr 1.1
dn: employeeNumber=1000,ou=People,dc=example,dc=com
```

Example of searching for the user entry `lucyr` and displaying the current `userpassword` attribute value:

```
$./ldapsearch -h directoryserver_hostname -p ldap_port
-b "dc=example,dc=com" -D "cn=Directory Manager" -w password uid=
lucyr userpassword
dn: employeeNumber=1000,ou=People,dc=example,dc=com
userpassword: {SSHA}zDGUDF2HHAMzheLjjXSNSem/NS2YSmItdXh8cQ==
```

Example of binding to the directory server as user `lucyr` using the correct credentials:

```
$./ldapsearch -h directoryserver_hostname -p ldap_port
-b "dc=example,dc=com" -D "employeeNumber=1000,ou=People,dc=
example,dc=com" -w password uid=lucyr 1.1
dn: employeeNumber=1000,ou=People,dc=example,dc=com
```

If the preceding `ldapsearch` was not successful, you will see something similar to the following output:

```
ldap_simple_bind: Invalid credentials
```

Note – The correct way in LDAPv3 to request that no attributes be returned is to specify an attribute list of `1.1` as specified in RFC 2251.

---

9. Change the `passwordStorageScheme` attribute value to `{CLEAR}`.

It is not necessary to set the default storage scheme to `clear` in the directory server in order to store passwords in clear text. Passwords can be stored in clear text regardless of the default password storage scheme by prefixing the clear text password with `{CLEAR}`. This can be beneficial if there are only a few special accounts in the server for which authentication will be performed using DIGEST-

MD5 and it is not desirable to store passwords for those other accounts in clear text. However, you will not be able to use pam_unix for authentication if you store passwords in clear text.

Example changing the passwordStorageScheme attribute value:

```
$ cd /usr/sunone/servers/shared/bin
$ ./ldapmodify -h directoryserver_hostname -p ldap_port
-D "cn=Directory Manager" -w password
```

You are now in the ldapmodify interactive mode. Enter the information, and when you are done, use Control-d to exit ldapmodify.

Example:

```
dn: cn=Password Policy,cn=config
changetype: modify
replace: passwordStorageScheme
passwordStorageScheme: clear
modifying entry cn=Password Policy,cn=config
```

10. Confirm the passwordStorateScheme **attribute value change.**

When you verify this attribute change, you see that an entry has been added to cn=# Password Policy,cn=config which looks like this:

```
$ ./ldapsearch -h directoryserver_hostname -p ldap_port
-D "cn=Directory Manager" -w password
-b “cn=Password Policy, cn=config” “(objectclass=*)”
passwordStorageScheme
cn=Password Policy,cn=config
passwordStorageScheme: clear
```

In this particular example, we loaded the directory user entries with the passwordStorageScheme attribute value set to {SSHA}. It is possible to make the modification to the passwordStorageScheme attribute prior to loading your directory entries, however, in practice this is never really the case.

11. **Change the** `userpassword` **attribute value to be in the clear for a user.**

    Example:

```
$ ./ldapmodify -h directoryserver_hostname -p ldap_port
-D "cn=Directory Manager" -w password
dn: employeeNumber=1000,ou=People,dc=example,dc=com
changetype: modify
replace: userPassword
userPassword: password123
modifying entry employeeNumber=1000,ou=People,dc=example,dc=com
                    (Note: use contol-d to exit interactive mode)
```

12. **Verify that the password for the user is now in the clear.**

    When you search the directory server for the user entry `lucyr` and return the `userpassword` attribute value, you should see the clear text value of the password.

    Example:

```
$./ldapsearch -h directoryserver_hostname -p ldap_port
-b "dc=example,dc=com" -D "cn=Directory Manager" -w password uid=
lucyr userpassword
dn: employeeNumber=1000,ou=People,dc=example,dc=com
userpassword: {clear}password123
```

    Example of authenticating using SASL DIGEST-MD5:

```
$./ldapsearch -h directoryserver_hostname -p ldap_port -D "" -w password
-o mech=DIGEST-MD5 -o authid="dn:employeeNumber=1000,ou=
People,dc=example,dc=com" -o authzid="dn:employeeNumber=1000,ou=
People,dc=example,dc=com"
-b "dc=example,dc=com" -s base "(uid=lucyr)"
```

Note – Because of the way the Solaris OE packages install the SASL plug-ins, it is necessary to specify a value for the SASL_PATH environment variable that points to the *server-root*`/lib/sasl` directory.

GSSAPI Authentication and Kerberos v5

This section discusses the GSSAPI mechanism, in particular, Kerberos v5 and how this works in conjunction with the Sun ONE Directory Server 5.2 software and what is involved in implementing such a solution. Please be aware that this is not a trivial task.

It's worth taking a brief look at the relationship between the Generic Security Services Application Program Interface (GSSAPI) and Kerberos v5.

The GSSAPI does not actually provide security services itself. Rather, it is a framework that provides security services to callers in a generic fashion, with a range of underlying mechanisms and technologies such as Kerberos v5. The current implementation of the GSSAPI only works with the Kerberos v5 security mechanism. The best way to think about the relationship between GSSAPI and Kerberos is in the following manner: GSSAPI is a network authentication protocol abstraction that allows Kerberos credentials to be used in an authentication exchange. Kerberos v5 must be installed and running on any system on which GSSAPI-aware programs are running.

The support for the GSSAPI is made possible in the directory server through the introduction of a new SASL library, which is based on the Cyrus CMU implementation. Through this SASL framework, DIGEST-MD5 is supported as explained previously, and GSSAPI which implements Kerberos v5. Additional GSSAPI mechanisms do exist. For example, GSSAPI with SPNEGO support would be GSS-SPNEGO. Other GSS mechanism names are based on the GSS mechanisms OID.

Note – The Sun ONE Directory Server 5.2 software only supports the use of GSSAPI on Solaris OE. There are implementations of GSSAPI for other operating systems (for example, Linux), but the Sun ONE Directory Server 5.2 software does not use them on platforms other than the Solaris OE.

Understanding GSSAPI

The Generic Security Services Application Program Interface (GSSAPI) is a standard interface, defined by RFC 2743, that provides a generic authentication and secure messaging interface, whereby these security mechanisms can be plugged in. The most commonly referred to GSSAPI mechanism is the Kerberos mechanism that is based on secret key cryptography.

One of the main aspects of GSSAPI is that it allows developers to add secure authentication and privacy (encryption and or integrity checking) protection to data being passed over the wire by writing to a single programming interface. This is shown in FIGURE 3-2.

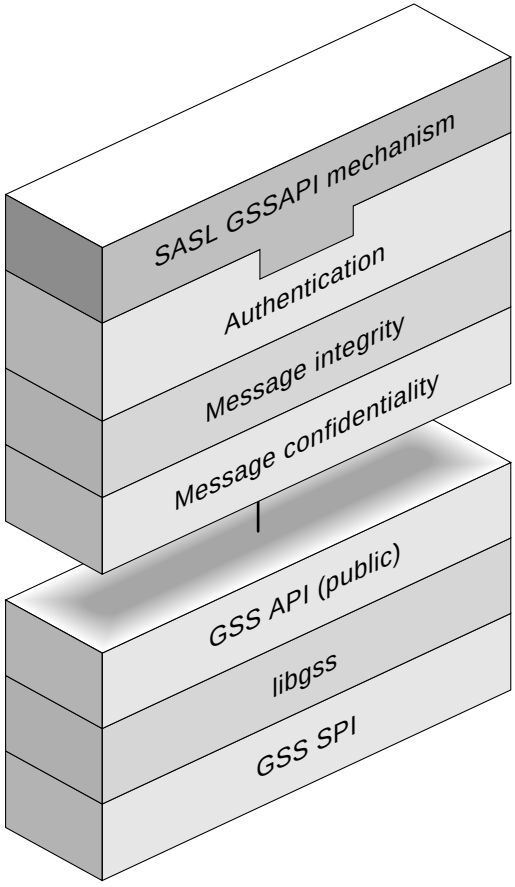

FIGURE 3-2 GSSAPI Layers

The underlying security mechanisms are loaded at the time the programs are executed, as opposed to when they are compiled and built. In practice, the most commonly used GSSAPI mechanism is Kerberos v5. The Solaris OE provides a few different flavors of Diffie-Hellman GSSAPI mechanisms, which are only useful to NIS+ applications.

What can be confusing is that developers might write applications that write directly to the Kerberos API, or they might write GSSAPI applications that request the Kerberos mechanism. There is a big difference, and applications that talk Kerberos directly cannot communicate with those that talk GSSAPI. The *wire* protocols are not

compatible, even though the underlying Kerberos protocol is in use. An example is `telnet` with Kerberos is a secure `telnet` program that authenticates a `telnet` user and encrypts data, including passwords exchanged over the network during the `telnet` session. The authentication and message protection features are provided using Kerberos. The `telnet` application with Kerberos only uses Kerberos, which is based on secret-key technology. However, a `telnet` program written to the GSSAPI interface can use Kerberos as well as other security mechanisms supported by GSSAPI.

The Solaris OE does not deliver any libraries that provide support for third-party companies to program directly to the Kerberos API. The goal is to encourage developers to use the GSSAPI. Many open-source Kerberos implementations (MIT, Heimdal) allow users to write Kerberos applications directly.

Note – *On the wire*, the GSSAPI is compatible with Microsoft's SSPI and thus GSSAPI applications can communicate with Microsoft applications that use SSPI and Kerberos.

The GSSAPI is preferred because it is a standardized API, whereas Kerberos is not. This means that the MIT Kerberos development team might change the programming interface anytime, and any applications that exist today might not work in the future without some code modifications. Using GSSAPI avoids this problem.

Another benefit of GSSAPI is its pluggable feature, which is a big benefit, especially if a developer later decides that there is a better authentication method than Kerberos, because it can easily be plugged into the system and the existing GSSAPI applications should be able to use it without being recompiled or patched in any way.

Understanding Kerberos v5

Kerberos is a network authentication protocol designed to provide strong authentication for client/server applications by using secret-key cryptography. Originally developed at the Massachusetts Institute of Technology, it is included in the Solaris OE to provide strong authentication for Solaris OE network applications.

In addition to providing a secure authentication protocol, Kerberos also offers the ability to add privacy support (encrypted data streams) for remote applications such as `telnet`, `ftp`, `rsh`, `rlogin`, and other common UNIX network applications. In the Solaris OE, Kerberos can also be used to provide strong authentication and privacy support for Network File Systems (NFS), allowing secure and private file sharing across the network.

Because of its widespread acceptance and implementation in other operating systems, including Windows 2000, HP-UX, and Linux, the Kerberos authentication protocol can interoperate in a heterogeneous environment, allowing users on machines running one OS to securely authenticate themselves on hosts of a different OS.

The Kerberos software is available for Solaris OE versions 2.6, 7, 8, and 9 in a separate package called the Sun Enterprise Authentication Mechanism (SEAM) software. For Solaris 2.6 and Solaris 7 OE, Sun Enterprise Authentication Mechanism software is included as part of the Solaris Easy Access Server 3.0 (Solaris SEAS) package. For Solaris 8 OE, the Sun Enterprise Authentication Mechanism software package is available with the Solaris 8 OE Admin Pack.

For Solaris 2.6 and Solaris 7 OE, the Sun Enterprise Authentication Mechanism software is freely available as part of the Solaris Easy Access Server 3.0 package available for download from:
`http://www.sun.com/software/solaris/7/ds/ds-seas`.

For Solaris 8 OE systems, Sun Enterprise Authentication Mechanism software is available in the Solaris 8 OE Admin Pack, available for download from:
`http://www.sun.com/bigadmin/content/adminPack/index.html`.

For Solaris 9 OE systems, Sun Enterprise Authentication Mechanism software is already installed by default and contains the following packages listed in TABLE 3-1.

TABLE 3-1 Solaris 9 OE Kerberos v5 Packages

Package Name	Description
`SUNWkdcr`	Kerberos v5 KDC (root)
`SUNWkdcu`	Kerberos v5 Master KDC (user)
`SUNWkrbr`	Kerberos version 5 support (Root)
`SUNWkrbu`	Kerberos version 5 support (Usr)
`SUNWkrbux`	Kerberos version 5 support (Usr) (64-bit)

All of these Sun Enterprise Authentication Mechanism software distributions are based on the MIT KRB5 Release version 1.0. The client programs in these distributions are compatible with later MIT releases (1.1, 1.2) and with other implementations that are compliant with the standard.

How Kerberos Works

The following is an overview of the Kerberos v5 authentication system. From the user's standpoint, Kerberos v5 is mostly invisible after the Kerberos session has been started. Initializing a Kerberos session often involves no more than logging in and providing a Kerberos password.

The Kerberos system revolves around the concept of a *ticket*. A ticket is a set of electronic information that serves as identification for a user or a service such as the NFS service. Just as your driver's license identifies you and indicates what driving permissions you have, so a ticket identifies you and your network access privileges. When you perform a Kerberos-based transaction (for example, if you use `rlogin` to log in to another machine), your system transparently sends a request for a ticket to a Key Distribution Center, or KDC. The KDC accesses a database to authenticate your identity and returns a ticket that grants you permission to access the other machine. *Transparently* means that you do not need to explicitly request a ticket.

Tickets have certain attributes associated with them. For example, a ticket can be forwardable (which means that it can be used on another machine without a new authentication process), or postdated (not valid until a specified time). How tickets are used (for example, which users are allowed to obtain which types of tickets) is set by policies that are determined when Kerberos is installed or administered.

Note – You will frequently see the terms *credential* and *ticket*. In the Kerberos world, they are often used interchangeably. Technically, however, a credential is a ticket plus the session key for that session.

Initial Authentication

Kerberos authentication has two phases, an initial authentication that allows for all subsequent authentications, and the subsequent authentications themselves.

A client (a user, or a service such as NFS) begins a Kerberos session by requesting a ticket-granting ticket (TGT) from the Key Distribution Center (KDC). This request is often done automatically at login.

A ticket-granting ticket is needed to obtain other tickets for specific services. Think of the ticket-granting ticket as something similar to a passport. Like a passport, the ticket-granting ticket identifies you and allows you to obtain numerous "visas," where the "visas" (tickets) are not for foreign countries, but for remote machines or network services. Like passports and visas, the ticket-granting ticket and the other various tickets have limited lifetimes. The difference is that *Kerberized* commands notice that you have a passport and obtain the visas for you. You don't have to perform the transactions yourself.

The KDC creates a ticket-granting ticket and sends it back, in encrypted form, to the client. The client decrypts the ticket-granting ticket using the client's password.

Now in possession of a valid ticket-granting ticket, the client can request tickets for all sorts of network operations for as long as the ticket-granting ticket lasts. This ticket usually lasts for a few hours. Each time the client performs a unique network operation, it requests a ticket for that operation from the KDC.

Subsequent Authentications

The client requests a ticket for a particular service from the KDC by sending the KDC its ticket-granting ticket as proof of identity.

1. The KDC sends the ticket for the specific service to the client.

   For example, suppose user lucy wants to access an NFS file system that has been shared with krb5 authentication required. Since she is already authenticated (that is, she already has a ticket-granting ticket), as she attempts to access the files, the NFS client system automatically and transparently obtains a ticket from the KDC for the NFS service.

2. The client sends the ticket to the server.

   When using the NFS service, the NFS client automatically and transparently sends the ticket for the NFS service to the NFS server.

3. The server allows the client access.

   These steps make it appear that the server doesn't ever communicate with the KDC. The server does, though, as it registers itself with the KDC, just as the first client does.

Principals

A client is identified by its principal. A principal is a unique identity to which the KDC can assign tickets. A principal can be a user, such as `joe`, or a service, such as NFS.

By convention, a principal name is divided into three parts: the primary, the instance, and the realm. A typical principal could be, for example, `lucy/admin@EXAMPLE.COM`, where:

`lucy` is the primary. The primary can be a user name, as shown here, or a service, such as NFS. The primary can also be the word `host`, which signifies that this principal is a service principal that is set up to provide various network services.

admin is the instance. An instance is optional in the case of user principals, but it is required for service principals. For example, if the user lucy sometimes acts as a system administrator, she can use lucy/admin to distinguish herself from her usual user identity. Likewise, if Lucy has accounts on two different hosts, she can use two principal names with different instances (for example, lucy/california.example.com and lucy/boston.example.com).

Realms

A realm is a logical network, similar to a domain, which defines a group of systems under the same master KDC. Some realms are hierarchical (one realm being a superset of the other realm). Otherwise, the realms are non-hierarchical (or direct) and the mapping between the two realms must be defined.

Realms and KDC Servers

Each realm must include a server that maintains the master copy of the principal database. This server is called the master KDC server. Additionally, each realm should contain at least one slave KDC server, which contains duplicate copies of the principal database. Both the master KDC server and the slave KDC server create tickets that are used to establish authentication.

Understanding the Kerberos KDC

The Kerberos Key Distribution Center (KDC) is a trusted server that issues Kerberos tickets to clients and servers to communicate securely. A Kerberos ticket is a block of data that is presented as the user's credentials when attempting to access a Kerberized service. A ticket contains information about the user's identity and a temporary encryption key, all encrypted in the server's private key. In the Kerberos environment, any entity that is defined to have a Kerberos identity is referred to as a *principal*.

A principal may be an entry for a particular user, host, or service (such as NFS or FTP) that is to interact with the KDC. Most commonly, the KDC server system also runs the Kerberos Administration Daemon, which handles administrative commands such as adding, deleting, and modifying principals in the Kerberos database. Typically, the KDC, the admin server, and the database are all on the same machine, but they can be separated if necessary. Some environments may require that multiple realms be configured with master KDCs and slave KDCs for each realm. The principals applied for securing each realm and KDC should be applied to all realms and KDCs in the network to ensure that there isn't a single weak link in the chain.

One of the first steps to take when initializing your Kerberos database is to create it using the `kdb5_util` command, which is located in `/usr/sbin`. When running this command, the user has the choice of whether to create a stash file or not. The stash file is a local copy of the master key that resides on the KDC's local disk. The master key contained in the stash file is generated from the master password that the user enters when first creating the KDC database. The stash file is used to authenticate the KDC to itself automatically before starting the `kadmind` and `krb5kdc` daemons (for example, as part of the machine's boot sequence).

If a stash file is not used when the database is created, the administrator who starts up the `krb5kdc` process will have to manually enter the master key (password) every time they start the process. This may seem like a typical trade off between convenience and security, but if the rest of the system is sufficiently hardened and protected, very little security is lost by having the master key stored in the protected stash file. It is recommended that at least one slave KDC server be installed for each realm to ensure that a backup is available in the event that the master server becomes unavailable, and that slave KDC be configured with the same level of security as the master.

Currently, the Sun Kerberos v5 Mechanism utility, `kdb5_util`, can create three types of keys, `DES-CBC-CRC`, `DES-CBC-MD5`, and `DES-CBC-RAW`. DES-CBC stands for DES encryption with Cipher Block Chaining and the CRC, MD5, and RAW designators refer to the checksum algorithm that is used. By default, the key created will be `DES-CBC-CRC`, which is the default encryption type for the KDC. The type of key created is specified on the command line with the `-k` option (see the `kdb5_util`(1M) man page). Choose the password for your stash file very carefully, because this password can be used in the future to decrypt the master key and modify the database. The password may be up to 1024 characters long and can include any combination of letters, numbers, punctuation, and spaces.

The following is an example of creating a stash file:

```
kdc1 # /usr/sbin/kdb5_util create -r EXAMPLE.COM -s
Initializing database '/var/krb5/principal' for realm
'EXAMPLE.COM'
master key name 'K/M@EXAMPLE.COM'
You will be prompted for the database Master Password.
It is important that you NOT FORGET this password.
Enter KDC database master key: master_key
Re-enter KDC database master key to verify: master_key
```

Notice the use of the `-s` argument to create the stash file. The location of the stash file is in the `/var/krb5`. The stash file appears with the following mode and ownership settings:

```
kdc1 # cd /var/krb5
kdc1 # ls -l
-rw------- 1 root other 14 Apr 10 14:28 .k5.EXAMPLE.COM
```

Note – The directory used to store the stash file and the database should not be shared or exported.

Secure Settings in the KDC Configuration File

The KDC and Administration daemons both read configuration information from `/etc/krb5/kdc.conf`. This file contains KDC-specific parameters that govern overall behavior for the KDC and for specific realms. The parameters in the `kdc.conf` file are explained in detail in the `kdc.conf`(4) man page.

The `kdc.conf` parameters describe locations of various files and ports to use for accessing the KDC and the administration daemon. These parameters generally do not need to be changed, and doing so does not result in any added security. However, there are some parameters that may be adjusted to enhance the overall security of the KDC. The following are some examples of adjustable parameters that enhance security.

- `kdc_ports` – Defines the ports that the KDC will listen on to receive requests. The standard port for Kerberos v5 is 88. 750 is included and commonly used to support older clients that still use the default port designated for Kerberos v4. Solaris OE still listens on port 750 for backwards compatibility. This is not considered a security risk.
- `max_life` – Defines the maximum lifetime of a ticket, and defaults to eight hours. In environments where it is desirable to have users re-authenticate frequently and to reduce the chance of having a principal's credentials stolen, this value should be lowered. The recommended value is eight hours.
- `max_renewable_life` – Defines the period of time from when a ticket is issued that it may be renewed (using `kinit -R`). The standard value here is 7 days. To disable renewable tickets, this value may be set to 0 days, 0 hrs, 0 min. The recommended value is 7d 0h 0m 0s.
- `default_principal_expiration` – A Kerberos principal is any unique identity to which Kerberos can assign a ticket. In the case of users, it is the same as the UNIX system user name. The default lifetime of any principal in the realm may be defined in the `kdc.conf` file with this option. This should be used only if the realm will contain temporary principals, otherwise the administrator will

have to constantly be renewing principals. Usually, this setting is left undefined and principals do not expire. This is not insecure as long as the administrator is vigilant about removing principals for users that no longer need access to the systems.

- `supported_enctypes` – The encryption types supported by the KDC may be defined with this option. At this time, Sun Enterprise Authentication Mechanism software only supports `des-cbc-crc:normal` encryption type, but in the future this may be used to ensure that only strong cryptographic ciphers are used.
- `dict_file` – The location of a dictionary file containing strings that are not allowed as passwords. A principal with any password policy (see below) will not be able to use words found in this dictionary file. This is not defined by default. Using a dictionary file is a good way to prevent users from creating trivial passwords to protect their accounts, and thus helps avoid one of the most common weaknesses in a computer network-guessable passwords. The KDC will only check passwords against the dictionary for principals which have a password policy association, so it is good practice to have at least one simple policy associated with all principals in the realm.

The Solaris OE has a default system dictionary that is used by the spell program that may also be used by the KDC as a dictionary of common passwords. The location of this file is: `/usr/share/lib/dict/words`. Other dictionaries may be substituted. The format is one word or phrase per line.

The following is a Kerberos v5 `/etc/krb5/kdc.conf` example with suggested settings:

```
# Copyright 1998-2002 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
#ident  "@(#)kdc.conf   1.2     02/02/14 SMI"

[kdcdefaults]
        kdc_ports = 88,750

[realms]
        ___default_realm___ = {
                profile = /etc/krb5/krb5.conf
                database_name = /var/krb5/principal
                admin_keytab = /etc/krb5/kadm5.keytab
                acl_file = /etc/krb5/kadm5.acl
                kadmind_port = 749
                max_life = 8h 0m 0s
                max_renewable_life = 7d 0h 0m 0s
                default_principal_flags = +preauth
Needs moving -- dict_file = /usr/share/lib/dict/words
        }
```

Access Control

The Kerberos administration server allows for granular control of the administrative commands by use of an access control list (ACL) file (`/etc/krb5/kadm5.acl`). The syntax for the ACL file allows for wildcarding of principal names so it is not necessary to list every single administrator in the ACL file. This feature should be used with great care. The ACLs used by Kerberos allow privileges to be broken down into very precise functions that each administrator can perform. If a certain administrator only needs to be allowed to have read-access to the database then that person should not be granted full admin privileges. Below is a list of the privileges allowed:

- `a` – Allows the addition of principals or policies in the database.
- `A` – Prohibits the addition of principals or policies in the database.
- `d` – Allows the deletion of principals or policies in the database.
- `D` – Prohibits the deletion of principals or policies in the database.
- `m` – Allows the modification of principals or policies in the database.
- `M` – Prohibits the modification of principals or policies in the database.
- `c` – Allows the changing of passwords for principals in the database.
- `C` – Prohibits the changing of passwords for principals in the database.
- `i` – Allows inquiries to the database.
- `I` – Prohibits inquiries to the database.
- `l` – Allows the listing of principals or policies in the database.
- `L` – Prohibits the listing of principals or policies in the database.
- `*` – Short for all privileges (`admcil`).
- `x` – Short for all privileges (`admcil`). Identical to `*`.

Adding Administrators

After the ACLs are set up, actual administrator principals should be added to the system. It is strongly recommended that administrative users have separate `/admin` principals to use only when administering the system. For example, user Lucy would have two principals in the database - `lucy@REALM` and `lucy/admin@REALM`. The `/admin` principal would only be used when administering the system, not for getting ticket-granting-tickets (TGTs) to access remote services. Using the `/admin` principal only for administrative purposes minimizes the chance of someone walking up to Joe's unattended terminal and performing unauthorized administrative commands on the KDC.

Kerberos principals may be differentiated by the instance part of their principal name. In the case of user principals, the most common instance identifier is `/admin`. It is standard practice in Kerberos to differentiate user principals by defining some

to be /admin instances and others to have no specific instance identifier (for example, lucy/admin@REALM versus lucy@REALM). Principals with the /admin instance identifier are assumed to have administrative privileges defined in the ACL file and should only be used for administrative purposes. A principal with an /admin identifier which does not match up with any entries in the ACL file will not be granted any administrative privileges, it will be treated as a non-privileged user principal. Also, user principals with the /admin identifier are given separate passwords and separate permissions from the non-admin principal for the same user.

The following is a sample /etc/krb5/kadm5.acl file:

```
# Copyright (c) 1998-2000 by Sun Microsystems, Inc.
# All rights reserved.
#
#pragma ident   "@(#)kadm5.acl  1.1     01/03/19 SMI"

# lucy/admin is given full administrative privilege
lucy/admin@EXAMPLE.COM *
#
# tom/admin user is allowed to query the database (d), listing
principals
# (l), and changing user passwords (c)
#
tom/admin@EXAMPLE.COM dlc
```

Note – It is highly recommended that the kadm5.acl file be tightly controlled and that users be granted only the privileges they need to perform their assigned tasks.

Creating Host Keys

Creating host keys for systems in the realm such as slave KDCs is performed the same way that creating user principals is performed. However, the -randkey option should always be used, so no one ever knows the actual key for the hosts. Host principals are almost always stored in the keytab file, to be used by root-owned processes that wish to act as Kerberos services for the local host. It is rarely necessary for anyone to actually know the password for a host principal because the key is stored safely in the keytab and is only accessible by root-owned processes, never by actual users.

When creating keytab files, the keys should always be extracted from the KDC on the same machine where the keytab is to reside using the ktadd command from a kadmin session. If this is not feasible, take great care in transferring the keytab file from one machine to the next. A malicious attacker who possesses the contents of

the keytab file could use these keys from the file in order to gain access to another user or services credentials. Having the keys would then allow the attacker to impersonate whatever principal that the key represented and further compromise the security of that Kerberos realm. Some suggestions for transferring the keytab are to use Kerberized, encrypted ftp transfers, or to use the secure file transfer programs scp or sftp offered with the SSH package (http://www.openssh.org). Another safe method is to place the keytab on a removable disk, and hand-deliver it to the destination.

Hand delivery does not scale well for large installations, so using the Kerberized ftp daemon is perhaps the most convenient and secure method available.

Using NTP *to Synchronize Clocks*

All servers participating in the Kerberos realm need to have their system clocks synchronized to within a configurable time limit (default 300 seconds). The safest, most secure way to systematically synchronize the clocks on a network of Kerberos servers is by using the Network Time Protocol (NTP) service. The Solaris OE comes with an NTP client and NTP server software (SUNWntpu package). See the ntpdate(1M) and xntpd(1M) man pages for more information on the individual commands. For more information on configuring NTP, refer to the following Sun BluePrints OnLine NTP articles:

- "Using NTP to Control and Synchronize System Clocks – Part I: Introduction to NTP" (July 2001), http://www.sun.com/blueprints/0701/NTP.pdf
- "Using NTP to Control and Synchronize System Clocks – Part II: Basic NTP Administration and Architecture" (August 2001), http://www.sun.com/blueprints/0801/NTPpt2.pdf
- "Using NTP to Control and Synchronize System Clocks – Part III: NTP Monitoring and Troubleshooting" (September 2001), http://www.sun.com/blueprints/0901/NTPpt3.pdf

It is critical that the time be synchronized in a secure manner. A simple denial of service attack on either a client or a server would involve just skewing the time on that system to be outside of the configured clock skew value, which would then prevent anyone from acquiring TGTs from that system or accessing Kerberized services on that system. The default clock-skew value of five minutes is the maximum recommended value.

The NTP infrastructure must also be secured, including the use of server hardening for the NTP server and application of NTP security features. Using the Solaris Security Toolkit software (formerly known as JASS) with the secure.driver script to create a minimal system and then installing just the necessary NTP software is one such method. The Solaris Security Toolkit software is available at:

http://www.sun.com/security/jass/

Documentation on the Solaris Security Toolkit software is available at:

`http://www.sun.com/security/blueprints`

Establishing Password Policies

Kerberos allows the administrator to define password policies that can be applied to some or all of the user principals in the realm. A password policy contains definitions for the following parameters:

- Minimum Password Length – The number of characters in the password, for which the recommended value is 8.
- Maximum Password Classes – The number of different character classes that must be used to make up the password. Letters, numbers, and punctuation are the three classes and valid values are `1`, `2`, and `3`. The recommended value is `2`.
- Saved Password History – The number of previous passwords that have been used by the principal that cannot be reused. The recommended value is `3`.
- Minimum Password Lifetime (seconds) – The minimum time that the password must be used before it can be changed. The recommended value is `3600` (1 hour).
- Maximum Password Lifetime (seconds) – The maximum time that the password can be used before it must be changed. The recommended value is `7776000` (90 days).

These values can be set as a group and stored as a single policy. Different policies can be defined for different principals. It is recommended that the minimum password length be set to at least 8 and that at least 2 classes be required. Most people tend to choose easy-to-remember and easy-to-type passwords, so it is a good idea to at least set up policies to encourage slightly more difficult-to-guess passwords through the use of these parameters. Setting the Maximum Password Lifetime value may be helpful in some environments, to force people to change their passwords periodically. The period is up to the local administrator according to the overriding corporate security policy used at that particular site. Setting the Saved Password History value combined with the Minimum Password Lifetime value prevents people from simply switching their password several times until they get back to their original or favorite password.

The maximum password length supported is 255 characters, unlike the UNIX password database which only supports up to 8 characters. Passwords are stored in the KDC encrypted database using the KDC default encryption method, `DES-CBC-CRC`. In order to prevent password guessing attacks, it is recommended that users choose long passwords or pass phrases. The 255 character limit allows one to choose a small sentence or easy to remember phrase instead of a simple one-word password.

It is possible to use a dictionary file that can be used to prevent users from choosing common, easy-to-guess words (see “Secure Settings in the KDC Configuration File” on page 70). The dictionary file is only used when a principal has a policy association, so it is highly recommended that at least one policy be in effect for all principals in the realm.

The following is an example password policy creation:

If you specify a `kadmin` command without specifying any options, `kadmin` displays the syntax (usage information) for that command. The following code box shows this, followed by an actual `add_policy` command with options.

```
kadmin: add_policy
usage: add_policy [options] policy
options are:
[-maxlife time] [-minlife time] [-minlength length]
[-minclasses number] [-history number]
kadmin: add_policy -minlife "1 hour" -maxlife "90 days"
-minlength 8 -minclasses 2 -history 3 passpolicy
kadmin: get_policy passpolicy
Policy: passpolicy
Maximum password life: 7776000
Minimum password life: 3600
Minimum password length: 8
Minimum number of password character classes: 2
Number of old keys kept: 3
Reference count: 0
```

This example creates a password policy called `passpolicy` which enforces a maximum password lifetime of 90 days, minimum length of 8 characters, a minimum of 2 different character classes (letters, numbers, punctuation), and a password history of 3.

To apply this policy to an existing user, modify the following:

```
kadmin: modprinc -policy passpolicy lucyPrincipal
"lucy@EXAMPLE.COM" modified.
```

To modify the default policy that is applied to all user principals in a realm, change the following:

```
kadmin: modify_policy -maxlife "90 days" -minlife "1 hour"
-minlength 8 -minclasses 2 -history 3 default
kadmin: get_policy default
Policy: default
Maximum password life: 7776000
Minimum password life: 3600
Minimum password length: 8
Minimum number of password character classes: 2
Number of old keys kept: 3
Reference count: 1
```

The Reference count value indicates how many principals are configured to use the policy.

Note – The default policy is automatically applied to all new principals that are not given the same password as the principal name when they are created. Any account with a policy assigned to it is uses the dictionary (defined in the `dict_file` parameter in `/etc/krb5/kdc.conf`) to check for common passwords.

Backing Up a KDC

Backups of a KDC system should be made regularly or according to local policy. However, backups should exclude the `/etc/krb5/krb5.keytab` file. If the local policy requires that backups be done over a network, then these backups should be secured either through the use of encryption or possibly by using a separate network interface that is only used for backup purposes and is not exposed to the same traffic as the non-backup network traffic. Backup storage media should always be kept in a secure, fireproof location.

Monitoring the KDC

Once the KDC is configured and running, it should be continually and vigilantly monitored. The Sun Kerberos v5 software KDC logs information into the `/var/krb5/kdc.log` file, but this location can be modified in the `/etc/krb5/krb5.conf` file, in the logging section.

```
[logging]
        default = FILE:/var/krb5/kdc.log
        kdc = FILE:/var/krb5/kdc.log
```

The KDC log file should have read and write permissions for the root user only, as follows:

```
-rw------ 1 root other 750 25 May 10 17:55 /var/krb5/kdc.log
```

Kerberos Options

The `/etc/krb5/krb5.conf` file contains information that all Kerberos applications use to determine what server to talk to and what realm they are participating in. Configuring the `krb5.conf` file is covered in the *Sun Enterprise Authentication Mechanism Software Installation Guide.* Also refer to the `krb5.conf`(4) man page for a full description of this file.

The `appdefaults` section in the `krb5.conf` file contains parameters that control the behavior of many Kerberos client tools. Each tool may have its own section in the `appdefaults` section of the `krb5.conf` file.

Many of the applications that use the `appdefaults` section, use the same options; however, they might be set in different ways for each client application.

Kerberos Client Applications

The following Kerberos applications can have their behavior modified through the user of options set in the `appdefaults` section of the `/etc/krb5/krb5.conf` file or by using various command-line arguments. These clients and their configuration settings are described below.

kinit

The kinit client is used by people who want to obtain a TGT from the KDC. The /etc/krb5/krb5.conf file supports the following kinit options: renewable, forwardable, no_addresses, max_life, max_renewable_life and proxiable.

telnet

The Kerberos telnet client has many command-line arguments that control its behavior. Refer to the man page for complete information. However, there are several interesting security issues involving the Kerberized telnet client.

The telnet client uses a session key even after the service ticket which it was derived from has expired. This means that the telnet session remains active even after the ticket originally used to gain access, is no longer valid. This is insecure in a strict environment, however, the trade off between ease of use and strict security tends to lean in favor of ease-of-use in this situation. It is recommended that the telnet connection be re-initialized periodically by disconnecting and reconnecting with a new ticket. The overall lifetime of a ticket is defined by the KDC (/etc/krb5/kdc.conf), normally defined as eight hours.

The telnet client allows the user to forward a copy of the credentials (TGT) used to authenticate to the remote system using the -f and -F command-line options. The -f option sends a non-forwardable copy of the local TGT to the remote system so that the user can access Kerberized NFS mounts or other local Kerberized services on that system only. The -F option sends a forwardable TGT to the remote system so that the TGT can be used from the remote system to gain further access to other remote Kerberos services beyond that point. The -F option is a superset of -f. If the Forwardable and or forward options are set to false in the krb5.conf file, these command-line arguments can be used to override those settings, thus giving individuals the control over whether and how their credentials are forwarded.

The *-x* option should be used to turn on encryption for the data stream. This further protects the session from eavesdroppers. If the telnet server does not support encryption, the session is closed. The /etc/krb5/krb5.conf file supports the following telnet options: forward, forwardable, encrypt, and autologin. The autologin [true/false] parameter tells the client to try and attempt to log in without prompting the user for a user name. The local user name is passed on to the remote system in the telnet negotiations.

rlogin *and* rsh

The Kerberos rlogin and rsh clients behave much the same as their non-Kerberized equivalents. Because of this, it is recommended that if they are required to be included in the network files such as /etc/hosts.equiv and .rhosts that the root users directory be removed. The Kerberized versions have the added benefit of using Kerberos protocol for authentication and can also use Kerberos to protect the privacy of the session using encryption.

Similar to telnet described previously, the rlogin and rsh clients use a session key after the service ticket which it was derived from has expired. Thus, for maximum security, rlogin and rsh sessions should be re-initialized periodically. rlogin uses the -f, -F, and -x options in the same fashion as the telnet client. The /etc/krb5/krb5.conf file supports the following rlogin options: forward, forwardable, and encrypt.

Command-line options override configuration file settings. For example, if the rsh section in the krb5.conf file indicates encrypt false, but the -x option is used on the command line, an encrypted session is used.

rcp

Kerberized rcp can be used to transfer files securely between systems using Kerberos authentication and encryption (with the -x command-line option). It does not prompt for passwords, the user must already have a valid TGT before using rcp if they wish to use the encryption feature. However, beware if the -x option is not used and no local credentials are available, the rcp session will revert to the standard, non-Kerberized (and insecure) rcp behavior. It is highly recommended that users always use the -x option when using the Kerberized rcp client.The /etc/krb5/krb5.conf file supports the encrypt [true/false] option.

login

The Kerberos login program (login.krb5) is forked from a successful authentication by the Kerberized telnet daemon or the Kerberized rlogin daemon. This Kerberos login daemon is separate from the standard Solaris OE login daemon and thus, the standard Solaris OE features such as BSM auditing are not yet supported when using this daemon. The /etc/krb5/krb5.conf file supports the krb5_get_tickets [true/false] option. If this option is set to true, then the login program will generate a new Kerberos ticket (TGT) for the user upon proper authentication.

ftp

The Sun Enterprise Authentication Mechanism (SEAM) version of the `ftp` client uses the GSSAPI (RFC 2743) with Kerberos v5 as the default mechanism. This means that it uses Kerberos authentication and (optionally) encryption through the Kerberos v5 GSS mechanism. The only Kerberos-related command-line options are `-f` and `-m`. The `-f` option is the same as described above for `telnet` (there is no need for a `-F` option). `-m` allows the user to specify an alternative GSS mechanism if so desired, the default is to use the `kerberos_v5` mechanism.

The protection level used for the data transfer can be set using the `protect` command at the `ftp` prompt. Sun Enterprise Authentication Mechanism software `ftp` supports the following protection levels:

- Clear unprotected, unencrypted transmission
- Safe data is integrity protected using cryptographic checksums
- Private data is transmitted with confidentiality and integrity using encryption

It is recommended that users set the protection level to `private` for all data transfers. The `ftp` client program does not support or reference the `krb5.conf` file to find any optional parameters. All `ftp` client options are passed on the command line. See the man page for the Kerberized `ftp` client, `ftp`(1).

In summary, adding Kerberos to a network can increase the overall security available to the users and administrators of that network. Remote sessions can be securely authenticated and encrypted, and shared disks can be secured and encrypted across the network. In addition, Kerberos allows the database of user and service principals to be managed securely from any machine which supports the SEAM software Kerberos protocol. SEAM is interoperable with other RFC 1510 compliant Kerberos implementations such as MIT Krb5 and some MS Windows 2000 Active Directory services. Adopting the practices recommended in this section further secure the SEAM software infrastructure to help ensure a safer network environment.

Implementing the Sun ONE Directory Server 5.2 Software and the GSSAPI Mechanism

This section provides a high-level overview, followed by the in-depth procedures that describe the setup necessary to implement the GSSAPI mechanism and the Sun ONE Directory Server 5.2 software. This implementation assumes a realm of `EXAMPLE.COM` for this purpose. The following list gives an initial high-level overview of the steps required, with the next section providing the detailed information.

1. Setup DNS on the client machine. This is an important step because Kerberos requires DNS.
2. Install and configure the Sun ONE Directory Server version 5.2 software.
3. Check that the directory server and client both have the SASL plug-ins installed.
4. Install and configure Kerberos v5.
5. Edit the `/etc/krb5/krb5.conf` file.
6. Edit the `/etc/krb5/kdc.conf` file.
7. Edit the `/etc/krb5/kadm5.acl` file.
8. Move the `kerberos_v5` line so it is the first line in the `/etc/gss/mech` file.
9. Create new principals using `kadmin.local`, which is an interactive command-line interface to the Kerberos v5 administration system.
10. Modify the rights for `/etc/krb5/krb5.keytab`. This access is necessary for the Sun ONE Directory Server 5.2 software.
11. Run `/usr/sbin/kinit`.
12. Check that you have a ticket with `/usr/bin/klist`.
13. Perform an `ldapsearch`, using the `ldapsearch` command-line tool from the Sun ONE Directory Server 5.2 software to test and verify.

The sections that follow fill in the details.

Configuring a DNS Client

To be a DNS client, a machine must run the *resolver*. The resolver is neither a daemon nor a single program. It is a set of dynamic library routines used by applications that need to know machine names. The resolver's function is to resolve users' queries. To do that, it queries a name server, which then returns either the requested information or a referral to another server. Once the resolver is configured, a machine can request DNS service from a name server.

The following example shows you how to configure the `resolv.conf`(4) file in the server `kdc1` in the `example.com` domain.

```
;
; /etc/resolv.conf file for dnsmaster
;
 domain             example.com
 nameserver         192.168.0.0
 nameserver         192.168.0.1
```

The first line of the `/etc/resolv.conf` file lists the domain name in the form:

`domain` *domainname*

Note – No spaces or tabs are permitted at the end of the domain name. Make sure that you press return immediately after the last character of the domain name.

The second line identifies the server itself in the form:

`nameserver` *IP_address*

Succeeding lines list the IP addresses of one or two slave or cache-only name servers that the resolver should consult to resolve queries. Name server entries have the form:

`nameserver` *IP_address*

IP_address is the IP address of a slave or cache-only DNS name server. The resolver queries these name servers in the order they are listed until it obtains the information it needs.

For more detailed information of what the `resolv.conf` file does, refer to the `resolv.conf`(4) man page.

▼ To Configure Kerberos v5 (Master KDC)

In the this procedure, the following configuration parameters are used:

- Realm name = `EXAMPLE.COM`
- DNS domain name = `example.com`
- Master KDC = `kdc1.example.com`
- admin principal = `lucy/admin`
- Online help URL = `http://example:8888/ab2/coll.384.1/SEAM/@AB2PageView/6956`

Note – This procedure requires that DNS is running.

Note – Before you begin this configuration process, make a backup of the `/etc/krb5` files.

1. **Become superuser on the master KDC. (`kdc1`, in this example)**
2. **Edit the Kerberos configuration file (`krb5.conf`).**

Note – You need to change the realm names and the names of the servers. See the `krb5.conf`(4) man page for a full description of this file.

```
kdc1 # more /etc/krb5/krb5.conf
       [libdefaults]
               default_realm = EXAMPLE.COM

       [realms]
                       EXAMPLE.COM = {
                       kdc = kdc1.example.com
                       admin server = kdc1.example.com
               }

       [domain_realm]
               .example.com = EXAMPLE.COM

       [logging]
               default = FILE:/var/krb5/kdc.log
               kdc = FILE:/var/krb5/kdc.log

       [appdefaults]
           gkadmin = {
               help_url =
http://example:8888/ab2/coll.384.1/SEAM/@AB2PageView/6956
               }
```

In this example, the lines for `domain_realm`, `kdc`, `admin_server`, and all `domain_realm` entries were changed. In addition, the line with `___slave_kdcs___` in the `[realms]` section was deleted and the line that defines the `help_url` was edited.

3. **Edit the KDC configuration file (`kdc.conf`).**

   You must change the realm name. See the `kdc.conf`(4) man page for a full description of this file.

```
kdc1 # more /etc/krb5/kdc.conf
[kdcdefaults]
               kdc_ports = 88,750

       [realms]
               EXAMPLE.COM= {
                       profile = /etc/krb5/krb5.conf
                       database_name = /var/krb5/principal
                       admin_keytab = /etc/krb5/kadm5.keytab
                       acl_file = /etc/krb5/kadm5.acl
                       kadmind_port = 749
                       max_life = 8h 0m 0s
                       max_renewable_life = 7d 0h 0m 0s
Need moving ---------> default_principal_flags = +preauth
               }
```

   In this example, only the realm name definition in the `[realms]` section is changed.

4. **Create the KDC database by using the `kdb5_util` command.**

   The `kdb5_util` command, which is located in `/usr/sbin`, creates the KDC database. When used with the `-s` option, this command creates a stash file that is used to authenticate the KDC to itself before the `kadmind` and `krb5kdc` daemons are started.

```
kdc1 # /usr/sbin/kdb5_util create -r EXAMPLE.COM -s
Initializing database '/var/krb5/principal' for realm
'EXAMPLE.COM'
master key name 'K/M@EXAMPLE.COM'
You will be prompted for the database Master Password.
It is important that you NOT FORGET this password.
Enter KDC database master key: key
Re-enter KDC database master key to verify: key
```

Note – The `-r` option followed by the realm name is not required if the realm name is equivalent to the domain name in the server's name space.

---

5. **Edit the Kerberos access control list file (`kadm5.acl`).**

   Once populated, the `/etc/krb5/kadm5.acl` file contains all principal names that are allowed to administer the KDC. The first entry that is added might look similar to the following:

```
lucy/admin@EXAMPLE.COM    *
```

   This entry gives the `lucy/admin` principal in the `EXAMPLE.COM` realm the ability to modify principals or policies in the KDC. The default installation includes an asterisk (*) to match all admin principals. This default could be a security risk, so it is more secure to include a list of all of the admin principals. See the `kadm5.acl`(4) man page for more information.

6. **Edit the `/etc/gss/mech` file.**

   The `/etc/gss/mech` file contains the GSSAPI based security mechanism names, its object identifier (OID), and a shared library that implements the services for that mechanism under the GSSAPI. Change the following from:

```
# Mechanism Name        Object Identifier       Shared Library
Kernel Module
#
diffie_hellman_640_0    1.3.6.4.1.42.2.26.2.4   dh640-0.so.1
diffie_hellman_1024_0   1.3.6.4.1.42.2.26.2.5   dh1024-0.so.1
kerberos_v5             1.2.840.113554.1.2.2    gl/mech_krb5.so
gl_kmech_krb5
```

   To the following:

```
# Mechanism Name        Object Identifier       Shared Library
Kernel Module
#
kerberos_v5             1.2.840.113554.1.2.2    gl/mech_krb5.so
gl_kmech_krb5
diffie_hellman_640_0    1.3.6.4.1.42.2.26.2.4   dh640-0.so.1
diffie_hellman_1024_0   1.3.6.4.1.42.2.26.2.5   dh1024-0.so.1
```

7. **Run the** `kadmin.local` **command to create principals.**

   You can add as many admin principals as you need. But you must add at least one admin principal to complete the KDC configuration process. In the following example, `lucy/admin` is added as the principal.

```
kdc1 # /usr/sbin/kadmin.local
kadmin.local: addprinc lucy/admin
Enter password for principal "lucy/admin@EXAMPLE.COM":
Re-enter password for principal "lucy/admin@EXAMPLE.COM":
Principal "lucy/admin@EXAMPLE.COM" created.
kadmin.local:
```

8. **Create a** `keytab` **file for the** `kadmind` **service.**

   The following command sequence creates a special `keytab` file with principal entries for `lucy` and `tom`. These principals are needed for the `kadmind` service. In addition, you can optionally add NFS service principals, host principals, LDAP principals, and so on.

Note – When the principal instance is a host name, the fully qualified domain name (FQDN) must be entered in lowercase letters, regardless of the case of the domain name in the `/etc/resolv.conf` file.

```
kadmin.local: ktadd -k /etc/krb5/kadm5.keytab
kadmin/kdc1.example.com
Entry for principal kadmin/kdc1.example.com with kvno 3,
encryption type DES-CBC-CRC
                  added to keytab WRFILE:/etc/krb5/kadm5.keytab.
kadmin.local: ktadd -k /etc/krb5/kadm5.keytab
changepw/kdc1.example.com
Entry for principal changepw/kdc1.example.com with kvno 3,
encryption type DES-CBC-CRC
                  added to keytab WRFILE:/etc/krb5/kadm5.keytab.
kadmin.local:
```

Once you have added all of the required principals, you can exit from `kadmin.local` as follows:

```
kadmin.local: quit
```

9. **Start the Kerberos daemons as shown:**

```
kdc1 # /etc/init.d/kdc start
kdc1 # /etc/init.d/kdc.master start
```

Note – You stop the Kerberos daemons by running the following commands:

```
kdc1 # /etc/init.d/kdc stop
kdc1 # /etc/init.d/kdc.master stop
```

---

10. **Add principals by using the SEAM Administration Tool.**

    To do this, you must log on with one of the admin principal names that you created earlier in this procedure. However, the following command-line example is shown for simplicity.

```
kdc1 # /usr/sbin/kadmin -p lucy/admin
Enter password: kws_admin_password
kadmin:
```

11. **Create the master KDC host principal which is used by Kerberized applications such as** `klist` **and** `kprop`**.**

```
kadmin: addprinc -randkey host/kdc1.example.com
Principal "host/kdc1.example.com@EXAMPLE.COM" created.
kadmin:
```

12. **(Optional) Create the master KDC root principal which is used for authenticated NFS mounting.**

```
kadmin: addprinc root/kdc1.example.com
Enter password for principal root/kdc1.example.com@EXAMPLE.COM:
password
Re-enter password for principal
root/kdc1.example.com@EXAMPLE.COM: password
Principal "root/kdc1.example.com@EXAMPLE.COM" created.
kadmin:
```

13. **Add the master KDC's host principal to the master KDC's** `keytab` **file which allows this principal to be used automatically.**

```
kadmin: ktadd host/kdc1.example.com
kadmin: Entry for principal host/kdc1.example.com with
->kvno 3, encryption type DES-CBC-CRC added to keytab
->WRFILE:/etc/krb5/krb5.keytab
kadmin:
```

Once you have added all of the required principals, you can exit from `kadmin` as follows:

```
kadmin: quit
```

14. **Run the** `kinit` **command to obtain and cache an initial ticket-granting ticket (credential) for the principal.**

This ticket is used for authentication by the Kerberos v5 system. `kinit` only needs to be run by the client at this time. If the Sun ONE directory server were a Kerberos client also, this step would need to be done for the server. However, you may want to use this to verify that Kerberos is up and running.

```
kdclient # /usr/bin/kinit root/kdclient.example.com
Password for root/kdclient.example.com@EXAMPLE.COM: passwd
```

15. **Check and verify that you have a ticket with the** `klist` **command.**

The `klist` command reports if there is a `keytab` file and displays the principals. If the results show that there is no `keytab` file or that there is no NFS service principal, you need to verify the completion of all of the previous steps.

```
# klist -k
Keytab name: FILE:/etc/krb5/krb5.keytab
KVNO Principal
---- --------------------------------------------------------
3 nfs/host.example.com@EXAMPLE.COM
```

Note – The example given here assumes a single domain. The KDC may reside on the same machine as the Sun ONE directory server for testing purposes, but there are security considerations to take into account on where the KDCs reside.

With regards to the configuration of Kerberos v5 in conjunction with the Sun ONE Directory Server 5.2 software, you are finished with the Kerberos v5 part. It's now time to look at what is required to be configured on the Sun ONE directory server side.

Sun ONE Directory Server 5.2 GSSAPI Configuration

As previously discussed, the Generic Security Services Application Program Interface (GSSAPI), is standard interface that enables you to use a security mechanism such as Kerberos v5 to authenticate clients. The server uses the GSSAPI to actually validate the identity of a particular user. Once this user is validated, it's up to the SASL mechanism to apply the GSSAPI mapping rules to obtain a DN that is the bind DN for all operations during the connection.

The first item discussed is the new identity mapping functionality.

The identity mapping service is required to map the credentials of another protocol, such as SASL DIGEST-MD5 and GSSAPI to a DN in the directory server. As you will see in the following example, the identity mapping feature uses the entries in the `cn=identity` mapping, `cn=config` configuration branch, whereby each protocol is defined and whereby each protocol must perform the identity mapping. For more information on the identity mapping feature, refer to the Sun ONE Directory Server 5.2 Documents.

▼ To Perform the GSSAPI Configuration for the Sun ONE Directory Server Software

1. **Check and verify, by retrieving the `rootDSE` entry, that the GSSAPI is returned as one of the supported SASL Mechanisms.**

   Example of using `ldapsearch` to retrieve the rootDSE and get the supported SASL mechanisms:

```
$./ldapsearch -h directoryserver_hostname -p ldap_port -b ""
-s base "(objectclass=*)" supportedSASLMechanisms
supportedSASLMechanisms=EXTERNAL
supportedSASLMechanisms=GSSAPI
supportedSASLMechanisms=DIGEST-MD5
```

2. **Verify that the GSSAPI mechanism is enabled.**

   By default, the GSSAPI mechanism is enabled.

   Example of using `ldapsearch` to verify that the GSSAPI SASL mechanism is enabled:

```
$./ldapsearch -h directoryserver_hostname -p ldap_port
-D"cn=Directory Manager" -w password -b "cn=SASL, cn=security,cn=
config" "(objectclass=*)"
#
# Should return
#
cn=SASL, cn=security, cn=config
objectClass=top
objectClass=nsContainer
objectClass=dsSaslConfig
cn=SASL
dsSaslPluginsPath=/var/Sun/mps/lib/sasl
dsSaslPluginsEnable=DIGEST-MD5
dsSaslPluginsEnable=GSSAPI
```

3. **Create and add the GSSAPI `identity-mapping.ldif`.**

   Add the LDIF shown below to the Sun ONE Directory Server so that it contains the correct suffix for your directory server.

   You need to do this because by default, no GSSAPI mappings are defined in the Sun ONE Directory Server 5.2 software.

   Example of a GSSAPI identity mapping LDIF file:

```
#
dn: cn=GSSAPI,cn=identity mapping,cn=config
objectclass: nsContainer
objectclass: top
cn: GSSAPI

dn: cn=default,cn=GSSAPI,cn=identity mapping,cn=config
objectclass: dsIdentityMapping
objectclass: nsContainer
objectclass: top
cn: default
dsMappedDN: uid=${Principal},ou=people,dc=example,dc=com

dn: cn=same_realm,cn=GSSAPI,cn=identity mapping,cn=config
objectclass: dsIdentityMapping
objectclass: dsPatternMatching
objectclass: nsContainer
objectclass: top
cn: same_realm
dsMatching-pattern: ${Principal}
dsMatching-regexp: (.*)@example.com
dsMappedDN: uid=$1,ou=people,dc=example,dc=com
```

   It is important to make use of the `${Principal}` variable, because it is the only input you have from SASL in the case of GSSAPI. Either you need to build a `dn` using the `${Principal}` variable or you need to perform pattern matching to see if you can apply a particular mapping. A principal corresponds to the identity of a user in Kerberos.

Note – You can find an example GSSAPI LDIF mappings files in *ServerRoot*/*slapd-server*/`ldif/identityMapping_Examples.ldif`.

The following is an example using `ldapmodify` to do this:

```
$./ldapmodify -a -c -h directoryserver_hostname -p ldap_port
-D "cn=Directory Manager" -w password -f identity-mapping.ldif
-e /var/tmp/ldif.rejects 2> /var/tmp/ldapmodify.log
```

4. **Perform a test using** `ldapsearch`**.**

   To perform this test, type the following `ldapsearch` command as shown below, and answer the prompt with the `kinit` value you previously defined.

   Example of using `ldapsearch` to test the GSSAPI mechanism:

```
$./ldapsearch -h directoryserver_hostname -p ldap_port -o mech=GSSAPI
-o authzid="root/hostname.domainname@EXAMPLE.COM" -b ""
-s base "(objectclass=*)"
```

   The output that is returned should be the same as without the `-o` option.

Note – If you do not use the `-h` *hostname* option, the GSS code ends up looking for a `localhost`.*domainname* Kerberos ticket, and an error occurs.

TLSv1/SSL Protocol Support

This section discusses the Transport Layer Security (TLS) and how it provides the encrypted communications between two hosts, such as a directory server and client. The topic is covered in two categories:

- Server (directory server)
- Client (Secured LDAP Client)

To dispel any misunderstandings you might have, this section explains the differences between TLS and SSL.

SSL Background

The Secure Sockets Layer (SSL) was originally designed for the World Wide Web (WWW) environment to provide a secure channel between two machines. The SSL protocol has gone through various incarnations, beginning with version 1, and evolving into it's present state with its adoption be the Internet Engineering Task Force (IETF), which is now referred to as the Transport Layer Security (TLS) standard.

All the previous versions of the SSL protocol were developed by engineers who worked for Netscape Communications. Netscape's intention was to develop a security model whereby they could provide a single solution to address all the security issues around not only the Web, but also messaging, and news.

TABLE 3-2 shows the development cycle of SSL / TLS.

TABLE 3-2 SSL and TLS Development Cycle

SSL / TLS Protocol Version	Description
SSLv1	Developed by Netscape in 1994, but was never released.
SSLv2	Developed by Netscape in 1994, and the first release.
SSLv3	Developed by Netscape in 1995, and provided authentication only.
TLS	Adoption by the IETF in 1997. Provided new functionality such as a new MAC algorithm and new key expansion.

SSLv2

The goal of SSLv2 was to provide a secure channel between two hosts on the WWW environment. With this in mind, the SSL protocol needed to fit in well with the HTTP protocol, which is used by the Web. Netscape also wanted to provide a single security solution, which meant that this solution would have to work with other protocols and not just HTTP. Unfortunately, not all protocols use or require the same security properties.

SSLv3

There is no question that SSLv2 was widely adopted resulting in a great deal of popularity, thus ensuring that the design goals and principles of SSLv2 were carried forward into SSLv3. The main goal for SSLv3 was to fix a number of security

problems found in SSLv2. This meant designing a more secure model that could negotiate multiple cryptographic algorithms. The end result is that SSLv3 supports many more multiple cryptographic algorithms.

TLS Background

In 1996 the IETF chartered the Transport Layer Security (TLS) working group to attempt to standardize an SSL-like protocol. It became apparent early on that there was very little support for changing the existing SSLv3 protocol, with the exception of a few minor bugs and enhancements. To this end, the new protocol just became a minor cleanup of SSLv3.

Understanding TLSv1 Transport Support

In your directory server (LDAP) deployment, it is highly likely that you have some form of security requirements that must be addressed. Specific security requirements are different from one organization to another. For example, a directory server (LDAP) that is available on the Internet has very specific security needs.

To provide secure communications over a network, your directory server (in particular, the Sun ONE Directory Server), includes and supports the LDAPS communications protocol. LDAPS (LDAP over SSL) is the standard LDAP protocol, which runs on top of the Secure Sockets Layer (SSL).

It is possible to not only use TLSv1 protocol to secure communications between a directory server (LDAP) and directory clients (LDAP), but also between directory servers (LDAP) that are bound by a replication agreement, or between a database link and a remote database. You can use TLSv1 with Simple authentication (bind DN and password), or with certificate-based authentication.

Two kinds of authentication mechanisms can be performed using TLSv1:

- Server authentication
- Client authentication

With server authentication, the client decides whether it trusts the certificate presented by the server. With client authentication, the client decides whether it trusts the certificate presented by the server, and the server decides whether it trusts the certificate presented by the client. In the case of server authentication, the client is not authenticated to the server at all. In the case of client authentication, the client may be authenticated to the server if it also performs a bind using the SASL EXTERNAL mechanism.

Using TLSv1 with the Simple authentication mechanism guarantees confidentiality and data integrity. One of the benefits of using a certificate to authenticate to the Directory Server (LDAP) instead of a bind DN and password is improved security.

The use of the certificate-based authentication mechanism is more secure than non-certificate bind operations. This is because certificate-based authentication uses public-key cryptography. As a result, bind credentials cannot be intercepted across the network.

The Sun ONE Directory Server software is capable of simultaneous TLSv1 and non-SSL communications. This means that you do not have to choose between TLSv1 or non-SSL communications for your directory server, because you can use both at the same time.

But one of the downsides to using TLSv1 is reduced efficiency.

The process of data encryption, with generally DES or RC4 does significantly reduce the throughput that can be achieved, and the initial negotiation and key agreement with RSA or DSA is even more expensive. There are potentially three cases for which you must be considered when dealing with the performance impact of TLSv1 in the Sun ONE Directory Server 5.2 software:

- Clients establish a connection over TLSv1 and maintain that connection for a number of operations (persistent connections). This is the best scenario for TLSv1 because the process of initializing the TLSv1-based connection is the most expensive part. In this case, using TLSv1 will introduce degradation over the performance when not using TLSv1.
- Clients establish a connection over TLSv1, perform one or two operations (for example, a bind and a search), close the connection, and then repeat. That is, the same system or set of systems are repeatedly used to establish short-lived TLSv1-based connections. This is much less efficient because as indicated above, the process of establishing TLSv1-based connections is rather expensive. However, it is not as bad as it could be because the same TLSv1 session (which is also expensive to set up) can be re-used across multiple TCP connections.
- Different client systems establish a connection over TLSv1, perform one or two operations (for example, a bind and a search), and then close the connection. That is, different systems are used to establish short-lived SSL-based connections. This is the least efficient of the three scenarios because the TLSv1 sessions cannot be reused across the different TCP connections, which means that the full negotiation process must be performed for each new connection. This case is a very uncommon scenario for real-world directory use.

Why Use TLSv1?

TLSv1 is used to protect sensitive information. Data that travels over a network is visible to a number of other machines on that network. This is especially of concern for information traveling over the Internet. Normally, the other machines simply

ignore the information if it isn't intended for them, but that isn't necessarily the case. Most network interfaces support a feature known as promiscuous mode, in which they sue to pay attention to all traffic and not just information that pertains specifically to that machine. This can be a very helpful diagnostic feature for network administrators or even people that support a product that works in a networked environment. Applications like `snoop` (included in the Solaris OE) or the Network Monitor that comes with Windows NT, provide a mechanism for capturing and displaying that information. These applications are often called sniffers or protocol analyzers. More advanced protocol analyzers like Ethereal (available for free on a number of platforms, or as source code from `http://www.ethereal.com`) can even interpret the information that is captured so that it can be more easily understood by the user. This is helpful with text-based protocols like HTTP because it provides formatting for the request. It is invaluable for binary protocols like LDAP because otherwise the task of decoding the information and figuring out exactly what was going on between the client and the server is much more difficult.

Sniffers can be very helpful tools when trying to track down problems that are occurring in a networked environment. However, they can also be very helpful tools to those with less honorable intentions. They make it easy to see any information that is transferred over the network, so it is possible to capture sensitive information like credit card numbers being used to buy a product on the Web, passwords being used to bind to a directory server, or any other kind of information that would otherwise be protected. Using TLSv1 can thwart these attempts because the encrypted information is completely unintelligible except to the two machines that are having the conversation.

The layer of privacy provided by TLSv1 does not come without a price. Because TLSv1 is used to encapsulate information in another protocol, each machine must deal with the extra overhead of encrypting information before sending it over the network, and decrypting information received before attempting to interpret it. The primary form of overhead is in CPU utilization, but it is also necessary to transfer more information between the client and the server. For that reason, TLSv1 should generally be used only when it is necessary to ensure the privacy of the information that is being sent over the network.

How Does TLSv1 Work?

From a high level, TLSv1 works by encrypting information using data that is only available to the two machines having the encrypted conversation. The foundation of this set of information is the certificate. A certificate is a portion of binary data that can be used to establish proof of identity. There are two important parts of a certificate:

- The public key
- The private key

The public key is freely available and is used as the initial proof that the server is the system that the client believes it is. The private key is available only to the server and can be used to decrypt information that is encrypted using the server's public key.

Before information can be encrypted using TLSv1, a preliminary conversation must occur between the client and the server, known as the TLSv1 handshake, which is discussed in some detail later in this section. For now, this is what happens during the TLSv1 handshake:

1. The client sends some information to the server, including the TLSv1 version number that the client wants to use and some randomly generated data.
2. The server sends back some information that includes the TLSv1 version number the server will use, some randomly generated data, and the server's public key.
3. If the client decides to trust the server's certificate as proof of identification, it generates a shared secret. This shared secret is encrypted using the server's public key and sent to the server.
4. The server decrypts the data from the client using its private key to determine the shared secret.
5. All communication between the client and the server beyond that point is encrypted with that shared secret.

There is actually more that occurs during this TLSv1 handshake, but the above description is a good starting point.

Types of TLSv1

Two-types of TLSv1 are commonly used:

- Server authentication
- Client authentication

Server authentication is the most common form and is the most basic level of authentication that can be performed using TLSv1, and was explained in a brief description in the previous section. Essentially, server authentication is used to obtain enough information to get the shared secret to encrypt the information. It is called server authentication because the process involves a mechanism whereby the server sends proof of its identity to the client and the client is then able to decide whether to trust that information and continue its conversation with the server. In server authentication, the server automatically trusts the client, or trusts the client through some mechanism built into the encapsulated protocol (for example, the password used in an LDAP bind request). No proof of the client's identity is required in the TLSv1 handshake.

Client authentication extends the process of server authentication in that the server requires proof of the client's identity in addition to having to prove its identity to the client. In this case, the TLSv1 handshake is extended to include the server requesting that proof of identity from the client.

In this scenario, the client must have its own certificate, and send the public key to the server so that the server can determine whether to trust the identity of the client. The client does not require a certificate. This is only used when there is client authentication. There is also an additional step involved in the generation of the shared secret when client authentication is used.

FIGURE 3-3 shows that the TLS protocol runs above TCP/IP and below high-level application protocols.

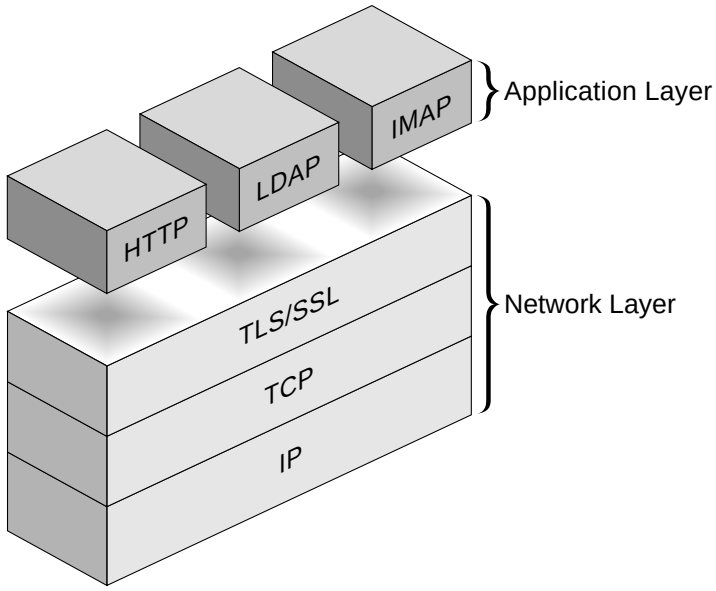

FIGURE 3-3 TLS Protocol in the Network Layer

TLS Protocol

The TLSv1 protocol is composed primarily of two subprotocols, which are:

- TLSv1 Record Protocol
- TLSv1 Handshake Protocol

The TLSv1 is the actual data transfer which is accomplished by the Record Protocol. This is achieved by breaking up the data stream to be transmitted into a series of fragments, with each fragment being independently protected and transmitted. Before any fragment can be transmitted, it must be protected against any potential

attack. To provide the integrity protection, a message authentication code (MAC) is computed over the data, and is transmitted along with the fragment. This MAC is appended to the fragment and the concatenated data and MAC are encrypted to form the encrypted payload. A header is then attached to the payload. It is the concatenated header and encrypted payload that is referred as a *record*, which is then actually transmitted. FIGURE 3-4 shows the record protocol.

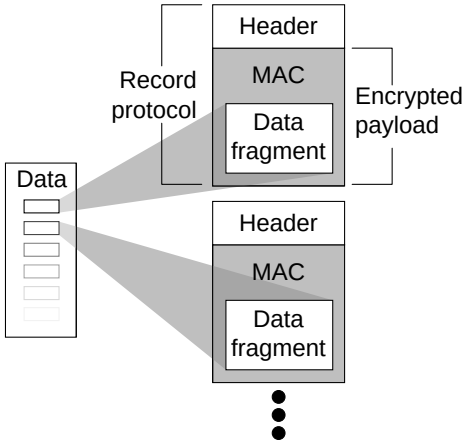

FIGURE 3-4 Composition of TLSv1 Record Protocol

The TLSv1 handshake protocol has various purposes, one of which includes the client and server negotiating on a set of algorithms which are used to protect the data. The client and server establish a set of cryptograhic keys which are used by these algorithms. The process works like this (also see FIGURE 3-5, and FIGURE 3-6 through FIGURE 3-10):

1. Server and client exchange hello messages
   a. Establish protocol version
   b. Set session ID
   c. Agree to use one of the following cipher suites, choosing the strongest cipher suite that is common between client and server:
      - Key Exchange Algorithm (public-key: RSA, DH)
      - Encryption Cipher
      - MAC Algorithm
   d. Establish compression method
   e. Exchange random values
2. Establish server authentication

a. Client confirms server's identity

b. Server sends certificate (necessary for KEA)

The TLS protocol supports a variety of different cryptographic algorithms, or ciphers, for use in operations such as authenticating the server and client to each other, transmitting certificates, and establishing session keys. Clients and servers can support different cipher suites, or sets of ciphers, depending on factors such as the version of TLS they support, company policies regarding acceptable encryption strength, and government restrictions on export of TLS-enabled software. The TLS handshake protocol determines how the server and client negotiate; which cipher suites they use, to authenticate each other, to transmit certificates, and to establish session keys. TABLE 3-3 describes the cipher suite algorithms.

TABLE 3-3 Cipher Suite Algorithms

DES	Data Encryption Standard, an encryption algorithm used by the U.S. Government
DSA	Digital Signature Algorithm, part of the digital authentication standard used by the U.S. Government
KEA	Key Exchange Algorithm, an algorithm used for key exchange by the U.S. Government
MD5	Message Digest algorithm developed by Rivest
RC2 and RC4	Rivest encryption ciphers developed for RSA Data Security
RSA	A public-key algorithm for both encryption and authentication. Developed by Rivest, Shamir, and Adleman
RSA key exchange	A key-exchange algorithm for SSL based on the RSA algorithm
SHA-1	Secure Hash Algorithm, a hash function used by the U.S. Government
SSHA	Salted Secure Hash Algorithm, a hash function used by the U.S. Government
SKIPJACK	A classified symmetric-key algorithm implemented in Fortezza-compliant hardware used by the U.S. Government
Triple-DES	DES applied three times

Key-exchange algorithms like KEA and RSA key exchange govern the way in which the server and client determine the symmetric keys they both use during a TLS session. The most commonly used TLS cipher suites use RSA key exchange.

c. Check certificate DN versus server's DN (used to protect against man-in-the-middle attack).

d. Premaster secret encrypted with server's public key (successful decryption by server provides additional authentication evidence)

e. Optionally check the host name used to connect to the server against the value of the CN attribute in the server certificate's subject DN. This is an optional step, whereby the host names are compared to host names and not to the DNs.

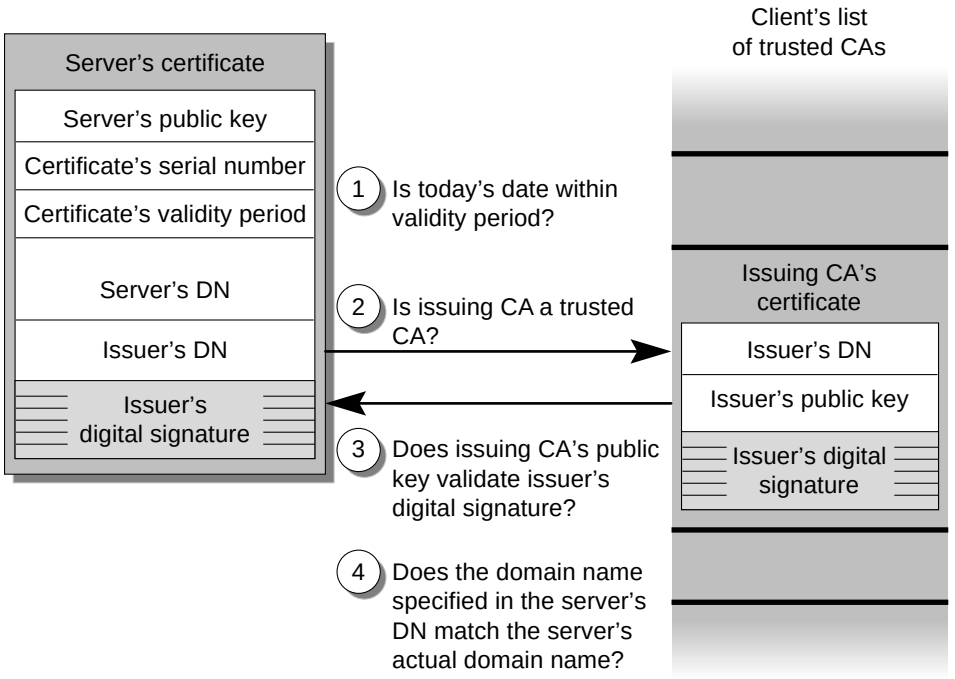

FIGURE 3-5 SSL Handshake Steps

3. Perform key exchange:

   a. Generate Pre-master secret

   b. Client and server share pre-master secret

   c. Secret shared using KEA

   d. RSA KEA example:

      - Client creates pre-master secret
      - Encrypt pre-master secret with server's public key
      - Server decrypts pre-master secret with its own private key

4. Perform client authentication (optional) – For client authentication, it is up to the client whether it wishes to authenticate itself to the server. The client sends its certificate which contains the client's name (and possibly alternate names) and the client's public key and the CA – which the server should trust.

   a. Server confirms client's identity

   b. Client sends certificate at server's request with the following:

      - Client's identity

      - Client's public key

   c. Client authenticates identity / public key binding. The ability to encrypt using the private key proves that the client is the owner of the certificate based on the assumption that only that user knows the private key.

   d. Digital signature: random data encrypted with client's private key to validate signature with client's public key

   e. Verify certificate within client's LDAP entry to allow certificate revocation. This is only done after the TLSv1 negotiation is complete, and then only if the client sends a SASL bind request using the EXTERNAL mechanism and the server is configured to verify the certificate presented by the client against the certificate stored in the user's entry. If this is not the case, then the client certificate is not verified against anything in the user's entry, nor is any attempt made to associate the client certificate with any user entry during the TLSv1 negotiation process.

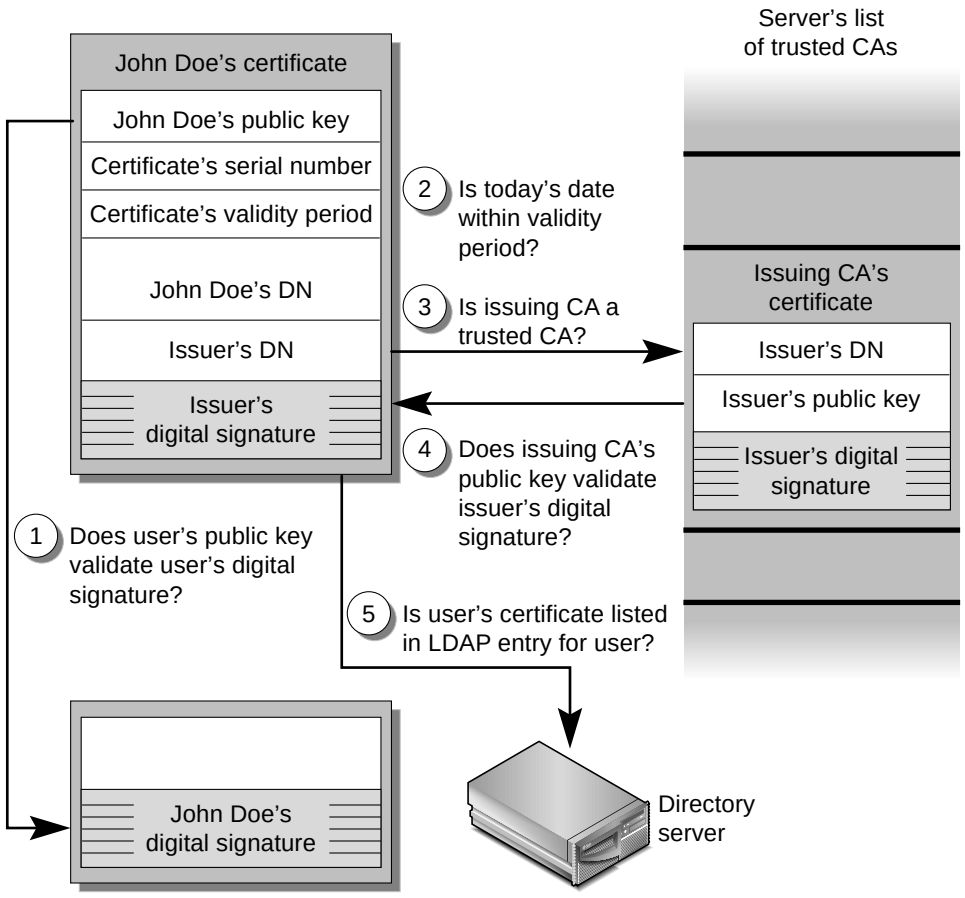

FIGURE 3-6 Client Authentication

5. Generate master secret (client and server)

   This is generated during first connection, and shared between connections.

6. Generate session keys (client and server, new for each connection)

7. Exchange cipher spec messages and finished messages

   a. Change cipher spec and announce start of encrypted message exchange

   b. Finished messages already encrypted with session key

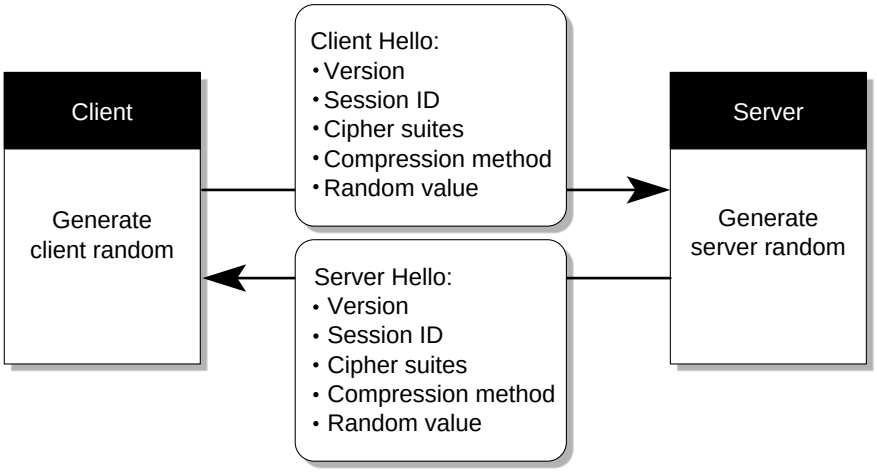

FIGURE 3-7 SSL Handshake Flow Chart (1 of 4)

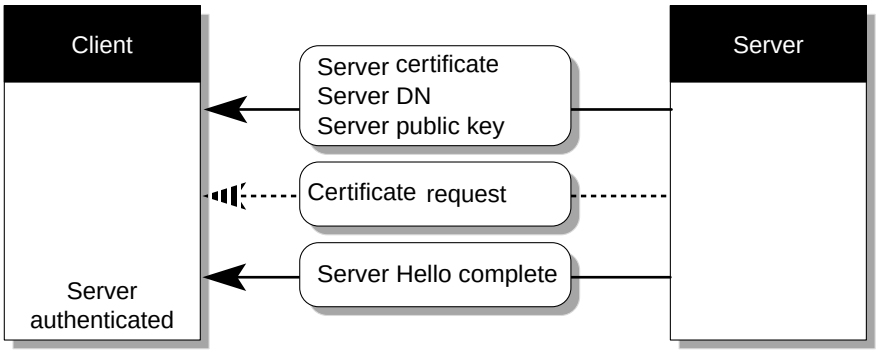

FIGURE 3-8 SSL Handshake Flow Chart (2 of 4)

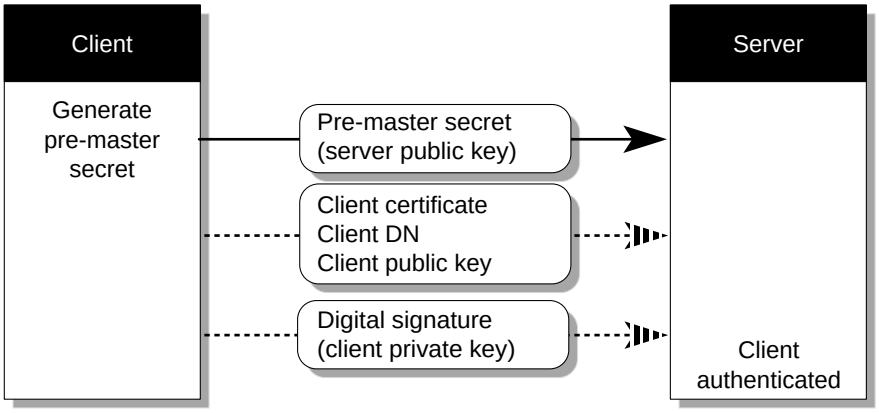

FIGURE 3-9 SSL Handshake Flow Chart (3 of 4)

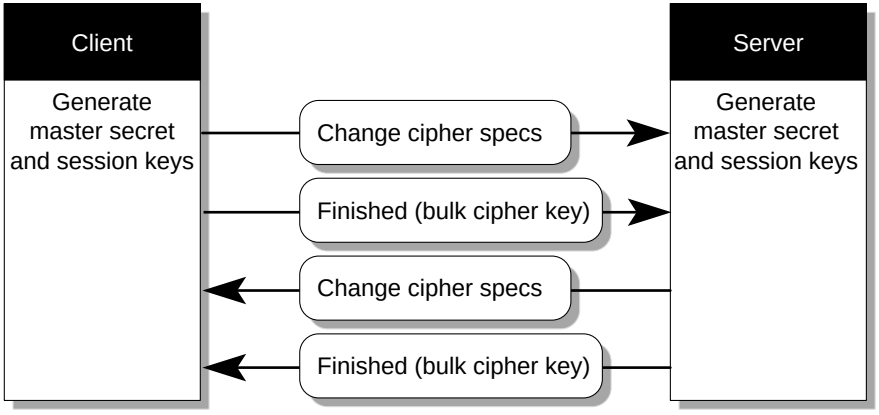

FIGURE 3-10 SSL Handshake Flow Chart (4 of 4)

TLSv1/SSL in the Sun ONE Directory Server 5.2 Software

The Sun ONE Directory server has the ability to support the TLSv1/SSL protocol in multiple areas, and can be enabled in the following situations:

- Both the administration server and DSML access are listening to HTTPS (HTTP over SSL). In this case HTTP over SSL refers to DSMLv2.
- Replication over SSL
- LDAP operations over SSL
- Chaining over SSL
- Console connected via SSL

The TLSv1/SSL Layer in the Sun ONE Directory Server 5.2 software is derived from the Network Security Services (NSS), and Netscape Portable Runtime (NSPR). NSS is a set of libraries designed to support cross-platform development of security-enabled server applications, and NSPR which provides low-level cross-platform support for operations such as threading and I/O. It is the Network Security Services that provides support for SSLv2, SSLv3, and TLSv1 and other security standards.

TABLE 3-4 lists the NSS and NSPR versions that are a component of the Sun ONE Directory Server 5.2 software and indicates where you can find more information.

TABLE 3-4 NSS and NSPR versions

Component	Version	Additional Information
Network Security Services (NSS)	3.3.4	`http://www.mozilla.org/projects/security/pki/nss`
Netscape Portable Runtime (NSPR)	4.1.4	`http://www.mozilla.org/projects/nspr`

FIGURE 3-11 shows a simplified view of the relationships between the NSS and NSPR shared libraries.

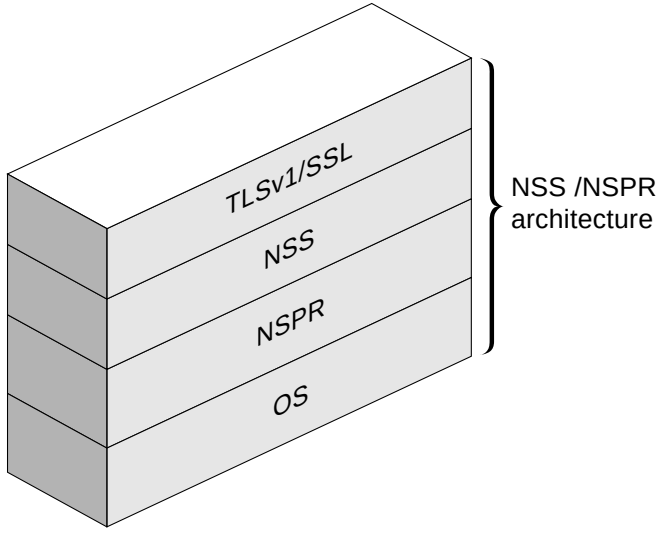

FIGURE 3-11 NSS/NSPR Architecture

The Network Security Services (NSS) has various security tools available, which may prove useful in the debugging and managing of your TLSv1/SSL implementation.

TLSv1/SSL Tools

Before we take a look at some of the TLSv1/SSL tools (TABLE 3-5), please be aware that these tools are not integrated into the Solaris 9 OE. You can get these tools from the Sun ONE Directory Server Resource Kit (SDRK) which is available from the Sun Download Center:

`http://wwws.sun.com/software/download/products/sunone`

Tools such as `ssltap` and many others can be obtained from:

`http://www.mozilla.org/projects/security/pki/nss/tools/`

This section describes some of the security certificate databases such as `cert7.db`, `key3.db`, and the `secmod.db`, which can list, generate, modify, or delete certificates within the `cert7.db` file and create or change the password, generate new public and private key pairs, display the contents of the key database, or delete key pairs within the `key3.db` file.

Security Databases

Public Key Cryptography Standard (PKCS) #11 specifies an API used to communicate with devices that hold cryptographic information and perform cryptographic operations. PKCS #11 supports a wide range of hardware and software devices intended for such purposes.

A PKCS #11 module (also referred to as a cryptographic module or cryptographic service provider) manages cryptographic services such as encryption and decryption through the PKCS #11 interface. PKCS #11 modules can be thought of as drivers for cryptographic devices that can be implemented in either hardware or software.

A PKCS #11 module always has one or more slots, which can be implemented as physical hardware slots in some form of a physical reader (for example, smart cards) or as conceptual slots in software. Each slot for a PKCS #11 module can contain a token, which is the hardware or software device that actually provides cryptographic services and optionally stores certificates and keys. The following is a brief explanation of the security databases:

- `cert7.db` – The database where the certificates (and therefore the public keys attached to them) are stored. Each certificate has a series of flags that account for the role that every certificate can take, as well as the uses for which that certificate is to be trusted. You can list, add, and modify the certificates within the database with `certutil`.
- `key3.db` – The database where the private keys associated to the public keys of the user certificates in `cert7.db` are kept. This file is protected by a password or pin.

- `secmod.db` – The file that keeps track of the available security modules. It lists each module with its slots and tokens, and specifies a default module. This is where an external module must be declared for the server to be able to detect it. By default, `secmod.db` manages two modules; the internal software module, and the built-in CA certificate module. `modutil` is the tool used to add, list, and modify modules to `secmod.db`.

TABLE 3-5 TLSv1/SSL Tools

Tool	Description
`certutil` 2.0	Manages certificate and key databases (`cert7.db` and `key3.db`).
`dbck` 1.0	Analyzes and repairs certificate databases.
`modutil` 1.1	Manages the database of PKCS#11 modules (`secmod.db`). The PKCS#11 modules refer to hardware encryption tokens like SSL accelerator cards or secure key storage mechanisms. Adds modules and modifies the properties of existing modules (such as whether a module is the default provider of some crypto service).
`pk12util` 1.0	Imports and exports keys and certificates between the cert/key databases and files in PKCS#12 format.
`ssltap` 3.2	Performs proxy requests for an SSL server and for contents of the messages exchanged between the client and server. The `ssltap` tool does not decrypt data, but it shows things like the type of SSL message (`clientHello`, `serverHello`, and so forth) and connection data (protocol version, cipher suite, and so forth). This tool is very useful for debugging.

Note – The Sun Crypto Accelerator 1000 card is the only PKCS#11 module that is officially supported in the Sun ONE Directory Server 5.2 software.

TLSv1/SSL Configuration Overview

Before implementing the TLSv1/SSL functionality, you need to be aware of the attributes and attribute values that are or may be required by the Sun ONE Directory Server 5.2 software for the following attributes:

- Certificate name
- Supported cipher suites
- Cryptographic token
- Optionally, secure port number

When using the Sun ONE Directory Server software, you also need to be aware of the following configuration entries:

- `dn: cn=config`
- `dn: cn=encryption, cn=config`

In the `cn=config` entry, pay particular attention to the `nsslapd-security` and `nsslapd-secureport` attributes. The `nsslapd-security` attribute enables the use of the security features (TLSv1/SSL and attribute encryption) in the Sun ONE Directory Server 5.2 software. If you require secure connections, or the use of the attribute encryption feature, this attribute must be set to `on`. With the `nsslapd-secureport`, you must select the TCP/IP port number that will be used for TLSv1/SSL communications. The default TCP/IP port number is 636, and is only used if the server has been configured with a private key and a certificate; otherwise the server does not listen on this port.

In the `cn=encryption`, `cn=config` entry, there are a number of attributes and attribute values to deal with such as specifying the support for a particular SSL version. The `nsSSL2` attribute supports SSLv2, while the `nsSSL3` attribute supports SSLv3. Both of these attributes can be set to `on` or `off`.

The `nsSSLSessionTimeout` specifies the session lifetime in the session cache (in seconds). The Default is `0` (zero), which results in the following:

- SSLv2 =100 sec
- SSLv3 =24 h

The `nsSSLClientAuth` attribute has the following values associated with a TLSv1/SSL connection:

- `Off` – no client authentication
- `Allowed` – request certificate, no error if no client certificate received
- `Required` – request certificate, error if no client certificate received
- `Default` – allowed

Next is the `nsSSLServerAuth` attribute, which stipulates the action that the TLSv1/SSL client should take on the server certificate sent by the TLSv1/SSL server in a TLSv1/SSL connection. The points of interest are:

- `Weak` – accept a server's certificate without checking the issuers CA
- `Cert` – accept server's certificate if the issuer CA is trusted
- `cncheck` – accept the server's certificate if:
  - The issuer CA is trusted
  - The certificate's `cn` matches the server's DNS host name
- `Default` – `cert`

The following is an example of the cn=encryption,cn=config entry.

```
# Example of using ldapsearch to show the cn=encryption,cn=config
# entry.
$./ldapsearch -h directoryserver_hostname -p ldap_port -b "cn=
encryption,cn=config" "(objectclass=*)"

#
# Should return
#

cn=encryption,cn=config
objectClass=top
objectClass=nsEncryptionConfig
cn=encryption
nsSSLSessionTimeout=0
nsSSLClientAuth=allowed
nsSSLServerAuth=cert
nsSSL2=off
nsSSL3=off
```

The nsSSL3Ciphers is a multi-valued attribute that specifies a set of encryption ciphers that the Sun ONE Directory Server 5.2 software will use during TLSv1/SSL communications. The following values are to be noted as interest:

- enable/disable – to enable and disable cipher suites
- +all – enable all cipher suites but rsa_null_md5
- -all – disable all cipher suites
- cipher_suite – enable or disable a cipher suite. Use the following syntax:

  +cipher_suite1,-cipher_suite2,...
- Default – all but rsa_null_md5 enabled

The nsKeyfile attribute provides the following:

- Key database path relative to the SERVER_ROOT
- Key database usually in the alias directory (alias/slapd-*instancename*-key3.db)

The nsCertfile attribute provides the following:

- Certificate database path relative to the directory server installation directory (/var/Sun/mps by default)
- Certificate database usually in the alias directory (alias/slapd-*instancename*-cert7.db)

The nsSSLToken specifies where the certificate will be stored. In the vast majority of installations, this will be internal (software), which means that the certificate will be contained in the *cert7.db and *key3.db database files. However, it can be different if the certificate is stored elsewhere (it will be something like username@realm if you are using the Sun™ Crypto Accelerator 1000). The nsSSLToken attribute can have the following values:

- Token for the cryptograhic operations: internal / external
- Default: internal (software)

The nsSSLPersonalityssl attribute specifies the nickname of the certificate that is used as the TLSv1/SSL certificate for the directory server. It is generally something like server-cert. The nsSSLPersonalityssl attribute has the following values:

- Certificate name
- If external, token certname:tokenname - Even with external tokens, the format is still just the nickname of the certificate. This is true at least with the SCA 1000 card, which is the only external token supported in the 5.2 directory server. However, the nsSSLToken may be different for external tokens.

The nsSSLActivation attribute indicates whether the associated cipher family should be considered enabled for use. Given that there will generally only be a single cipher family (RSA), then it should be on if you want to use TLSv1/SSL in the directory server. Finally, the nsSSLActivation has the following values:

- on (the default)
- off

Now that we have covered the essential attributes and their values, we can now take a look at how we can now enable TLSv1/SSL in the Sun ONE Directory Server 5.2 software

Enabling TLSv1/SSL in the Sun ONE Directory Server 5.2 Software

This section describes the process of creating a certificate database, obtaining and installing a certificate for use with your Sun ONE Directory Server 5.2 software, and configuring the Sun ONE Directory Server 5.2 software to trust the certification authority's (CA) certificate. There are two methods you can use to perform these tasks. One method uses the Console, the other is through the command line. Both methods are covered in this chapter.

The following process is necessary before you can turn on TLSv1/SSL in the Sun ONE Directory Server software.

1. Obtain and install a certificate for your directory server, and configure the directory server to trust the certification authority's certificate. See "Obtaining and Installing Server Certificates" on page 113.
2. Turn on TLSv1/SSL in the Sun ONE Directory Server 5.2 software. See "Activating TLSv1/SSL in the Sun ONE Directory Server 5.2 Software" on page 126.
3. (Optional) Ensure that each user of the directory server obtains and installs a personal certificate for all clients that will authenticate using TLSv1/SSL. This procedure is not covered in this book.

Note – LDAPS implicitly requires you to have a secure port to listen to. With the Start TLS operation, this is no longer a requirement.

Obtaining and Installing Server Certificates

You must perform the following tasks to obtain and install server certificates.

- "Task 1: Generate a Certificate Request (Console)" on page 113
- "Task 2: Obtain the Certificate From a Certificate Authority (CA)" on page 118
- "Task 3: Install the Certificate" on page 121
- "Task 4: Trust the Certificate Authority" on page 123
- "Task 5: Confirm That Your New Certificates Are Installed" on page 124

These tasks use wizards where possible. You can accomplish the same objectives on the command line by performing the following procedure:

- "To Obtain and Install Server Certificates Using the Command-Line Interface" on page 124

For testing purposes, you can generate a self-signed certificate as described in:

- "To Generate a Self-Signed Certificate Request" on page 125

▼ Task 1: Generate a Certificate Request (Console)

1. **On the Sun ONE Directory Server Console, select the Tasks tab and click Manage Certificates.**

   If this is the first time that you've opened this window, you are asked to assign a password to protect the key db as shown in FIGURE 3-12. This password is required to start the directory server when TLSv1/SSL is enabled.

   The Manage Certificates window is displayed (FIGURE 3-13).

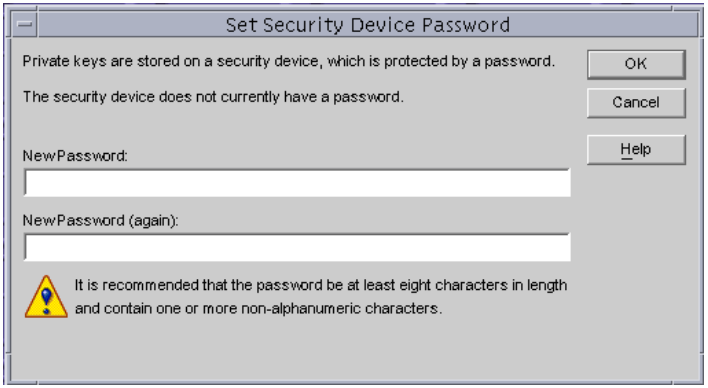

FIGURE 3-12 Console Security Device Password Window

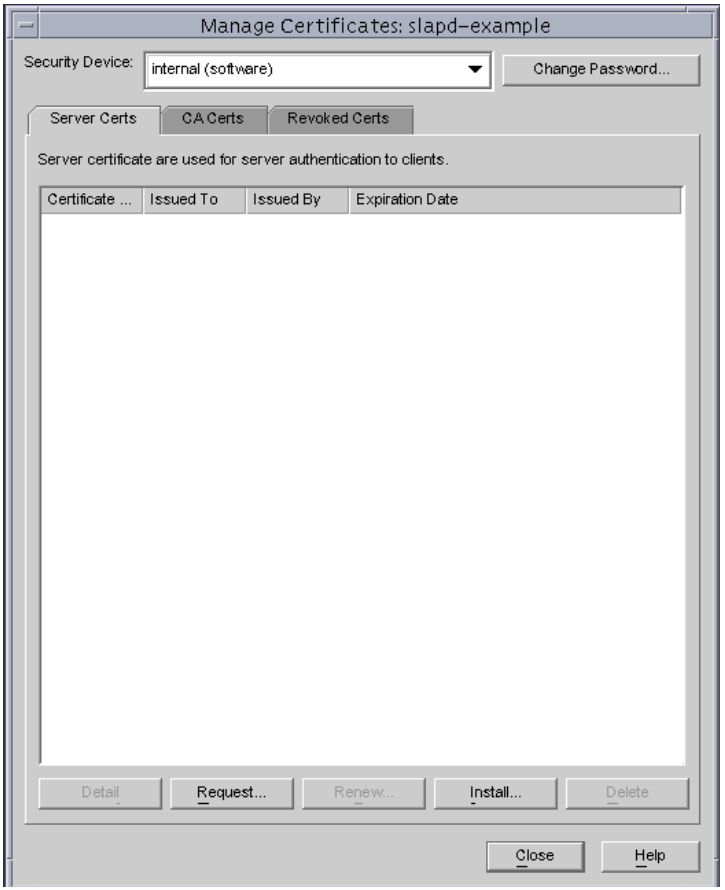

FIGURE 3-13 Manage Certificates Window

2. **Select the Server Certs tab, and click the Request button.**

   The Certificate Request Wizard is displayed (FIGURE 3-14).

3. **Click Next.**

4. **Enter the Requestor Information in the text fields (**FIGURE 3-14**), then click Next.**

   The following information is required:

   - Server Name – Enter the fully qualified host name of the directory server as it is used in DNS lookups, for example, `blueprints.example.com`.
   - Organization – Enter your organization name.
   - Organizational Unit – Enter your organizational unit information.
   - City / locality – Enter your city name.
   - State / province – Enter the full name of your state or province (no abbreviations).
   - Country / region – Select the two-character abbreviation for your country's name (ISO format). The country code for the United States is US, Great Britain is GB, Holland is NL, Singapore is SG, and so on.

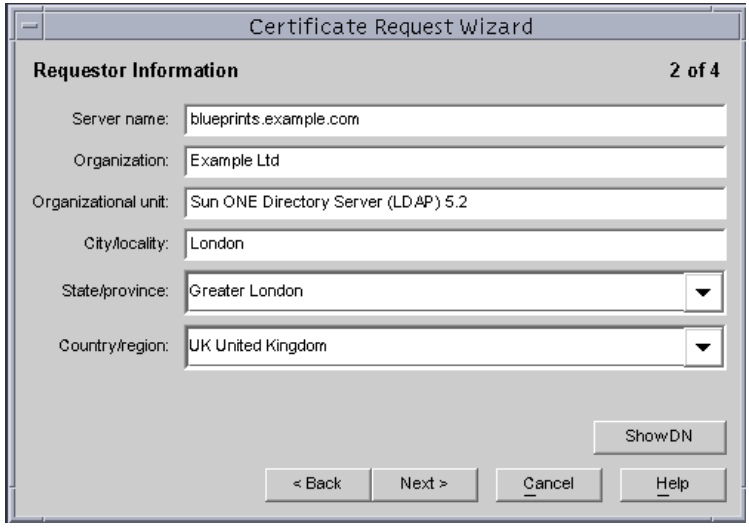

FIGURE 3-14 Certificate Request Wizard – Requestor Information Dialog Box

5. **In the Token Password dialog box (**FIGURE 3-15**), enter the password that will be used to protect the private key, and click Next.**

Note – The Next button is greyed out until you supply a password. When you click Next, the Request Submission dialog box is displayed.

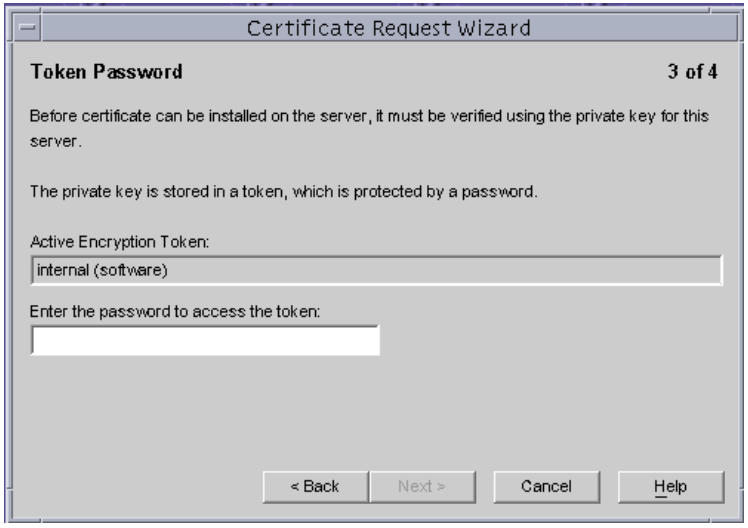

FIGURE 3-15 Certificate Request Wizard – Token Password Dialog Box

6. **In the Request Submission dialog box (**FIGURE 3-16**), select Copy to Clipboard to copy the certificate request information that you will send to the Certificate Authority.**

FIGURE 3-16 Certificate Request Wizard – Request Submission Dialog Box

Example of the PKCS #10 Request:

```
# Example PKCS #10 Request
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIB3jCCAUcCAQAwgZ0xCzAJBgNVBAYTAkdCMRcwFQYDVQQIEw5HcmVhdGVyIExv
bmRvbjEPMA0GA1UEBxMGTG9uZG9uMRgwFgYDVQQKEw9CbHVlUHJpbnRzIEluYy4x
KDAmBgNVBAsTH1N1biBPTkUgRGlyZWN0b3J5IChMREFQKSBTZXJ2ZXIxIDAeBgNV
BAMTF3Npcm9lLnVrLmJsdWVwcmludHMuY29tMIGfMA0GCSqGSIb3DQEBAQUAA4GN
ADCBiQKBgQDFSPHLHPJaFXDIiLphKhaDOBB4QAOUr40o8QIVB4gzsRVtPxeGUuy8
o+mGprCpgXpu0fNG5v8tgjiv4pzFL+r1UjJrTWQTLWMO6znGuAufR35B//nO2e6d
GQQvvYAPcxQFTOcfXcmJuoDyfR38DkGbVUdHFpa3ELADTnd2HGW/NQIDAQABoAAw
DQYJKoZIhvcNAQEEBQADgYEAeITbrpLpG0ODyJmLh1eQMCM1ZgD2A7v9I5q1eDWz
xiWZVMPXPzmMFXjA+YOnfBd/UGBCHF6cmNCoTugolsGhir3dTIjACsoStcNf8x1P
IfCkUZ0C6pQBOIbb1ochcojU8Al6jd2s26vhC+6xmEwf9Z3vfLcI/1mevQ8HCC8n
uBM=
-----END NEW CERTIFICATE REQUEST-----
```

You use this generated request to request a certificate from a certificate authority (the next task).

▼ Task 2: Obtain the Certificate From a Certificate Authority (CA)

In this task you have a couple of options:

- Send the previously generated certificate request to a certificate authority in email, or
- Go to a Certificate Authority web site and paste your request.

The outcome should be the same with either choice. That is, the CA will send you a certificate through e-mail. It is worth noting that the way to request a certificate from a CA varies depending on the CA.

The Sun ONE Certificate Manager software can be used to manage and sign digital certificates. This procedure uses the Sun ONE Certificate Manager as the CA. For small organizations, self-signed certificates are acceptable. However, for enterprise configurations, using an established CA is recommended.

Note – Sun ONE Certificate Manager version 4.7 is used in this procedure.

---

1. **Using a browser, go to the Sun ONE Certificate Manager secure URL.**

   This example uses `https://example.blueprints.com:443/`

2. **Select SSL Application Server under the Enrollment tab.**

3. **Copy the certificate request from the directory server, and paste it into the certificate manager system (in the TLSv1/SSL Server PKCS#10 request area, as shown in** FIGURE 3-17**):**

Note – It is advisable to have multiple Registration Authorities (RA) that you can select from, for example, BluePrints Inc. CA for production systems, and BluePrints Inc. TEST CA for development and test systems.

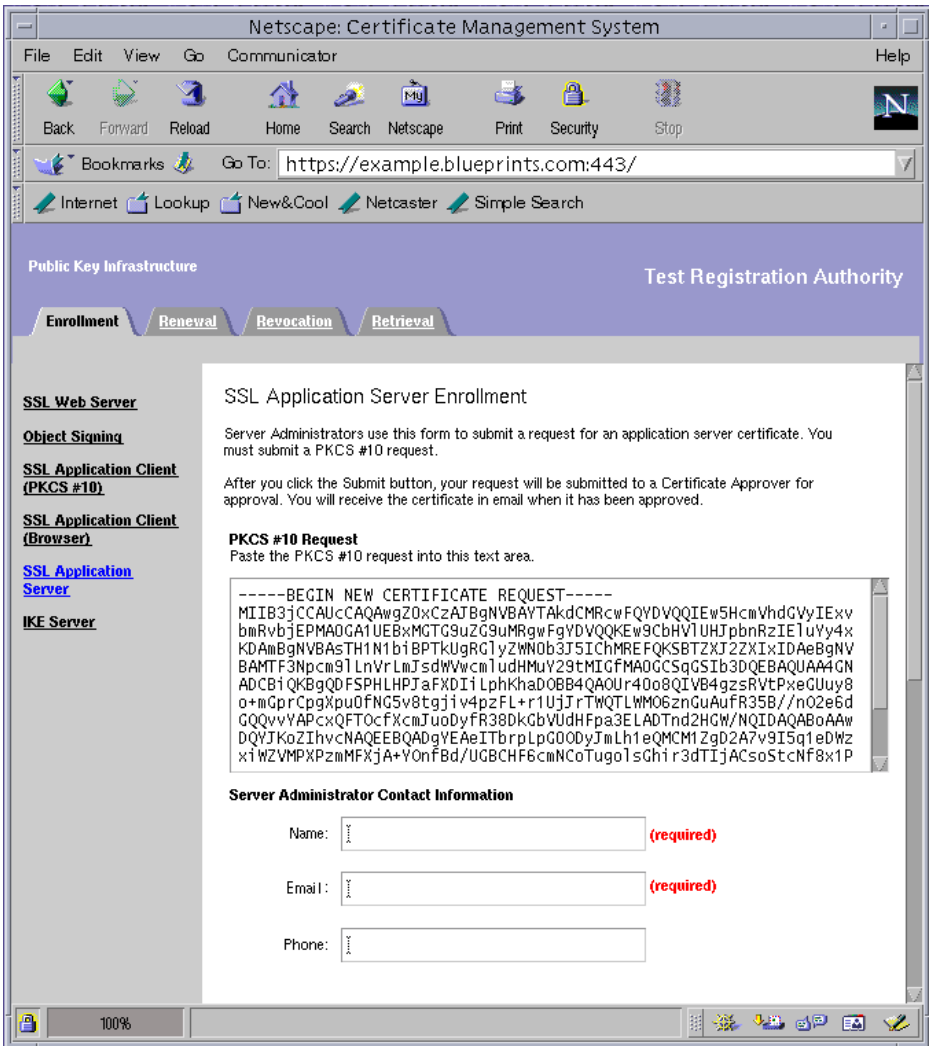

FIGURE 3-17 Sun ONE Certificate Manager Test Registration Authority

---

Note – Be sure to Include the “-----BEGIN CERTIFICATE-----” and “-----END CERTIFICATE-----” tags in the information pasted into the certificate install wizard.

---

4. **Fill out all the other information required by the Certificate Authority (CA) as described in** TABLE 3-6**.**

TABLE 3-6 Information Required by the CA

Name	Enter your name here
Email	Enter your email address.
Additional Comments	Enter your request details here. As an example, your request may be for enabling TLSv1/SSL on your own Enterprise Directory Service Infrastructure.
Revocation Password	Enter the database PIN for this system

The revocation password is needed if a certificate holder wants the certificate revoked, but does not have access to the private key in order to sign the revocation request.

5. **Click Submit at the bottom of the page.**

The Certificate Request Result screen appears (FIGURE 3-18), confirming that the request has been submitted. Note the request ID provided in the response message in the example below (You can use it later to retrieve the certificate, once the certificate has been issued).

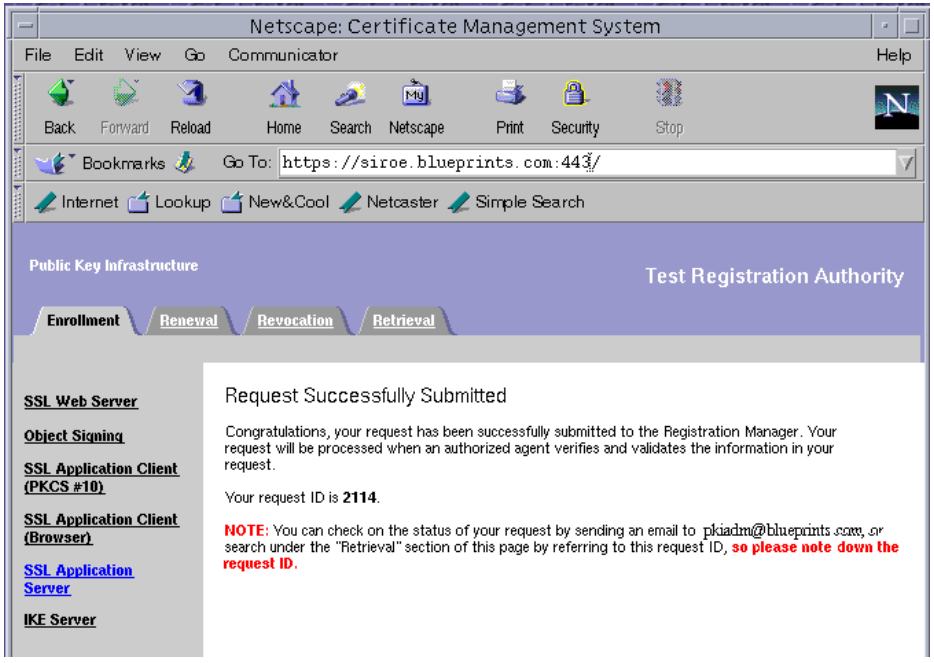

FIGURE 3-18 Sun ONE Certificate Manager Test Registration Authority

Next, your request gets added to the agent queue of the Certificate Manager for approval by that Certificate Manager's agent. If you have permissions to access that Certificate Manager's Agent interface, you can issue the certificate. Otherwise, you should wait for the other agent to approve the request you submitted and issue the certificate.

The Sun ONE Certificate Manager administrator must approve the request by going to the services URL. Example: `http://example.blueprints.com:8100`

Once you are in the Certificate Request Approval Page, you can perform the following to view that status of your request:

1. On the Certificate Management Retrieval Services menu, click Check Request Status.
2. Enter the request ID that was given to you when you submitted the initial request.

The following is an example of the information returned to you:

Request: 2114
Submitted on: 4/8/2003 12:38:59
Status: pending

▼ Task 3: Install the Certificate

This task is dependent on receiving an email from your CA with instructions on how to pick up your certificate (usually the CA provides you with a URL).

1. **Using a browser, go to the URL that was provided by the CA.**
2. **In the Sun ONE Directory Server Console, select the Tasks tab and click Manage Certificates.**

   The Manage Certificates window is displayed.
3. **Select the Server Certs tab, and click Install.**

   The Certificate Install Wizard is displayed.
4. **Select In the following encoded text block.**
5. **Copy and Paste the base-64 formatted certificate.**

   See FIGURE 3-19 for an example of entering the certificate into the Certificate Wizard.

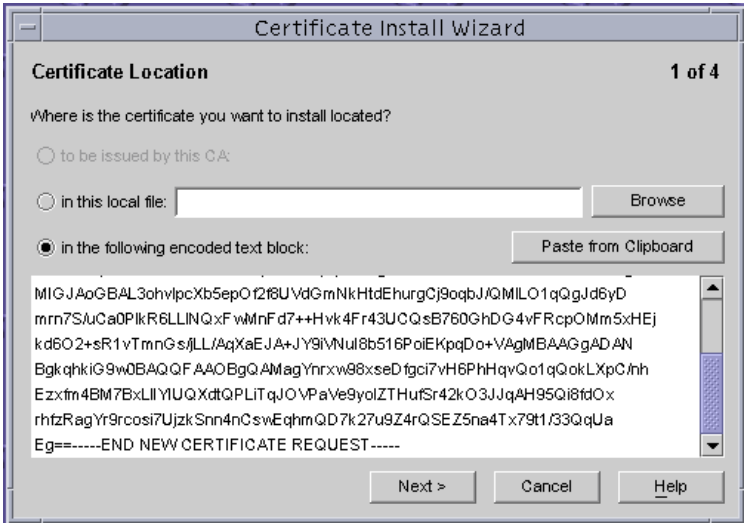

FIGURE 3-19 Certificate Install Wizard

6. **Check that the certificate information you have pasted is displayed correctly, and click Next.**

   The new TLSv1/SSL server certificate appears in the list of Server Certs in the Certificate Manager Window (FIGURE 3-20).

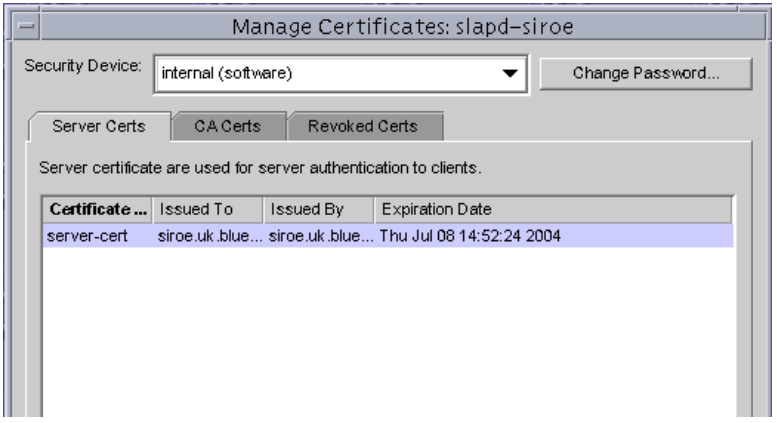

FIGURE 3-20 List of Server Certificates

▼ Task 4: Trust the Certificate Authority

Configuring the Sun ONE Directory Server 5.2 software to trust the certificate authority consists of obtaining your CA's certificate and installing it into your server's certificate database. Once you have the CA certificate, you can use the Certificate Install Wizard to configure the Sun ONE Directory Server software to trust the Certificate Authority.

1. **On the Sun ONE Directory Server 5.2 Console, select the Tasks tab and click Manage Certificates.**

   The Manage Certificates window is displayed.

2. **For each of the CA certs, go to the CA Certs tab, and click Install.**

   The Certificate Install Wizard is displayed.

3. **Locate the CA certificate (or CA certificate chain in its base-64 encoded format), and select Retrieval and then Import CA Certificate Chain.**

4. **Select Display Certificates in the CA Certificate Chain for Importing Individually into a Server and then click Submit.**

5. **Copy the text of the base 64 cert.**

   Copy the section between "-----BEGIN CERTIFICATE-----" and "-----END CERTIFICATE-----" and paste it into the window.

Note – Repeat this install for each of the Certificates (sections between "-----BEGIN CERTIFICATE-----" and "-----END CERTIFICATE-----") present on the page.

---

6. **Check that the certificate information that is displayed is correct, and click Next.**

7. **Select both of the following as the purpose of trusting this Certificate Authority**:

   Accepting connections from clients (Client Authentication). The server checks that the client's certificate has been issued by a trusted Certificate Authority.

   Accepting connections to other servers (Server Authentication). The server checks that the directory to which it is making a connection (for example, for replication updates) has a certificate that has been issued by a trusted Certificate Authority.

8. **Click Done to dismiss the wizard.**

   Once you have installed your certificate and trusted the CA's certificate, you are ready to activate TLSv1/SSL. However, before proceeding, you should first make sure that the certificates have been installed correctly.

▼ Task 5: Confirm That Your New Certificates Are Installed

1. **On the Sun ONE Directory Server Console, select the Tasks tab and click Manage Certificates.**

   The Manage Certificates window is displayed.

2. **Select the Server Certs tab.**

   A list of all the installed certificates for the server are displayed.

3. **Scroll through the list. You should find the certificates that you have installed.**

   The Sun ONE Directory Server software is now ready for TLSv1/SSL activation.

Using the Command Line to Obtain and Install Server Certificates

The following steps are performed on the command line instead of through a GUI. The five steps listed here have the same results as all the tasks that use the GUI.

▼ To Obtain and Install Server Certificates Using the Command-Line Interface

1. **If you do not have** `certutil` **in you current PATH, you must either change directory to** *server-root*`/shared/bin` **or set the** `LD_LIBRARY_PATH` **shell variable to** *server-root*`/lib`**.**

2. **Run the** `certutil` **command to generate a Certificate Signing Request (CSR) as shown:**

   The `certutil` binary is located in *server-root*`/shared/bin`**.**

   ```
   $cd server-root/shared/bin
   $./certutil -R -s subject -a -d cert-dir -P "slapd-instancename-"
   ```

3. **Take this output and provide it to a third-party CA to be signed (just like you would with the request generated through the Sun ONE Directory Server Console).**

4. **To install the certificate once it has been signed use the** `certutil` **command as shown**:

```
$cd server-root/shared/bin
$./certutil -A -n server-cert -t Pu,Pu,Pu -a -i certfile -D cert-dir -P
"slapd-instancename-"
```

5. **Import the CA certificate withe the** `certutil` **command**:

```
$cd server-root/shared/bin
$./certutil -A -n ca-cert -t CT,CT,CT -a -i ca_certfile -D cert-dir -P
"slapd-instancename-"
```

▼ To Generate a Self-Signed Certificate Request

Use this procedure when you want to test TLSv1/SSL without using a real CA.

1. **If you do not have** `certutil` **in you current PATH, you must either change directory to** *server-root*`/shared/bin` **or set the** `LD_LIBRARY_PATH` **shell variable to** *server-root*`/lib`**.**

2. **Issue the** `certutil` **command as shown**:

```
$cd server-root/shared/bin
$ ./certutil -N -d <serveroot>/alias -P "slapd-example -”
```

Running the above command will request that a password is given to protect the key db.

Note – The password must be at least eight characters long, and must contain at least one non-alphabetic character.

You are prompted for a password to protect the private key store, which you should provide, and then it will create a new certificate database in the *server-root*`/alias` directory.

3. **Issue the following command**:

```
$cd server-root/shared/bin
$ ./certutil -S -n "server-cert" -s subject -t CTPu,CTPu,CTPu -x -v
12 -d <serveroot>/alias -P "slapd-example -" -5
```

The `-t` option specifies the trust attributes to modify in an existing certificate or to apply to a certificate when creating it or adding it to a database. In this example, we used `CTPu`, where `C` is the trusted CA to issue server certificates (SSL only), where `T` is the trusted CA to issue the client certificates to, where `P` is the trusted peer, and `u` is the certificate that can be used for authentication or signing. For more information on `certutil`, refer to:
`http://www.mozilla.org/projects/security/pki/nss/tools/certutil.html`

4. **Respond as prompted.**

   You are asked to randomly enter a text until the progress meter is full, then asked for the password for the private key store (the one you just used above). You receive a list of options for certificate extensions. Enter a `1` to indicate that it is an SSL server, followed by a `9` to indicate that there are no more extensions. Say `y` to indicate that it is a critical extension.

   After completing the above successfully, you can use the certificate in this database for your directory server. It is self-signed, and nothing will trust it by default, so you must use that newly-created database as the trust store as well.

   The following databases are also created:

   - `slapd-example-cert7.db`
   - `slapd-example-key3.db`

Activating TLSv1/SSL in the Sun ONE Directory Server 5.2 Software

Before you can activate TLSv1/SSL, you must create a certificate database, obtain and install a server certificate, and trust the CA's certificate as described in "Obtaining and Installing Server Certificates" on page 113.

Once those tasks are complete, you can then enable the TLSv1/SSL capabilities of the Sun ONE Directory Server software. This process is simple, but involves a few tasks. The first task is to configure the directory server to use TLSv1/SSL.

▼ To Configure the Directory Server to Use TLSv1/SSL

1. **Select the Configuration tab from the Sun ONE Directory Server Console (**FIGURE 3-21**).**

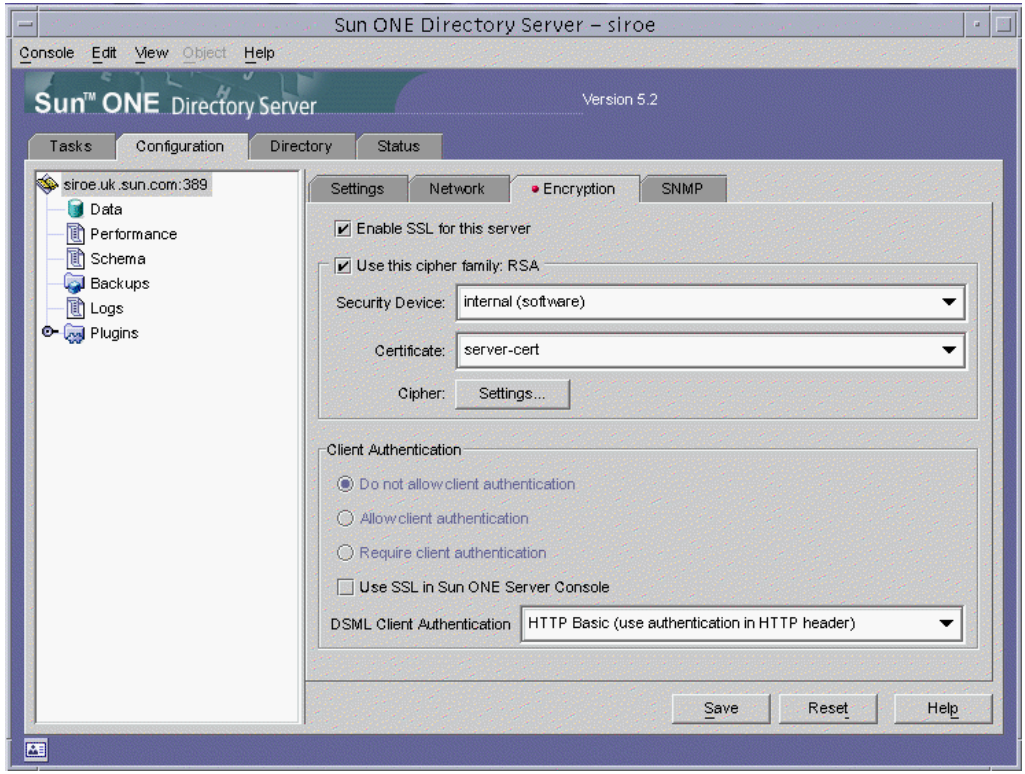

FIGURE 3-21 Encryption Tab in the Configuration Window in the Sun ONE Directory Console

2. **Select the Encryption tab from the right pane, and choose:**
   - Security Device (token)
   - Certificate
3. **Choose Cipher family preferences by clicking the corresponding check box and clicking on the Settings button next to "Cipher:".**

   FIGURE 3-22 is an example of the Cipher Preferences for the Sun ONE Directory Server 5.2 software.

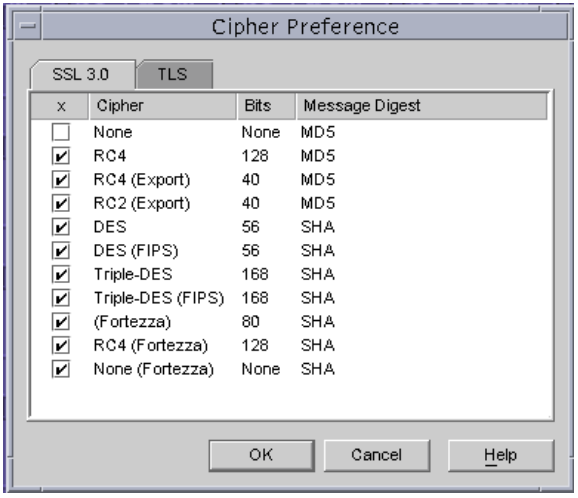

FIGURE 3-22 Cipher Preference Panel

4. **Set the client authentication policy.**
5. **Check Enable SSL for this server.**
6. **Select the server certificate by name.**
7. **(Optional) Choose to connect the Sun ONE Server Console to Directory Server over TLSv1/SSL.**

   It is strongly recommended that you do not select Use SSL in Sun ONE Server Console while initially enabling SSL in the server. If you do this and there is a problem with the way that TLSv1/SSL has been configured, it will not be possible to administer the server through the console without manually editing information under `o=NetscapeRoot`. If you want to administer the directory server through the admin console using TLSv1/SSL, then you should enable that after you have confirmed that TLSv1/SSL has been properly configured.

8. **(Optional) Configure and set the Secure Port for the Sun ONE Directory Server software.**

   a. **Go to the Configuration➤Network Tab as shown in** FIGURE 3-23**.**

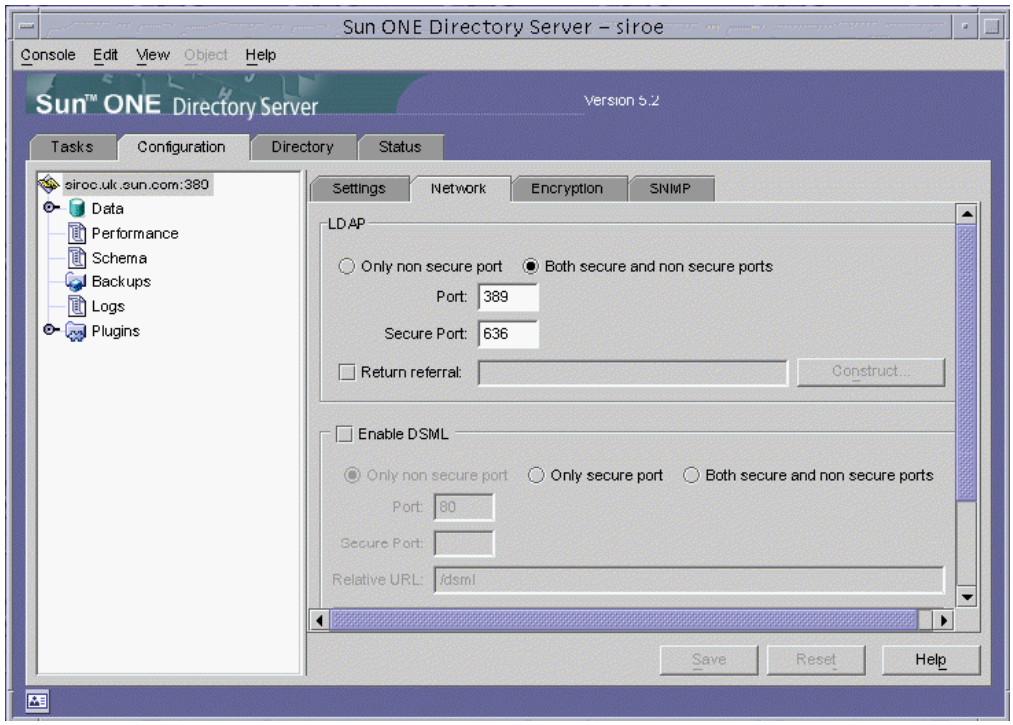

FIGURE 3-23 Network Tab in the Sun ONE Directory Console

b. Specify in the bottom of this pane whether you wish to disable, allow, or require client authentication.

In most cases, the default of Allow client authentication is acceptable.

Note – It is possible to require client authentication for TLSv1/SSL-based connections and still be able to use the admin console, as long as the admin console is not communicating with the directory server over TLSv1/SSL. Therefore, if you want to require client authentication, one possibility is to set `nsslapd-listenhost` to `127.0.0.1` so that it only listens for non-secure connections on the loopback interface, and run the admin console from the directory server machine itself, connecting over LDAP rather than LDAPS.

---

9. Click Save.

After clicking Save, the Console shows a dialog box telling you changes do not take affect until the server is restarted. You must use the command line for this, because the command line provides a means of entering a password.

10. **Stop and restart the directory server.**

Example:

```
# directoryserver_install_dir/slapd-instancename/stop-slapd
# directoryserver_install_dir/slapd-instancename/start-slapd
Enter PIN for Internal (Software) Token:
#
```

Note – If you configure the directory server to verify the client certificate against a certificate stored in the user's entry, then you can use either a binary or an ASCII representation of the certificate to add the data to the user's entry. Usually it's easier to use the ASCII version because it is easier to include in an LDIF file.

Additional Information about TLSv1/SSL in the Sun ONE Directory Server Software

By default, the Sun ONE Directory Server software uses port 636 for encrypted communication because this is the standard LDAPS port. If this port is already in use, or if you would like to use a different port, you can also specify that in the administration server. Under the Configuration tab, click on the Settings tab in the right pane instead of the Encryption tab. The port to use for LDAPS communications is specified in the text field labeled Encrypted port in the Network Settings section. Enter the correct value, and click the Save button to update the configuration.

Alternatively, you can specify the port to use for TLSv1/SSL communication with the secure-port directive (`nsslapd-secureport`) in the `dse.ldif` configuration file.

Although you can use the `nsslapd-listenhost` parameter in the `dse.ldif` configuration file to specify the IP address on which the directory server will listen, this applies only to unencrypted LDAP traffic. If you wish to restrict LDAPS communication to a single IP address, you can do so with the `nsslapd-securelistenhost` directive.

When all of the TLSv1/SSL-related configuration of the Sun ONE Directory Server software is complete, you must restart the server in order for the changes to take effect and cause the directory server to actually listen for LDAPS requests. This introduces a problem, however, because the directory server must be able to access the private key on startup, but this is stored in the certificate database key store, which is password-protected. You will be prompted for this password when the directory server starts, either on the command line (on UNIX systems) or in a dialog box (on Windows NT). On UNIX systems, the fact that the password will be requested from the command line means that you will not be able to use the

administration console to restart the directory server. It also means that the directory server will not automatically start when the machine boots on either UNIX or Windows NT systems. To remedy this, you can create a file that contains the password so that you will not be prompted for the password when the server is restarted. This file should be stored in the `alias` directory under the directory server install root and should be called `slapd-`*`instancename`*`-pin.txt`, where *instancename* is the name of the directory server instance for which you have installed the certificate. You will also see files named `slapd-`*`instancename`*`-cert7.db` and `slapd-`*`instancename`*`-key3.db` in that directory, which are the certificate trust and key stores, respectively. The text that should be placed in your file is `Internal (Software) Token:`*password*, where *password* is the password for the certificate key store (there should not be any spaces on either side of the colon). Permissions on that file should be configured so that they are identical to the permissions of the certificate trust and key store, respectively.

At this point, you can restart the directory server and you should not be asked for a password. Use the command `netstat -an` to look at the network sockets that are in use on the machine. You should be able to see that the directory server is listening on both ports 389 and 636 (or whatever LDAP and LDAPS ports you have chosen). If you do not, then there is likely a problem with the TLSv1/SSL configuration, and you should check the directory server's error log for more information.

Using TLSv1/SSL in the Sun ONE Server Console

The Sun ONE Administration Server Console is actually a web server that uses the HTTP protocol to communicate with the administration console and LDAP to communicate with the configuration directory with the use of CGI. However, it doesn't have to be that way. The administration server can use HTTPS to communicate with the administration console and LDAPS to communicate with the configuration directory. In order to do the former, it needs a server certificate, and the latter requires that the certificate used by the configuration directory is trusted by the administration server. This section describes how to do each of these.

The process of requesting and installing certificates in the administration server is almost exactly the same as requesting and installing certificates in the directory server. The only difference is that you do it in the administration server configuration of the Administration Server console instead of in one of the directory server instances. Upon opening the Sun ONE Server Console for the Administration Server, click on the Configuration tab and then the Encryption tab (FIGURE 3-24). You will see what is essentially the same interface as the Encryption tab in the directory server console.

Note – In the Sun ONE Server Console (5.2) there are now options to either Disable Client Authentication, or Require Client Authentication.

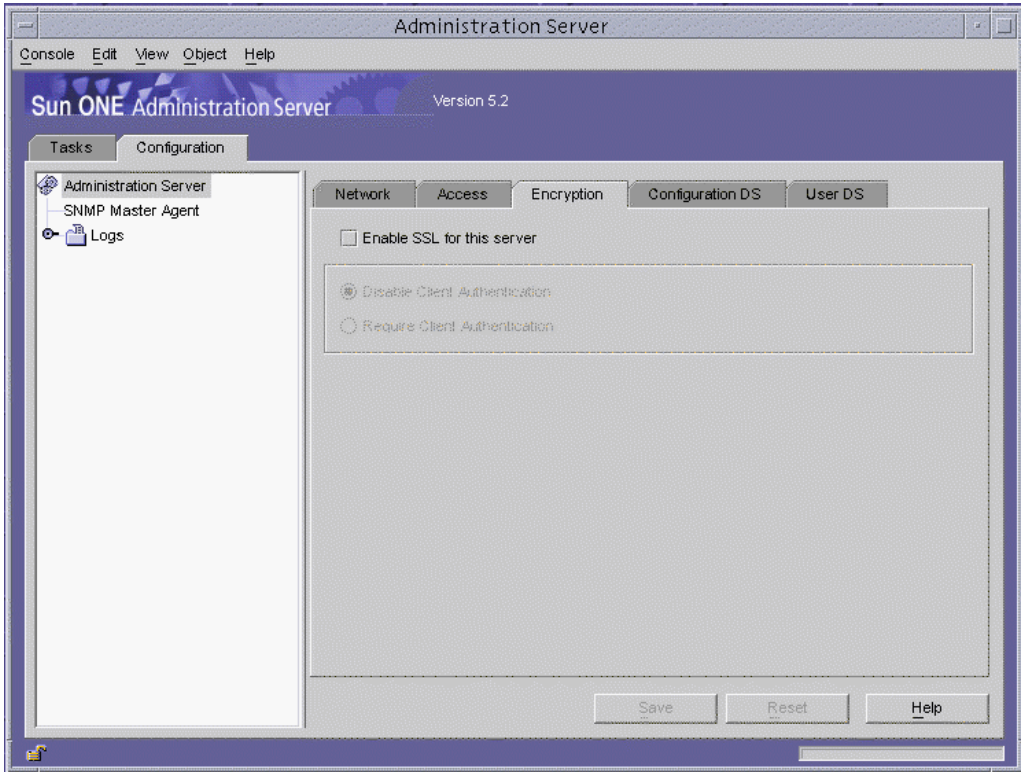

FIGURE 3-24 Sun ONE Server Console

To request and install a server certificate, and to install CA certificates and trusted certificate chains, use the certificate setup wizard exactly as you would use it to perform those functions if the certificates were for the directory server.

Similarly, the process of enabling TLSv1/SSL in the Sun ONE Administration Server Console is virtually identical to enabling TLSv1/SSL in the Sun ONE Directory Server software. Simply check the Enable SSL for this Server checkbox. At that point, it is necessary to restart the administration server. This should be done on the command line in UNIX systems because you will be prompted for the key store password. You cannot use a PIN file to automate this process as you can with the directory server, however a simple workaround exists. Because the `start-admin` program is simply a shell script that sets up the appropriate environment to invoke the `ns-httpd` executable to start the administration server, and because the TLSv1/SSL password is read from STDIN, you can pipe the password to the process when it is started.

Enabling TLSv1/SSL in the Sun ONE Administration Server Console

Unlike the directory server, which has the ability to listen for both LDAP and LDAPS requests at the same time, the administration server can only listen for HTTP or HTTPS traffic, but not both. Therefore, once TLSv1/SSL is enabled in the administration server, the console must use TLSv1/SSL to communicate with it. The change required to do this is simple, but is often overlooked, and might be a source of confusion when you can no longer log in to the administration console once the administration server is using TLSv1/SSL.

The change that needs to be made is in the Administration URL, which is typically `http://servername:port`. Once TLSv1/SSL is enabled for the administration server, the URL must change from `http` to `https`, as shown in FIGURE 3-25:

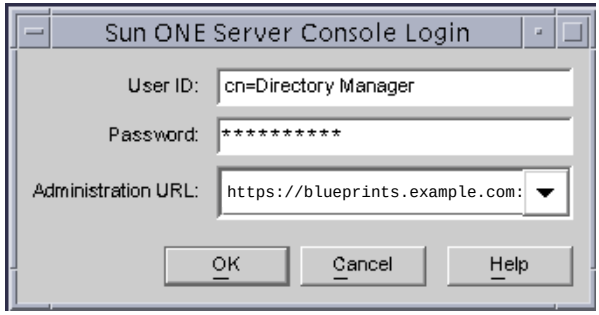

FIGURE 3-25 Using TLSv1/SSL in the Sun ONE Administration Server Console With an `https` URL

Understanding and Using TLSv1/SSL LDAP Client Architecture

This section describes the LDAP operations using both the `ldapsearch` and `ldapmodify` commands over TLSv1/SSL and the Secured LDAP Client implementation in the Solaris OE. Before processing, the following requirements must be meet. The following is only given as an outline with the details discussed later:

- Certificate database:
  - Install the server cert's issuer CA cert – trust for server authentication (trust flag: `C`)
- If cert-based client authentication is required:
  - Key database
  - Request and install user certificate in cert db

- Install user cert's issuer CA cert in server's cert db - trust for client authentication (trust flag: T)

Once a certificate has been installed in the directory server and TLSv1/SSL has been enabled, you should test this functionality to ensure that everything is working properly. The easiest and most convenient way to do this is by using the `ldapsearch` command-line utility. This utility is located in the `shared/bin` directory under the directory server root and has the ability to communicate with the directory server using TLSv1/SSL using either server or client authentication.

Note – This is not the Solaris 9 OE integrated version of the `ldapsearch` command-line tool.

First, it is necessary to understand the syntax for the `ldapsearch` utility without TLSv1/SSL enabled. The basic syntax for the `ldapsearch` command is as follows:

```
$./ldapsearch options query attributes-to-return
```

The options described in TABLE 3-7 are the most commonly used:.

TABLE 3-7 Common `ldapsearch` Options

Command-line Options	Comment
`-h`	The DNS host name or IP address of the directory server. If this parameter is not specified, the default localhost (`127.0.0.1`) is used.
`-p`	The port on which the directory server is listening. If this parameter is not specified, the default `389` is used.
`-D`	The DN to use to bind to the directory server. If this parameter is not specified, the search will be performed anonymously.
`-w`	The password to use for the bind DN. If a bind DN is specified, the password is required.
`-b`	The DN of the entry to use as the search base. This parameter is required.
`-s`	The search scope. It must be one of the following: • `base` • `one` • `onelevel` • `sub` • `subtree` If this parameter is not specified, then a default of `subtree` is used.

The easiest way to test connectivity to the directory server using `ldapsearch` is to retrieve the root DSE. This entry is available anonymously (so no bind credentials are required), and it is known to exist in the directory server no matter how the administrator has configured the server. The root DSE has a null DN, which you can specify in `ldapsearch` as "", and you must perform a base-level search to get any results. Assuming that the directory server is listening on `127.0.0.1:389`, the correct syntax for the search is:

```
$./ldapsearch -h 127.0.0.1 -p 389 -b "" -s base "(objectclass=*)"
```

You will see the root DSE in LDIF format returned. This is returned using the unencrypted LDAP protocol, but confirms that the directory server is up and responding properly.

Look at the `ldapsearch` operational arguments for the `ldapsearch` command over TLSv1/SSL as shown in TABLE 3-8.

TABLE 3-8 Common Command-line Options to `ldapsearch` with TLSv1/SSL

Command-line Options	Comment
Simple Authentication:	
`-p`	Secure port
`-Z`	TLSv1/SSL encrypted connection
`-P`	cert-db-path
Certificate-based client authentication:	
`-K`	key-db-path
`-W`	key-db-pwd
`-N`	certificate-name
`-3`	cn check in server authentication

Note – Earlier versions of the directory server did not require the `-K` argument to `ldapsearch` if you were only using TLSv1/SSL server authentication. However, starting with version 5.1, and still applicable to version 5.2, it is necessary to provide the path to the private-key store even if client authentication is not going to be used. This is because of a change in the underlying LDAP SDK for C used to build tools like `ldapsearch` and `ldapmodify`.

To test the TLSv1/SSL capabilities of the Sun ONE Directory Server software, you must add a couple more parameters to the search request. The `-Z` parameter, as mentioned above, indicates that you are using TLSv1/SSL to make an LDAPS connection, and the `-P` *path-to-trust-db* parameter specifies the location of the certificate trust database. The easiest certificate trust database to use is the directory server's own certificate database, which is the `slapd-`*instancename*`-cert7.db` file in the *server-root*`/alias` directory, because we know that the directory server's own certificate is included in that trust database and is trusted by default.

Additionally, you must change the port from the *insecure* LDAP port to the TLSv1/SSL-enabled LDAPS port of the directory server. With the above changes, the following command should be able to retrieve the root DSE of the directory server using LDAPS, (assuming that the directory is listening for LDAPS requests on `127.0.0.1:636` and the instance name of the server is `example`).

```
$./ldapsearch -h 127.0.0.1 -p 636 -b "" -s base -Z
-P ../../alias/slapd-example-cert7.db "(objectclass=*)"
```

Note – The above command should all be on one line.

In the above example, you can see exactly the same results as in the previous search, but the search is done using LDAPS instead of LDAP. If so, then the directory server is responding properly to TLSv1/SSL requests using server authentication.

The `ldapsearch` utility can also be configured to make LDAPS requests using client authentication. This process is significantly more complex than server authentication and carries a few additional requirements. Those requirements include:

- The public and private keys of the client certificate must exist in the certificate database that is used.
- The client certificate must exist in the directory server in the `usercertificate;binary` attribute of the entry with which you are binding. This is only true if the `certmap.conf` file has been configured to verify the certificate (`verifycert` set to `on`).
- The certificate mapping mechanism must be used to uniquely map the certificate subject DN to an entry in the database. There are a number of ways of establishing this mapping if the DN of the user does not match the subject DN of the certificate (and it is not very common for a user certificate's subject DN to match the DN of the user's entry in the directory). But unless the subject of the client certificate exactly matches the DN of the directory entry containing the certificate, you must provide the `-D` parameter that contains the DN of the user entry as whom you wish to bind. The `certmap.conf` allows some flexibility here.

The mechanism for fulfilling these requirements is discussed in the next section of configuring an LDAP Client to use TLSv1/SSL. It must be noted, however, that if you want to use client authentication, it is recommended that you use a Netscape Communicator (any browser can be used that supports certificate and key databases) certificate database (this is shown in our example) rather than the directory server's certificate database. Once you have met those requirements, then you must specify the additional parameters, as shown in TABLE 3-8, for the `ldapsearch` command.

All of the examples in this section used the `ldapsearch` command-line utility to interact with the Sun ONE Directory Server software using LDAPS. The same functionality exists in the `ldapmodify` utility, which can be used to add, delete, or modify entries in the directory server. The use of the `ldapmodify` command is not discussed here, but all of theTLSv1/SSL related options are exactly the same for `ldapmodify` as for `ldapsearch`. Therefore, if you can use `ldapsearch` to search the directory using an TLSv1/SSL-encrypted connection, then you can use those same options with `ldapmodify` to modify the directory server data.

▼ To Generate a TLSv1/SSL Client Certificate

In this procedure showing an example of using client authentication, all of the users on a client that wish to use TLSv1/SSL when connecting to a Sun ONE directory server using LDAP client applications, must generate a TLSv1/SSL client certificate. To create a certificate we need to follow these steps listed below:

1. **Request a signed certificate from the CA using the Netscape browser.**

   This example in FIGURE 3-26 uses the Sun ONE Certificate Server URL to request a client certificate (`https://blueprints.example.com:443`):

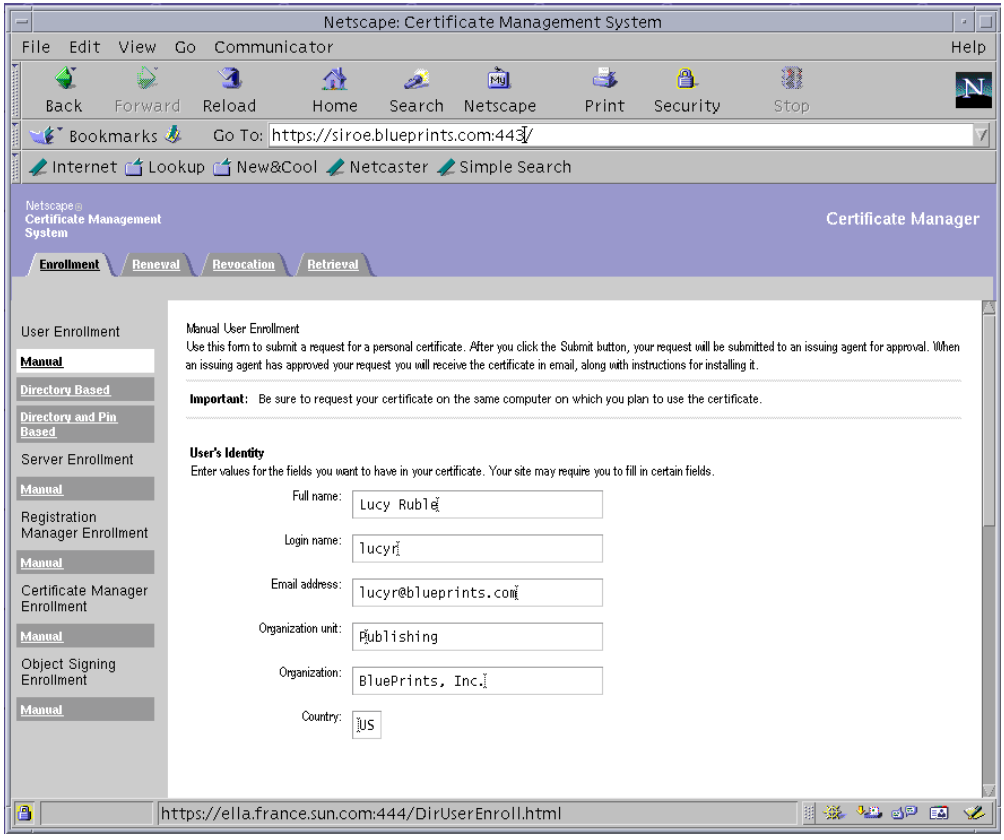

FIGURE 3-26 Sun ONE Certificate Manager

2. **Fill in all the necessary information.**

3. **Select the Submit button at the bottom of the request page.**

   The browser generates a keypair and sends the public part of this keypair to the Sun ONE Certificate Server in our example. The Sun ONE Certificate Server software then signs the key together with the additional information that you previously provided.

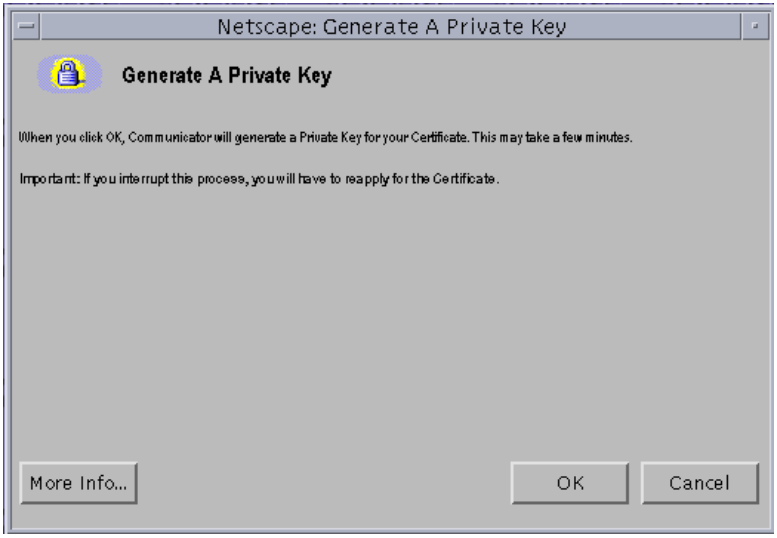

FIGURE 3-27 Key Generation Dialog Box

4. **Click OK (**FIGURE 3-27**).**
5. **Enter a password to protect your private key (**FIGURE 3-28**).**

FIGURE 3-28 Password Request Dialog Box

After you enter your password, the private key is generated. If you are using Netscape Communicator or the Mozilla browser, you should see `cert7.db`, `key3.db` and `secmod.db` under your browser's directory (for example, the `.netscape` or `.mozilla` directories). The Netscape browser uses the security database files in the `~/.netscape` directory. Therefore, it is already set up to query the Sun ONE directory server using TLSv1/SSL. The `secmod.db` is the file where Network Security Services (NSS) lists the different PCKS#11 modules available to you. Each module can have several slots, each slot being usually based on a token or security device.

For instance, the NSS library delivers an internal module by default, which consists of a couple of slots (you can see this by using `modutil` on the `secmod.db/secmodule.db`):

- Slot NSS Internal Cryptographic Services -> Token "NSS Generic Crypto Services"
- Slot NSS User Private Key and Certificate Services -> Token "NSS Certificate DB"

This file is also where you plug in your external modules, such as accelerator cards.

If you experience a message like the following:

```
Please enter the password of pin for the Communicator
Certificate DB
```

Then it assumes you have already provided a password the first time you accessed your browser's certificate/key databases. If you don't remember the password, there is nothing you can do, and you will have to generate a new cert/key database pair.

The Sun ONE Certificate software presents a request ID to you. Before you can import the certificate, the Certificate Server Administrator must approve your request. You can receive the request notification by email or verbally.

Example of a Certificate Request being sent by email:

```
<html>
<body>
<h2>An automatically generated notification from <i>ca</i></h2>
Your certificate request has been processed successfully.
<p>
SubjectDN= <b>E=lucy.ruble@example.com,CN=Lucy Ruble,UID=
lucyr,OU=Publishing,O=Example,C=US</b><br>
IssuerDN= <b>CN=Certificate Manager,O=Example,Inc,C=us</b><br>
notAfter= <b>09-Jan-04 9:28:59 PM</b><br>
notBefore= <b>10-Jan-03 9:28:59 PM</b><br>
Serial Number= <b>153</b><p>
<p>
To get your certificate, please follow this
<A HREF="https://cms.blueprints.com:444/displayBySerial?op=
displayBySerial&serialNumber=153">URL</A>

Please contact your admin if there is any problem.
</body>
</html>
```

Example of a Request that has been successfully submitted is shown in FIGURE 3-29.

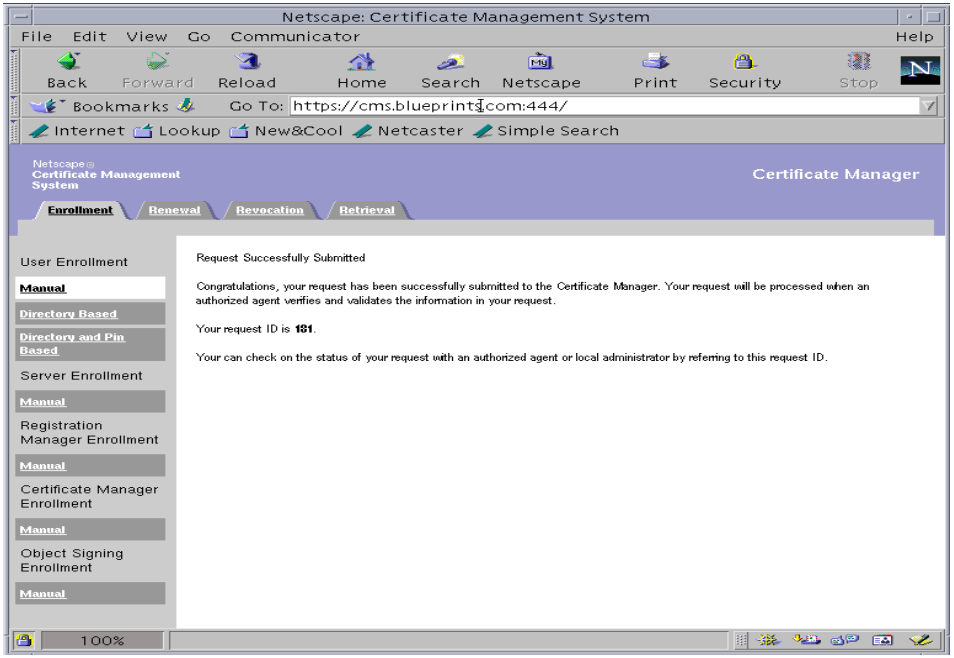

FIGURE 3-29 Example of a Successfully Submitted Request

The following examples show actual signed certificate components.

This is the certificate section:

```
# Example certificate
Certificate  0x099


 Certificate contents


    Certificate:
        Data:
            Version:  v3
            Serial Number: 0x99
          Signature Algorithm: MD5withRSA - 1.2.840.113549.1.1.4
            Issuer: CN=Certificate Manager,O=blueprints, C=us
            Validity:
                Not Before: Thursday, April 10, 2003 9:28:59 PM
                Not  After: Friday, April 9, 2004 9:28:59 PM
```

```
            Subject: E=lucy.ruble@sun.com,CN=Lucy Ruble,UID=
lucyr,OU=Publishing,O=Example, Inc,C=US
            Subject Public Key Info:
                Algorithm: RSA - 1.2.840.113549.1.1.1
                Public Key:
                30:81:89:02:81:81:00:AE:1D:D5:23:20:AC:F7:BD:B8:
                44:42:77:BE:23:AA:FD:32:46:41:CA:D1:F0:F2:24:94:
                43:71:ED:63:22:84:DB:EC:68:2B:FF:32:D1:FC:F6:B4:
                98:39:7C:B4:ED:B7:A7:12:89:EE:C2:DF:8D:71:D3:35:
                07:56:0E:33:F0:F5:A6:EE:6B:DD:43:92:FD:90:31:8B:
                0B:B9:DD:5A:8E:05:79:15:F4:21:87:FC:DC:81:73:49:
                03:32:78:D2:AA:13:0F:32:D5:E4:C1:88:92:B7:B3:B5:
                B6:CF:2B:AF:68:C8:A4:8C:D6:1B:02:74:81:45:93:D1:
                    8F:E8:A5:C9:59:ED:85:02:03:01:00:01
            Extensions:
             Identifier: Certificate Type - 2.16.840.1.113730.1.1
                    Critical: no
                    Certificate Usage:
                        SSL Client
                        Secure Email
                Identifier: Key Type - 2.5.29.15
                    Critical: yes
                    Key Usage:
                        Digital Signature
                        Non Repudiation
                        Key Encipherment
                Identifier: Authority Key Identifier - 2.5.29.35
                    Critical: no
                    Key Identifier:

DB:6B:27:D7:93:90:3B:68:BB:41:10:12:AB:36:D8:95:
                        02:60:F0:6C
        Signature:
            Algorithm: MD5withRSA - 1.2.840.113549.1.1.4
            Signature:
                94:5B:04:2D:0B:82:A6:FD:C9:C0:49:95:B1:C1:8D:09:
                67:7C:AA:E0:A1:ED:4D:CF:4A:2F:FF:66:87:B1:88:D0:
                FA:B0:AA:EB:68:15:7F:92:87:52:FD:7E:A1:2B:0C:AA:
                D6:FE:BE:05:B4:09:97:E9:6D:CC:27:7A:88:4D:87:09
```

The following is the certificate fingerprint section:

```
# Example certificate fingerprint

Certificate fingerprints


    MD2:   4F:22:38:50:E2:C4:A4:09:95:06:E0:E4:A0:1F:9B:3F
    MD5:   9E:8F:5F:ED:9F:FB:D2:14:2D:AF:74:E0:62:90:60:CE
    SHA1:
2D:8E:33:90:19:CE:18:7E:B3:9B:5C:4D:DC:AE:B5:05:3A:FD:F8:87
```

These are the details on how to install the certificate:

```
# Example of how to install the certificate

The following format can be used to install this certificate into
a Netscape server.

-----BEGIN CERTIFICATE-----
MIICVjCCAgCgAwIBAgICAJkwDQYJKoZIhvcNAQEEBQAwOTELMAkGA1UEBhMCdXMx
DDAKBgNVBAoTA3N1bjEcMBoGA1UEAxMTQ2VydGlmaWNhdGUgTWFuYWdlcjAeFw0w
MzA0MTAxNjU4NTlaFw0wNDA0MDkxNjU4NTlaMIGgMQswCQYDVQQGEwJVSzEeMBwG
A1UEChMVU3VuIE1pY3Jvc3lzdGVtcywgSW5jMRgwFgYDVQQLEw9TdW4gRW5naW5l
ZXJpbmcxFzAVBgoJkiaJk/IsZAEBEwdtaDEzNzQ5MRcwFQYDVQQDEw5NaWNoYWVs
IEhhaW5lczElMCMGCSqGSIb3DQEJARYWbWljaGFlbC5oYWluZXNAc3VuLmNvbTCB
nzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEArh3VIyCs9724REJ3viOq/TJGQcrR
8PIklENx7WMihNvsaCv/MtH89rSYOXy07benEonuwt+NcdM1B1YOM/D1pu5r3UOS
/ZAxiwu53VqOBXkV9CGH/NyBc0kDMnjSqhMPMtXkwYiSt7O1ts8rr2jIpIzWGwJ0
gUWT0Y/opclZ7YUCAwEAAaNGMEQwEQYJYIZIAYb4QgEBBAQDAgWgMA4GA1UdDwEB
/wQEAwIF4DAfBgNVHSMEGDAWgBTbayfXk5A7aLtBEBKrNtiVAmDwbDANBgkqhkiG
9w0BAQQFAANBAJRbBC0Lgqb9ycBJlbHBjQlnfKrgoe1Nz0ov/2aHsYjQ+rCq62gV
f5KHUv1+oSsMqtb+vgW0CZfpbcwneohNhwk=
-----END CERTIFICATE-----
```

6. **Once you receive a signed certificate, import the signed certificate.**

   With the signed certificate, it is now possible to import the signed client certificate. To perform this, scroll down to the end of the page, and look for:

   ```
   Importing this certificate
   ```

   To import the certificate into your client, select the Import Your Certificate button.

7. **Enter the password for the Communicator Certificate Database (**FIGURE 3-30**).**

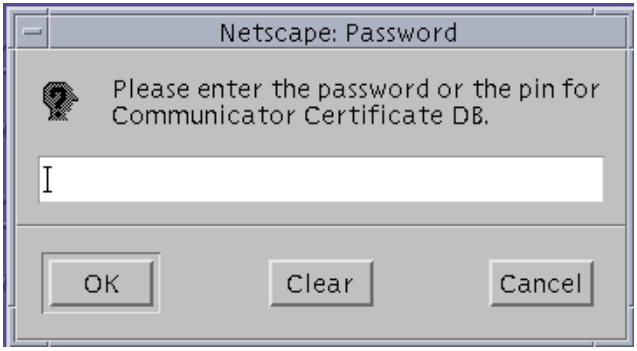

FIGURE 3-30 Password Dialog Box for the Communicator Certificate Database

Once you have successfully typed in the correct password, you will have a new private key stored in `~/.netscape/key3.db`, and a new certificate stored in `~/.netscape/cert7.db`.

You must make sure that the certificate of the CA that signed our TLSv1/SSL-client certificate is trusted.

8. **Verify and set up the appropriate trust relations.**

   a. **From the Netscape browsers menu select Communicator ➤ Tools ➤ Security Info.**

   b. **Under the Certificates, select Your certificate.**

      You should see the certificate you just imported from the Certificate Server.

   c. **Select the certificate and click the Verify button.**

      The browser shows you a dialog box (FIGURE 3-31) showing that the certificate is not trusted.

FIGURE 3-31 Netscape Verify a Certificate Dialog Box Showing Failed Verification

d. Go to the Security Info window, select Signers in the navigation frame and select the certificate signer from the list in the right frame.

This changes the certificate so that it is trusted.

The CA that signed the certificate must have the appropriate trust relations. To accomplish this task from the Netscape Browser.

e. Click the Edit button.

The Edit Certification Authority dialog box appears (FIGURE 3-32).

f. Modify the trust relation by checking each Accept option (FIGURE 3-32**).**

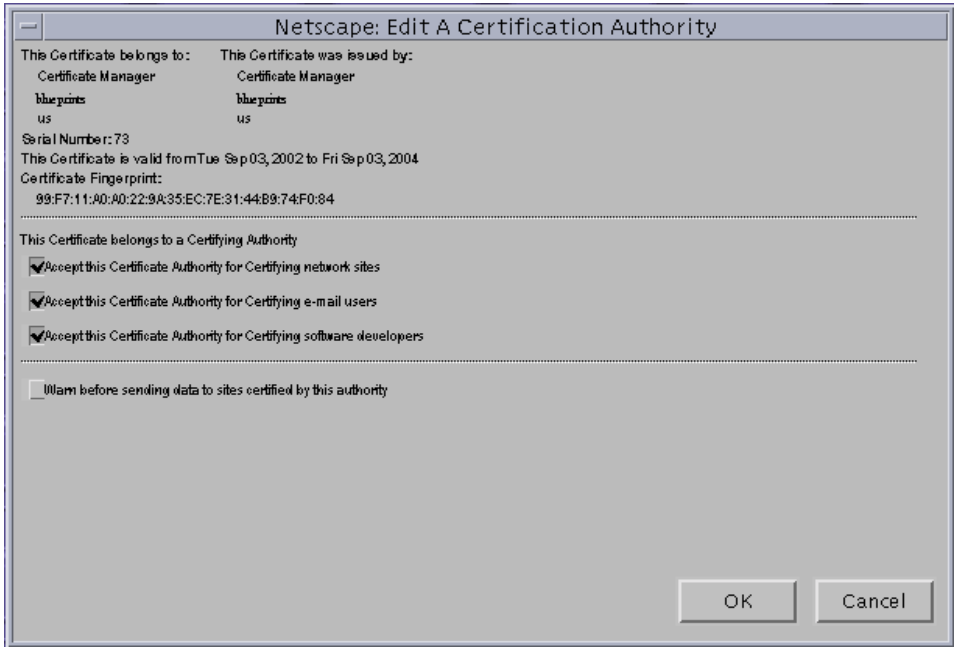

FIGURE 3-32 Edit a Certification Authority Dialog Box

g. Verify the client certificate again, as you did previously.

The browser shows that the certificate is now trusted (FIGURE 3-33).

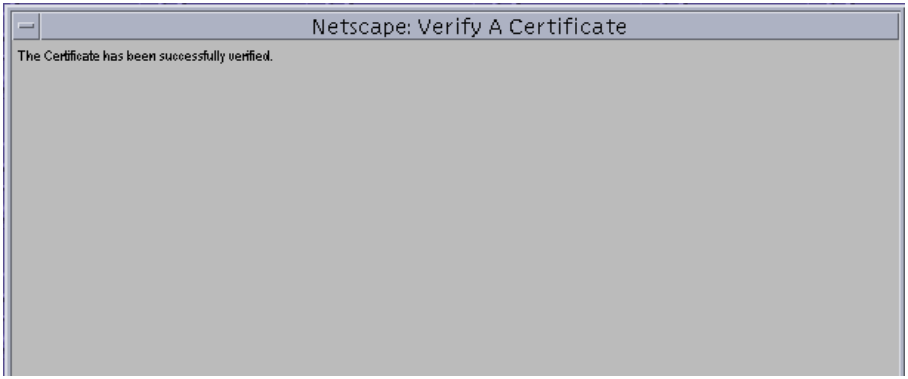

FIGURE 3-33 Verify a Certificate Dialog Box

Initializing the Secured LDAP Client

When you initially created your client certificate, you chose to use the manual option from the certificate management system (CMS). Remember, however, that by selecting this option, you end up having your client certificate in your browser's certificate/key database pair, so these are the databases that you have to use with the Secured LDAP client application.

To use the TLSv1/SSL security databases (`cert7.db` and `key3.db`), these databases must be placed in the directory defined by the clients `certificatePath` attribute. This is required because the Secured LDAP Client library (`libsldap.so.1`) uses the `ldapssl_client_init` API from `libldap` to initialize itself to connect to a secure LDAP server over TLsv1/SSL. This call requires the path to the database containing certificates and the database must be a `cert7.db` certificate database.

You can use the `NS_LDAP_HOST_CERTPATH` parameter in the Secured LDAP Client profile to specify the path. If you don't, the path by default is the `/var/ldap` directory. So copy the database files and give read access as shown in the following example:

```
# /usr/bin/cp /.netscape/cert7.db /var/ldap
# /usr/bin/cp /.netscape/key3.db /var/ldap
# /usr/bin/chmod 400 /var/ldap/cert7.db /var/ldap/key3.db
```

Note – The Netscape browser uses the security database files in the `~/.netscape` directory. Therefore, it is already set up to query the Sun ONE Directory Server software using TLSv1/SSL.

The Solaris OE version of the `libldap` library and TLSv1/SSL require mutual authentication. Therefore, the servers IP address that the client uses must resolve to the same name that is contained in the servers certificate. Because the servers certificate uses its fully qualified domain name, `example.com` for example, if you are using the Secured LDAP client, the address must resolve to the name in the certificate.

Note – The LDAP name service cannot be used to resolve the address to the LDAP server. DNS can be used for host resolution. If you add the host to the `/etc/hosts` file, be sure to add it so that the host name resolves to the same name that is in the certificate.

First add the full host name and address of the server to the `/etc/hosts` file. Edit the `/etc/nsswitch.ldap` file to use `files` and then `LDAP` for hosts resolution. The modified `nsswitch.ldap` file should have an entry as follows:

```
hosts: files ldap
```

You must modify the `/etc/nsswitch.ldap` file because when you run the `ldapclient init` command, it is copied to the `/etc/nsswitch.conf` file.

The TLSv1/SSL support in the Secured LDAP Client is implemented as a library, and it is `libldap.so.5` that actually implements the client side of it. This works in the following way.

In the Secured LDAP client profile, the `authenticationMethods` that you can specify are:

- `NONE`
- `SIMPLE`
- `SASL/DIGEST-MD5`
- `SASL/CRAM-MD5`
- `TLS:NONE`
- `TLS:SIMPLE`
- `TLS:SASL/CRAM-MD5`
- `TLS:SASL/DIGEST-MD5`

Those that start with `TLS:` indicate that a TLSv1/SSL session is required. When the `libsldap` library sets up the connection to the Sun ONE Directory (LDAP) Server (or any directory server for that matter), it first calls a private interface in `libldap.so.5` to initialize the client application for TLSv1/SSL (open the certificate database), then calls the private interface again to initialize an LDAP session with the secure directory server. After this, everything is performed in the same way as a non-TLS session.

In the Secured LDAP Client, there is a list of encryption types with encryption strengths of TLSv1/SSl certificates that can be used by the Solaris OE Secured LDAP Client.

The Secured LDAP client informs the directory server which cipher suites it supports (in preferential order – see below). The directory server replies with the subset of mechanisms it supports (in preferential order). The policy is to allow all of these cipher suites, except those that are not enabled. The following cipher suites are present by default:

- SSL 3.0:
- SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5
- SSL_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_RSA_EXPORT1024_WITH_RC4_56_SHA
- TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_FIPS_WITH_DES_CBC_SHA
- SSL_RSA_WITH_DES_CBC_SHA

These may work (see Note):

- SSL_FORTEZZA_DMS_WITH_NULL_SHA
- SSL_FORTEZZA_DMS_WITH_FORTEZZA_CBC_SHA
- SSL_FORTEZZA_DMS_WITH_RC4_128_SHA

Note – The three Fortezza cases listed above require other things installed, such as cards, boards, and tokens. Sun does not currently support them at this time.

For SSL2:

- SSL_CK_RC4_128_WITH_MD5
- SSL_CK_RC4_128_EXPORT40_WITH_MD5
- SSL_CK_RC2_128_CBC_WITH_MD5
- SSL_CK_RC2_128_CBC_EXPORT40_WITH_MD5
- SSL_CK_IDEA_128_CBC_WITH_MD5
- SSL_CK_DES_64_CBC_WITH_MD5
- SSL_CK_DES_192_EDE3_CBC_WITH_MD5

▼ To Verify That TLSv1/SSL Is Working

This procedure verifies that TLSv1/SSL is working on the directory server and that the client profile is set up.

1. **Use the** `ldapsearch` **command as shown in the following examples.**

```
$ ./ldapsearch -h directoryserver_hostname -p ldap_port
-D “cn=Directory Manager” -w password
-b “cn=encryption,cn=config” cn=*
objectClass=top
objectClass=nsEncryptionConfig
cn=encryption
nsSSLSessionTimeout=0
nsSSLClientAuth=allowed
nsSSLServerAuth=cert
nsSSL2=off
nsSSL3=on
```

```
$ ./ldapsearch -h directoryserver_hostname -p ldap_port
-b "ou=profile,dc=example,dc=com" "cn=default"
cn=default,ou=profile,dc=example,dc=com
objectClass=top
objectClass=DUAConfigProfile
defaultServerList=blueprints.example.com
defaultSearchBase=dc=example,dc=com
authenticationMethod=tls:simple
followReferrals=FALSE
defaultSearchScope=one
searchTimeLimit=30
profileTTL=43200
cn=default
credentialLevel=proxy
bindTimeLimit=10
```

2. **Initialize the Secured LDAP client with the** `ldapclient init` **command using the name of the server instead of the address.**

   Example:

```
$ ./ldapclient init -a proxydn=cn=proxyagent, ou=profile,dc=
example,dc=com -a proxypassword=proxy -a domainname=example.com
example.com
#
$ ldapclient list
NS_LDAP_FILE_VERSION= 2.0
NS_LDAP_BINDDN= cn=proxyagent,ou=profile,dc=example,dc=com
NS_LDAP_BINDPASSWD= {NS1}ecc423aad0
NS_LDAP_SERVERS= blueprints.example.com
NS_LDAP_SEARCH_BASEDN= dc=example,dc=com
NS_LDAP_AUTH= tls:simple
NS_LDAP_SEARCH_REF= FALSE
NS_LDAP_SEARCH_SCOPE= one
NS_LDAP_SEARCH_TIME= 30
NS_LDAP_PROFILE= default
NS_LDAP_CREDENTIAL_LEVEL= proxy
NS_LDAP_BIND_TIME= 10
```

3. **(Optional) Verify the encrypted traffic.**

   It is good practice to verify the encrypted traffic. Out of the standard Solaris OE command-line tools, only the `ldaplist` and `ldapaddent` commands access the Sun ONE Directory (LDAP) server using TLSv1/SSL. The command-line tools (`ldapsearch`, `ldapmodify` and `ldapdelete`) that are provided with the Sun ONE Directory software distribution are enhanced to use TLSv1/SSL. These are located in the *path/directory-server-instance*`/shared/bin` directory.

   Here is an example of verifying the encrypted traffic with TLSv1/ SSL using the `ldapsearch` command:

```
$ ./ldapsearch -h directoryserver_hostname -p ldap_port 636 -Z
-P /var/ldap/cert7.db -b "dc=example,dc=com" "cn=*"
```

   To actually verify that the traffic is encrypted, use something like the Solaris 9 OE `/usr/sbin/snoop` command (see Appendix C, “Using `snoop` with LDAP). The more advanced protocol analyzers like Ethereal (available for free for a number of platforms, and as source code from `http://www.ethereal.com`) can even interpret the information that is captured so that it can be more easily understood. This is helpful with text-based protocols like HTTP because it provides formatting

for the request. It is invaluable for binary protocols like LDAP because the task of decoding the information and figuring out exactly what is going on between the client and the server is much more difficult.

Note – Ethereal is available on the Solaris 8 and 9 OE Companion Software CDs. However, support for this utility is not provided by Sun.

Start TLS Overview

The TLS Protocol Version 1.0 is defined in RFC 2246. Before deciding to use the Start TLS functionality, it is worth taking the time to understand what TLS actually offers. The primary use of the TLS protocol with LDAP is to ensure connection confidentiality and integrity, and to optionally provide for authentication. Be aware that using the Start TLS operation on its own does not provide any additional security because the security element is accomplished through the use of TLS itself.

The level of security provided though the use of TLS is dependent directly on both the quality of the TLS implementation used and the style of usage of that implementation.

Note – The Start TLS extended operation in the Sun ONE Directory Server 5.2 software is available on all platforms. This was not the case with version 5.1, where NT is not supported.

The Start TLS operation is an extended operation defined by the LDAPv3 protocol that is initiated by a client starting the TLS protocol over an already established LDAP connection. What actually happens is that the client transmits an LDAP PDU (protocol data unit) containing the LDAPv3 `ExtendedRequest`, and specifying the OID for the Start TLS operation. The OID is:

```
1.3.6.1.4.1.1466.20037
```

This extended operation enables the ability of securing a connection that was not secure, based on a client's demand.

This extended operation is forwarded to the directory server in terms of an LDAP extended request that contains a specific OID as referenced above, identifying the Start TLS operation. It is up to the directory server to decide whether or not the request should be accepted or rejected. The server sends an extended LDAP PDU containing a Start TLS extended response back to the client with either a successful or a non-successful answer as to whether the directory server is willing and able to negotiate TLS. If the `ExtendedResponse` contains a result code indicating success, then the directory server is willing and able to negotiate TLS. If, on the other hand,

the `ExtendedResponse` contains a result code other than success, this indicates that the directory server is unwilling or unable to negotiate TLS. If the Start TLS extended request is not successful, the result code will be one of:

- `operationsError` – TLS already established.
- `protocolError` – TLS is not supported or incorrect PDU structure.
- `referral` – The directory server does not support TLS.
- `unavailable` – There is a serious problem with TLS, or the directory server is down.

In the case that a successful response is returned, the directory server initializes TLSv1/SSL (initializes both the certificate and key databases, and sets the cipher policy), imports the current socket into a TLSv1/SSL-socket, and configures it in order to behave as a TLSv1/SSL server.

In the Sun ONE Directory Server 5.2 software, the Start TLS extended operation is implemented as an internal extended operation plug-in. The implementation itself is based on the IETF RFC 2830 “Lightweight Directory Access Protocol (v3): Extension for Transport Layer Security” (`ftp://ftp.rfc-editor.org/in-notes/rfc2830.txt`).

When the Sun ONE directory server receives a Start TLS extended operation request, it performs a series of checks as specified in the above document (such as checking whether there are other operations still pending on the connection, whether security is enabled on the directory server, and so on). If successful, it performs the TLSv1/SSL handshake and uses a secure connection.

As to the configuration of the Sun ONE directory server, no specific configuration has to be taken into account except for the Windows platforms, where you must add the `ds-start-tls-enabled: on` attribute to the `cn=config` entry. This is necessary because on Windows, the server handles connections differently depending on whether they are secured or non-secured; secured connections need to be handled using NSPR (as NSS is built upon NSPR), whereas non-secured ones benefit from the Windows I/O completion ports architecture, which turns out to be more efficient. So, if you want to convert a non-secured connection into a secured one (perform Start TLS operation), you must know beforehand so that the server can also handle non-secured connections using NSPR right from the start-up.

For Start TLS to work, the security must be enabled in the server, with all the necessary configuration (certificate and key databases, server certificate available, cipher preferences set, client authentication policy specified, and so forth).

Note – You don’t need to have a dedicated secure port open. That is, in fact, one of the strong points of Start TLS— it allows you to have secure connections on the non-secure LDAP port.

In summary, the complexity of setting up TLSv1/ SSL is mainly on the Sun ONE directory server-side, and is categorized in the following main points:

1. Find a CA. You can use an existing one, or you will have to set up something like the Sun ONE/iPlanet™ Certificate Server.
2. The Sun ONE Directory (LDAP) server has to get a server certificate from the CA, and has to activate TLSv1/SSL afterwards.
3. The Secured LDAP Client has to get a CA certificate from the same CA.
4. You must copy `$HOME/cert7.db` and `$HOME/key3.db` to `/var/ldap`. The path `/var/ldap` can be overwritten by setting up `certificatepath` with `ldapclient -a certificatepath` *path*

Note – Currently the Secured LDAP does not use Start TLS.

---

5. If you want to test an LDAP client authentication, the server has to get a CA certificate and the client has to get a client certificate from the same CA. But it's not required by the Secured LDAP Client.

Enhanced Solaris OE PAM Features

The Pluggable Authentication Module (PAM) feature is an integral part of the authentication mechanism for the Solaris 9 OE. PAM provides you with the flexibility to choose any authentication service available on a system to perform end-user authentication. Other PAM implementations are Linux-PAM and OpenPAM.

By using PAM, applications can perform authentication regardless of what authentication method is defined for a given client.

PAM enables you to deploy the appropriate authentication mechanism for each service throughout the network. You can also select one or multiple authentication technologies without modifying applications or utilities. PAM insulates application developers from evolutionary improvements to authentication technologies, while at the same time, allows deployed applications to use those improvements.

PAM employs run-time pluggable modules to provide authentication for system entry services. PAM offers a number of benefits, including:

- Offers flexible configuration policy by enabling each application or service to use its own authentication policy. PAM provides the ability for you to choose a default authentication mechanism. By using the PAM mechanism to require

multiple passwords, protection is enhanced on high-security systems. For example, you might want users to be authenticated by Kerberos, and to bind to a directory server using SASL/DIGEST-MD5.

- Provides ease-of-use for end users. Password usage is easier using PAM. If users have the same passwords for different mechanisms, they do not need to retype the password. Configured and implemented properly, PAM offers a way to prompt the user for passwords for multiple authentication methods without having the user enter multiple commands. For example, a site may require certificate-based password authentication for `telnet` access, while allowing console login sessions with just a UNIX password.
- Enhances security and provides ease-of-use of the Solaris 9 OE in an extensible way. The security mechanisms accessible through PAM are implemented as dynamically loadable, shared software modules that are installed by system administrators in a manner that is transparent to applications. By increasing overall security, users enjoy greater service levels and lower cost of ownership.

Traditional Solaris OE Authentication and PAM

Traditional Solaris OE authentication is based on the method developed for early UNIX implementations. This method employs a one-way encryption hashing algorithm called `crypt(3c)`. The encrypted password is stored either in a file or in a Solaris OE naming service, from which it is retrieved during the user login process. The traditional UNIX method of the Solaris OE authentication, using `crypt`, is very popular and has been enhanced to use an LDAP directory as its data store.

Before proceeding with the details on authentication, you must have a good understanding of what `crypt` is. There is some confusion because of a naming conflict with an *application* named *crypt*. This is a standard tool that ships with the Solaris OE, and is a program for encrypting and decrypting the contents of a file. (This program is located in `/usr/bin/crypt`.)

However, when the term *crypt* is referred to in authentication, it is normally cited as `crypt(3c)` and refers to the standard UNIX password hashing algorithm (`crypt(3c)`), available to C programmers in the `libc.so` library.

A more sophisticated authentication method based on public key technology was introduced with the Network Information System (NIS+) naming service (now rebranded as the Sun OS™ 5.0 Network Information Service). The NIS+ naming service method does not replace `crypt(3c)`, but rather provides an additional security layer by introducing the concept of a network password. When users access network services through the secure remote procedure call (RPC) mechanism, the network password is required.

Originally developed by Sun Microsystems and adopted by the Open Software Foundation (OSF) for inclusion in Common Desktop Environment (CDE)/Motif, PAM provides a mechanism for dynamic system authentication and related services such as password, account, and session management. Realizing that new authentication models continue to be developed, Sun created the PAM architecture that allows additional methods to be added without disturbing existing ones. PAM was introduced in the Solaris 2.6 OE to overcome having to recode system entry services such as, `login`, `passwd`, `dtlogin`, `telnet`, and `rlogin` when a new authentication mechanism was introduced.

The PAM architecture and alternatives to traditional Solaris OE authentication are discussed in Appendix D, "Solaris OE 9 PAM Architecture."

UNIX Passwords

Passwords are created with the Solaris OE `passwd` command. This command prompts the user for a (new) password, which the user enters as a text string. In the Solaris OE, this text string is hashed—or one-way encrypted—using the `crypt(3c)` algorithm. The result is stored either in `/etc/shadow`, or in the `passwd.byname` and `passwd.byuid` NIS maps. If the NIS+ naming service is used, the results are stored in the `Passwd` and `Cred` table type. The `crypt(3c)` algorithm is provided with a random seed, known technically as a *salt string*, so that the result is different each time the `passwd` command is run, even if the same text string is used.

When a user logs in, the Solaris OE `login` program challenges the user to provide a password. This password is hashed in the same manner as the `passwd` command. If the output from this process matches the output that is stored in the password database, the user is authenticated.

FIGURE 3-34 illustrates how the UNIX password process works.

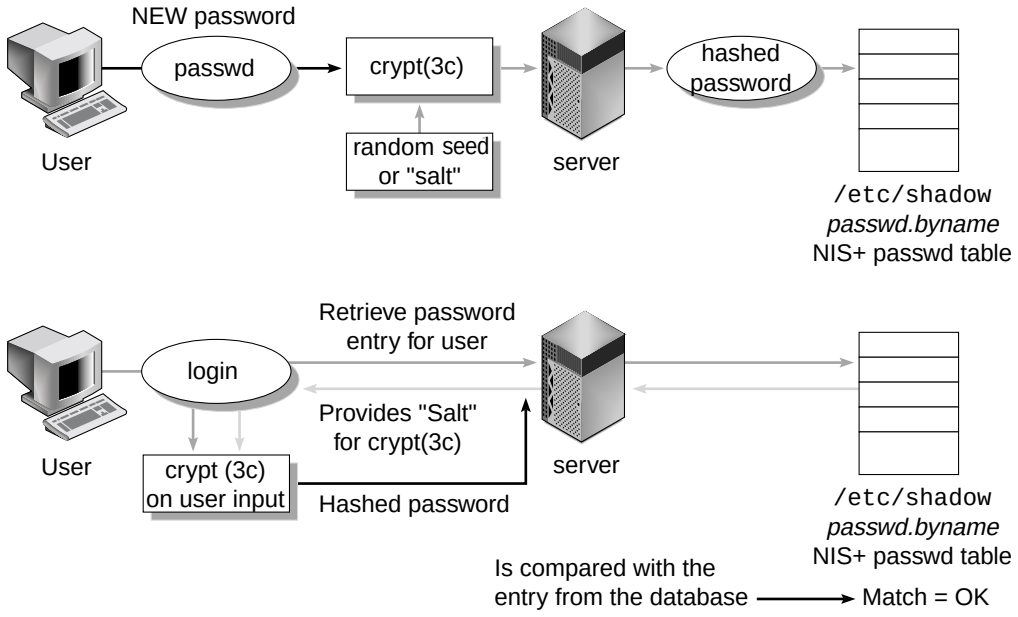

FIGURE 3-34 `Login` Program Text String Converting to a Hashed String

Benefits and Drawbacks of `crypt(3c)`

The major benefit of `crypt(3c)` is that it is easy to implement in a closed environment. Authentication takes place on the host that the user logs in to, so an authentication server is not required. In the case of local logins, the clear text passwords are never stored or sent over the network, so there is no reason to be concerned about eavesdroppers intercepting the password. However, when authenticating over a network using `telnet` or `rlogin`, passwords are sent in clear text.

Because `crypt(3c)` uses a one-way encryption algorithm, it is difficult to decrypt passwords stored on the server. Only the user knows what the actual password is. This means that there is no way to convert passwords stored in crypt to another format required by a different authentication method.

When the `crypt(3c)` function is called, it takes the first eight characters and returns its computation. This computation is then injected with a randomly generated value called the *salt*. In conventional crypt, the salt is stored as the first two characters. This salt value is then added, resulting in a sequence of 13 characters. The result is that the salt is actually an important part of the password string that is stored in the specific naming service.

As CPUs and storage capabilities increase, the `crypt(3c)` algorithm becomes vulnerable to attack. The `crypt(3c)` mechanism shipping with the Solaris 9 OE, along with PAM authentication, is exactly the same implementation that has been in the Solaris OE for many years now, and could one day change.

Introduction to Flexible `crypt(3c)`

The Solaris OE `crypt(3c)` mechanisms work well for authenticating local Solaris OE clients, but they are not the only methods used by applications and services running in the Solaris OE. This can make it difficult for system developers and system administrators, who must work with multiple password systems, and for users who must remember multiple passwords.

In the Solaris 9 12/02 OE, Sun updated the `crypt`(3c) API to allow different algorithms to be used for encrypting the users login password, this is known as *flexible crypt(3c)* for passwords.

The reason this feature was extended is because since the Solaris 2.6 OE, the Solaris OE has supported a `getpassphrase()` routine, which is identical to `getpass(3C)` routine, except it reads and returns a string of up to 256 characters in length. However, the `crypt(3c)` algorithm which it typically provides input to, is still limited to receiving only 8 ASCII characters.

The existing `crypt(3c)` API has been be preserved to provide applications that verify a user's password by calling `crypt(3c)` and using `strcmp(3c)` with the value returned by `getpwnam(3c)` so that they continue to work without any source code change or a recompile. This is obviously a very important aspect when adding any new or enhanced feature.

Functionally, a plug-in framework has been added to `crypt(3c)` to allow the changing of the underlying password hashing algorithm. Currently this ships with two new password hashing algorithms that use the Blowfish and MD5 hashes (for compatibility with BSD / Linux).

By default, the behavior of this new feature that provides extended `crypt(3c)` and adds `crypt_gensalt(3c)` is to use the old UNIX `crypt(3c)` on the password change, unless the user already has a new style password. This feature is turned on and used by changing the settings in the `/etc/security/policy.conf` file, which is the configuration file used for the security policy. For more information refer to `policy.conf(4)`.

The PAM interface in the Solaris 9 OE makes it easier for you to deploy different authentication technologies without modifying administrative commands such as `login`, `telnet`, and other administrative commands. Administrators are able to select one or multiple authentication technologies, without modifying applications

or utilities. PAM can also be an integral part of a single sign-on system. The PAM APIs provide a flexible mechanism that increases overall system security. The PAM APIs are described in Appendix D, “Solaris OE 9 PAM Architecture.”

Solaris 9 OE PAM Framework

The PAM framework enables new authentication technologies to be plugged in without the need to change commands such as `login`, `dtlogin`, `rsh`, `su`, `ftp`, and `telnetd`. PAM is also used to replace the UNIX login with other security mechanisms, such as Kerberos and LDAP authentication. Mechanisms for account, session, and password management can also be plugged in through this framework.

This framework consists of four specific components:

- PAM API presented to the application programs
- PAM framework responsible for implementing the API
- PAM service provider interface (SPI) implements the back-end functionality for the PAM API
- Configuration file `pam.conf` specifies which service providers are used for the various programs

PAM allows you to choose any combination of services to provide authentication. These include a flexible configuration policy that enables a per-application authentication policy, choice of a default authentication mechanism for non-specified applications, and multiple passwords on high security systems. Another valuable service is the ease of use for the end user that enables no retyping of user passwords if the passwords are the same, and optional parameters passed to the services.

With the introduction of the new PAM framework in the Solaris 9 OE, the LDAP service module for PAM has been extended to support the account service, which checks a user’s password and account status by binding to the directory (LDAP) server. The directory server returns the password status to `pam_ldap`, which in turn maps the status to the PAM error codes. A user might be rejected when logging in with an expired password, or might see a warning message after logging in when the password is about to expire.

The `pam_ldap` module has also been updated to support password syntax checking, which is performed through the Sun™ Open Net Environment (Sun ONE) Directory Server 5.1 and greater (formerly known as the iPlanet™ Directory Server software) password policy engine. When changing the password (using the `passwd` command), the user might see error messages such as `password too short`, `password in history`, and so forth. In addition, it adds mechanisms for account lockout after too many failed attempts, forced password change after reset (if reset by root DN in the directory server), minimum password ages, and different password policies for different groups of users.

Note – The `pam_ldap` account management feature is not supported with iPlanet Directory Server software version 5.0.

PAM Types

The PAM framework currently provides four different types of service modules, which are implemented by dynamic loadable module types to provide authentication related services. These modules are categorized based on the function they perform:

- Authentication (`auth`) – Provides authentication for users and enables credentials to be set, refreshed, or destroyed.
- Account management (`account`) – Checks for password aging, account expiration, and access hour restrictions. Once the user is identified by the authentication modules, the account management modules determine whether the user can be given access.
- Session management (`session`) – Manages the opening and closing of a session. The modules can log activity, or clean up after the session is over. For example, the `unix_session` module updates the `lastlog` file.
- Password management (`password`) – Contains functionality that enables the user to change an authentication token (usually a password).

Stacking

PAM enables authentication by multiple methods through stacking. When a user is authenticated through PAM, multiple methods can be selected to fully identify the user. Depending on the configuration, the user can be prompted for passwords for each authentication method. This means that the user need not execute another command to be fully authenticated. The order in which the methods are used is determined through the configuration file, `/etc/pam.conf`.

Note – Stacking has the potential of increased security risk because the security of each mechanism could be subject to the least secure password method used in the stack.

PAM Operation

The PAM software consists of a library, several modules, and a configuration file. The PAM library, `/usr/lib/libpam.so`, provides the framework to load the appropriate modules and manage stacking. It provides a generic structure for all of the modules to plug into.

FIGURE 3-35 illustrates the PAM framework.

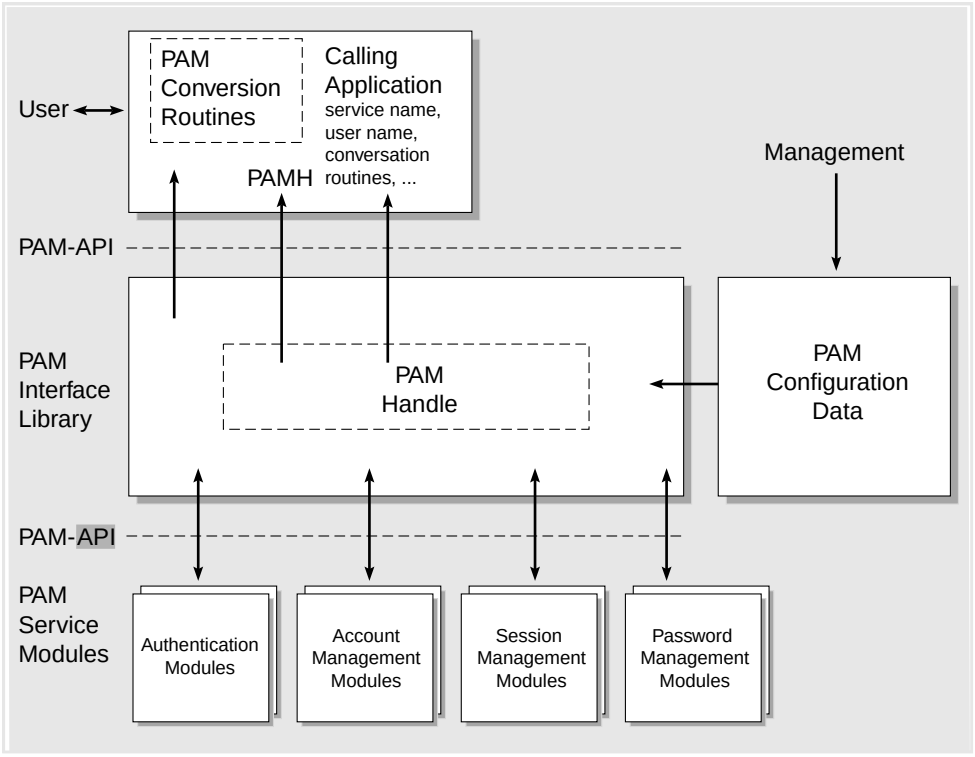

FIGURE 3-35 PAM Framework Architecture

FIGURE 3-36 illustrates the relationship between the applications, the library, and the modules. The `login`, `passwd`, and `su` applications use the PAM library to access the appropriate module. The `pam.conf` file defines which modules are used with each application. Responses from the modules are passed back through the library to the application.

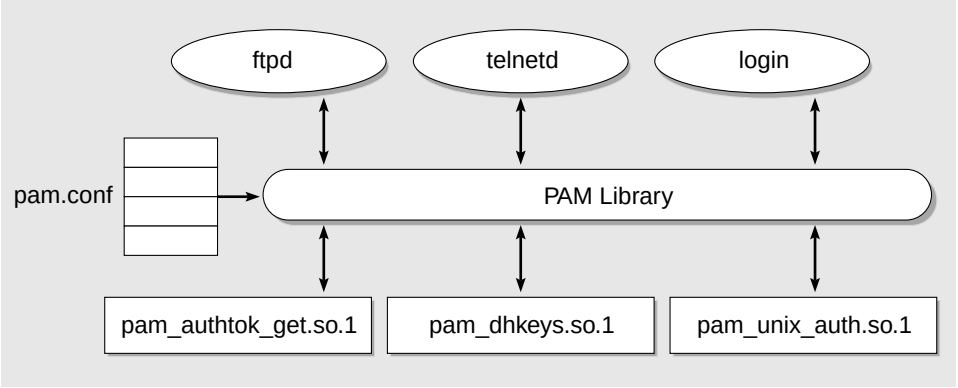

FIGURE 3-36 PAM and the Relationship Between Applications, Library, and Modules

Pluggable Authentication Service Modules

Each module provides the implementation of a specific mechanism. More than one module type (`auth`, `account`, `session`, or `password`) can be associated with each module, but each module needs to manage at least one module type. The following is a description of the modules that are part of the Solaris 9 OE.

- `pam_authtok_get` – Supports authentication and password management. This module takes care of obtaining (old or new) passwords from the user, so that other modules on the stack can concentrate on their task, and not worry about obtaining information from the user.
- `pam_authtok_check` – This module provides functionality to the password management stack. Specifically, it performs a number of checks on the construction of the newly entered password. See `pam_authtok_check(5)` man page for a description of the checks it performs.
- `pam_authtok_store` – Provides functionality to the PAM password management stack. When invoked with flags set to `pam_update_authtok`, this module updates the authentication token for the user specified by `pam_user`.
- `pam_dhkeys` – Supports authentication and password management. This module specifically deals with the establishment and modification of the Diffie-Hellman keys which are used, for example, for secure RPC calls (NIS+ and Secure NFS).
- `pam_passwd_auth` – Provides authentication functionality to the password service as implemented by `passwd(1)`. It differs from the standard PAM authentication modules in its prompting behavior.
- `pam_unix_account` – Provides functionality to the PAM account management stack, as the PAM account management module for UNIX. The `pam_acct_mgmt(3PAM)` function retrieves password aging information from the repositories specified in `nsswitch.conf(4)` and verifies that the user's account and password have not expired.

- pam_unix_auth – Verifies the password that the user has entered against any password repository specified in the nsswitch.conf using normal UNIX crypt(3c) style password encryption, and can only be used for authentication.
- pam_unix_session – Provides functions to initiate and to terminate session as the session management PAM module for UNIX.
- pam_ldap – Implements the functions that provide functionality for the PAM authentication, account management, and password management stacks. (new in Solaris 9 12/02 OE). pam_ldap has also been updated in Solaris 9 OE 12/02 to support password syntax checking, which is done through the Sun ONE Directory Server password policy engine.

In addition to the above pam_ldap service module, a new server_policy option can be specified with the pam_unix_auth, pam_unix_account, pam_passwd_auth, and pam_authtok.store modules. This option instructs these modules to ignore a user if the user is only found in the directory server (LDAP) repository, and let the stacked below pam_ldap module to process the user according to the password policy set in the Sun ONE Directory Server software.

For security, these files must be owned by root and have their permissions set so that the files are *not* writable through group or other permissions. If the file is not owned by root, PAM will not load the module. This requirement on permissions and owner for the modules is not documented anywhere, and might change in future releases.

Note – In FIGURE 3-36, pam_unix is not layered entirely on the LDAP server. The pam_unix module sits on the Name Service Switch (NSS) layer and the NSS back ends that could be files, NIS, NIS+, or LDAP.

PAM Configuration File Update

The PAM configuration file, /etc/pam.conf, determines what authentication services are used and in what order. Edit this file to select the desired authentication mechanisms for each system entry application.

Configuration File Syntax

The PAM configuration file consists of entries with the following syntax:

service_name module_type control_flag module_path module_options

TABLE 3-9 explains the functions of the syntax.

TABLE 3-9 Configuration File Syntax

Syntax	Description
service_name	Name of the service (for example, `ftp`, `login`, `telnet`)
module_type	Module type for the service (`auth`, `account`, `session`, `password`)
control_flag	Determines the continuation or failure semantics for the module (see note below)
module_path	Pathname of the module
module_options	Specific options passed to the service modules

Comments can be added to the `pam.conf` file by starting the line with a pound sign (#). Use white space to delimit the fields.

Note – An entry in the PAM configuration file is ignored if one of the following conditions exists: the line has fewer than four fields, an invalid value is given for *module_type* or *control_flag*, or the named module is not found.

TABLE 3-10 summarizes PAM configurations.

TABLE 3-10 PAM Configurations

Service Name	Daemon or Command	Module Type
`cron`	`/usr/sbin/cron`	`account`
`dtlogin`	`/usr/dt/bin/dtlogin`	`auth, account, session`
`ftp`	`/usr/sbin/in.ftpd`	`auth, account, session`
`init`	`/usr/sbin/init`	`session`
`login`	`/usr/bin/login`	`auth, account, session, password`
`passwd`	`/usr/bin/passwd`	`auth, account, password`
`ppp`	`/usr/bin/pppd`	`auth, account, session`
`rexecd`	`/usr/sbin/in.rexecd`	`auth, account`
`rexd`	`/usr/sbin/rpc.rexd`	`account, session`
`rlogin`	`/usr/sbin/in.rlogind`	`auth, account, session, password`
`rsh`	`/usr/sbin/in.rshd`	`auth, account`

TABLE 3-10 PAM Configurations *(Continued)*

Service Name	Daemon or Command	Module Type
sac	/usr/lib/saf/sac	session
sshd	/usr/lib/ssh/sshd	auth, account, session, password
su	/usr/bin/su	auth, account
telnet	/usr/sbin/in.telnetd	auth, account, session, password
ttymon	/usr/lib/saf/ttymon	session
uucp	/usr/sbin/in.uucpd	auth, account

Control Flags

To determine continuation or failure behavior from a module during the authentication process, you must select one of four control flags for each entry. Successful or failed attempts are indicated through control flags. Even though these flags apply to all module types, the following explanation assumes that the flags are being used for authentication modules. The control flags are as follows:

`required` – This module must return `success` in order to have an overall successful result. If all of the modules are labeled as `required`, authentication through all modules must succeed for the user to be authenticated. If some of the modules fail, an error value from the first failed module is reported. If a failure occurs for a module flagged `required`, all modules in the stack are still tried but failure is returned. If none of the modules are flagged `required`, at least one of the entries for that service must succeed for the user to be authenticated.

`requisite` – This module must return `success` for additional authentication to occur. If a failure occurs for a module flagged `requisite`, an error is immediately returned to the application and no additional authentication is done. If the stack does not include prior modules labeled `required` that failed, the error from this module is returned. If a earlier module labeled `required` has failed, the error message from the required module is returned.

`optional` – If this module fails, the overall result can be successful if another module in this stack returns `success`. The `optional` flag should be used when one success in the stack is enough for a user to be authenticated. This flag should only be used if it is not important for this particular mechanism to succeed. If your users need to have permission associated with a specific mechanism to get their work done, you should not label it `optional`.

`sufficient` – If this module is successful, skip the remaining modules in the stack, even if they are labeled `required`. The `sufficient` flag indicates that one successful authentication is enough for the user to be granted access. More information about these flags is provided in the next section, which describes the default `/etc/pam.conf` file.

`binding` – This is a new control flag that has been added to the PAM framework in Solaris 9 12/02 OE. The control flag `binding` has a meaning of *terminate processing upon success, and report the failure if unsuccessful.* This option effectively provides a local account overriding remote (LDAP) account functionality.

Generic `pam.conf` File

The following is an example of a generic `pam.conf` file:

```
# PAM configuration
# Authentication management
#
login   auth requisite pam_authtok_get.so.1
login   auth sufficient pam_unix_auth.so.1
login   auth required pam_ldap.so.1
#
rlogin  auth sufficient pam_rhosts_auth.so.1
rlogin  auth required   pam_authtok_get.so.1
rlogin  auth sufficient pam_unix_auth.so.1
#
dtlogin auth required   pam_authtok_get.so.1
dtlogin auth required   pam_unix_auth.so.1
#
rsh     auth sufficient pam_rhosts_auth.so.1
rsh     auth required   pam_unix_auth.so.1
#
dtsession auth required pam_authtok_get.so.1
dtsession auth required pam_unix_auth.so.1
#
other   auth required   pam_authtok_get.so.1
other   auth required   pam_unix_auth.so.1
#
# Account management
#
login   account requisite       pam_roles.so.1
login   account required        pam_projects.so.1
login   account required        pam_unix_account.so.1
#
dtlogin account requisite       pam_roles.so.1
dtlogin account required        pam_projects.so.1
```

```
dtlogin account required        pam_unix_account.so.1
#
cron    account required        pam_projects.so.1
#
cron    account required        pam_unix_account.so.1
#
other   account requisite       pam_roles.so.1
other   account required        pam_projects.so.1
other   account required        pam_unix_account.so.1
# Session management
#
other   session required        pam_unix_session.so.1
#
# Password management
#
other   password requisite      pam_authtok_get.so.1
other   password requisite      pam_authtok_check.so.1
other   password sufficient     pam_authtok_store.so.1
other   password required       pam_ldap.so.1
```

This generic `pam.conf` file specifies the following behavior:

- When running `login`, authentication must succeed for the `pam_authtok_get` module and for either the `pam_unix_auth` or the `pam_ldap` module.
- For `rlogin`, authentication through the `pam_authtok_get` and `pam_unix_auth` modules must succeed if authentication through `pam_rhost_auth` fails.
- The `sufficient` control flag for `rlogin's` `pam_rhost_auth` module indicates that if the authentication performed by the `pam_rhost_auth` module is successful, the remainder of the stack is not executed, and a success value is returned.
- Most of the other commands requiring authentication require successful authentication through the `pam_unix_auth` module.

Note – With the above configuration, `pam_unix` is tried first, and if the `userPassword` attribute is readable, and the password is correct, then the `pam_ldap` module is not called. As a result, the `pam_ldap` password management is not used.

The `other` service name allows a default to be set for any other commands requiring authentication that are not included in the file. The `other` option makes it easier to administer the file because many commands that use the same module can be covered by only one entry. Also, the `other` service name, when used as a catchall, can ensure that each access is covered by one module. By convention, the

other entry is included at the bottom of the section for each module type. The rest of the entries are in the file control account management, session management, and password management.

Normally, the entry for the module_path is root relative. If the file name entered for module_path does not begin with a slash (/), the path /usr/lib/security/$ISA is added to the file name, where $ISA is expanded by the framework to contain the instruction set architecture of the executing machine (refer to the isainfo(1) man page for additional information).

A full path name must be used for modules located in directories other than the default. The values for the module_options can be found in the man pages for the module (for example, pam_unix_auth(5)).

If login specifies authentication through both pam_unix_auth and pam_ldap, the user is prompted to enter a password for each module. Example:

```
# Authentication management
#
login auth required pam_authtok_get.so.1
login auth sufficient pam_unix_auth.so.1
login auth required pam_ldap.so.1
```

PAM and LDAP Password Management Extensions

It is important to provide a quick overview to clarify the difference between PAM Password Management Extensions and the new pam_ldap password management.

PAM Password Management Extensions provide the same functionality as the existing pam_unix module. The only difference is *how* the module is packaged. What used to be a single module is now split up into multiple components, known as *service modules*, each performing a separate function. This modular construction makes implementing custom password management policies easier.

The new pam_ldap password management facility includes two new account management features: *password aging* and *account expiration*. Because the directory server provides its own mechanism for account management, a conflict can occur if you want pam_ldap to implement a different password policy than what is set for the directory-wide policy. For example, the directory might force all users to change passwords after 60 days, but you might want some special user accounts to be able to keep their current password for a longer period of time.

To support this flexibility, the PAM framework has been enhanced by the addition of a new control flag called `binding`. The primary reason this *control* flag was introduced was the fact that prior to Solaris 9 12/02 OE, the PAM framework lacked sufficient control flags to provide functionality needed to return the appropriate failure semantics for service modules which should return immediately upon success, but report its error upon failure. In particular, `pam_ldap` depends on this change to correctly provide failure semantics for a mixture of local and server controlled accounts on the same machine. Effectively, this control flag allows you to override the password policy that the directory server enforces.

A `server_policy` option has been added to instruct `pam_unix` to allow users that only have LDAP accounts to be processed by the password policy set on the directory server. This option can be used to instruct the `pam_unix_account`, `pam_unix_auth`, and `pam_passwd_auth` service modules to ignore the user being authenticated and let the `pam_ldap` module stacked below them process the user according to the password policy established in the directory server. This effectively allows you to override the local `pam_unix` password policy.

Note – The `pam_authtok_store` module handles this option differently.

The `server_policy` option was introduced to solve a problem found when stacking the `pam_unix_account` and `pam_ldap` modules together. When used, it tells the module to rely on the policy specified on the LDAP server and not to apply a local policy.

Because the `pam_unix_account` receives incomplete information from the LDAP server, it might inadvertently decide that an active account has expired, or that an expired account is still active. Specifying `server_policy` in `/etc/pam.conf` tells `pam_unix_account` not to guess an account's status but to leave the decision to the LDAP server. The LDAP server keeps accurate current status of each account and can draw the correct conclusion about its expiration status.

Because this feature enables the `pam_ldap` module to fully support the account management, it is reasonable to use the following PAM configuration for account management.

```
other  account  requisite  pam_roles.so.1
other  account  required   pam_projects.so.1
other  account  binding pam_unix_account.so.1 server_policy
other  account  required   pam_ldap.so.1
```

Note – In this configuration, note the binding control flag for `pam_unix_account.so.1`.

This configuration specifies that the pam_unix_account should check the user's local account first. Because of the binding control flag, the stack succeeds or fails depending on the values returned by the pam_unix_account. If only the LDAP account exists for the user, the pam_unix_account does nothing and allows pam_ldap to determine the stack's success or failure.

Customer feedback indicated that the PAM functionality in the Solaris OE needed some enhancements. The requested changes included improving the mechanism used to validate password structures, adding the ability to change numbers of characters, total password length, and so forth.

In previous versions of the Solaris OE, this functionality was tightly coupled in a single monolithic module (pam_unix) and local extensions could not be incorporated in the module.

Only with a great deal of effort could you extend part of the operations performed by this module. Because of this, the pam_unix(5) functionality has been replaced with a new set of modular PAM service modules that are listed in this section. The functionality of pam_unix has been entirely replaced in the Solaris 9 OE. New PAM modules are now provided that replace a specific piece of pam_unix. This makes it easier to customize the PAM behavior by inserting or replacing individual modules. The Solaris 9 OE no longer uses pam_unix by default. During upgrades, any existing instances of pam_unix in pam.conf are replaced by the new modules.

In the Solaris 9 OE, the functionality provided by the old pam_unix module has been split over a number of small modules, each performing a well-defined task, that can be easily extended or replaced by modifying the pam.conf file.

These new modules are:

- pam_authtok_get(5)
- pam_authtok_check(5)
- pam_authtok_store(5)
- pam_unix_auth(5)
- pam_dhkeys(5)
- pam_unix_account(5)
- pam_unix_session(5)

You no longer have to replace the pam_authtok_check module to extend or replace the standard password strength checks. Just list the module in the /etc/pam.conf file right before, after, or instead of the pam_authtok_check file.

▼ To Add a PAM Module

1. **Determine the control flags and other options you want to use.**
2. **Become superuser.**
3. **Copy the new module to** /usr/lib/security**.**

Note – If you have a 64-bit version of the module, you should place that version in `/usr/lib/security/sparcv9`.

4. **Set the permissions so that the module file is owned by `root` and the permissions are `755`.**
5. **Edit the PAM configuration file, `/etc/pam.conf`, to add this module to the appropriate services.**

▼ To Verify the Configuration

It is essential to do some testing before logging out, in case the configuration file is misconfigured.

1. **Test the modified service or the other configuration.**
2. **Run `rlogin`, `su`, and `telnet` (if these services have been changed).**

   If the service is a daemon spawned only once when the system is booted, it might be necessary to reboot the system before you can verify that the module has been added, however it might be possible to restart the daemon using the appropriate `/etc/init.d/` script.

▼ To Disable `.rhosts` Access With PAM From Remote Systems

A common use of the `.rhosts` file is to simplify remote logins between multiple accounts owned by the same user. For example, if you have multiple accounts on more than one system, you might need to perform specific tasks, and using the `.rhosts` file is ideal.

However, using the `.rhosts` file as an authentication mechanism is a weak form of security and should be avoided.

- **Remove the `rlogin` and `rsh` (`pam_rhosts_auth.so.1`) entries from the PAM configuration file.**

  This prevents reading the `~/.rhosts` files during an `rlogin` session, and therefore, prevents unauthenticated access to the local system from remote systems. All `rlogin` access requires a password, regardless of the presence or contents of any `~/.rhosts` or `/etc/hosts.equiv` files.

Note – To prevent other unauthenticated access to the `~/.rhosts` file, remember to disable the `rsh` service. The best way to disable a service is to remove the service entry from `/etc/inetd.conf`. The remote shell server, `rshd`, and the remote login server, `rlogind`, only use PAM; they do not call the `ruserok()` function themselves.

PAM Error Reporting

Diagnostic messages generated by the PAM modules or the PAM framework are output using `syslog(3c)`. They are logged to the facility that was specified at the time the application (`login`, `telnet`, `sshd`) called `openlog(3c)`, so the exact location of these messages depends upon whether the application uses PAM. The facility indicates the application or system component generating the message. As an example, here are a few possible facility values:

- `LOG_KERN` – Messages generated by the kernel. These cannot be generated by any user processes.
- `LOG_USER` – Messages generated by random user processes. This is the default facility identifier if none is specified.
- `LOG_MAIL` – The mail system.

For example, `login` sends its messages to the `LOG_AUTH` facility, while `rlogind` sends its messages to the `LOG_DAEMON` facility. Other daemons might use a configurable facility (`sshd`, `ftpd`, and so forth) which can be set in the configuration file of the particular service.

Depending on the severity of the diagnostic message, the PAM module directs the message to one of the eight available log priorities.

Note – For additional details on the `syslog()` function and priorities, refer to the `syslog(3c)` and `syslog.conf(4)`man pages.

Debug messages are logged with:

```
syslog(LOG_DEBUG, "...")
```

Critical messages are logged with:

```
syslog(LOG_CRIT, "...")
```

For example, a general error message (LOG_ERR) from PAM, used by login, is directed to auth.crit and ends up in a log file as:

```
Jul 22 22:11:43 host login: [ID 887986 auth.error]
ACCOUNT:pam_sm_acct_mgmt: illegal option debuf
```

▼ To Initiate Diagnostics Reporting for PAM

1. **Back up the** syslog.conf **file before editing it.**
2. **Determine the** syslog **facility used by the application you want to receive diagnostic reports from.**

   The facility that we are going to use in this example is auth.
3. **Edit the** /etc/syslog.conf **to add a line describing where the message with the intended facility and priority will be logged.**

   Example of line added:

   ```
   auth.debug /var/adm/authlog
   ```

   Note that these message levels are part of a hierarchy:

   ```
   High ------------------------------------Low
   EMERG ALERT CRIT ERR WARNING NOTICE INFO DEBUG
   ```

   Due to this hierarchical ordering, a syslog channel specified to log debug messages also logs messages at all higher levels (for example, logs messages with priority debug *and up*).
4. **Make sure that the log file specified in the previous step actually exists.**

   If it doesn't exist, create it now.

   Example:

   ```
   # touch /var/adm/authlog
   ```
5. **Make** syslogd **re-read the configuration file by sending it a HUP signal:**

   ```
   # pkill -HUP syslogd
   ```

▼ To Initiate PAM Error Reporting

The following example displays all alert messages on the console. Critical messages are mailed to `root`. Debug messages are added to `/var/log/pamlog`.

```
auth.alert /dev/console
auth.crit root
auth.debug /var/log/pamlog
```

Each line in the log file contains a timestamp, the name of the system that generated the message, and the message itself. Be aware that a large amount of information may be written to the `pamlog` file.

The log format was changed in the Solaris 8 OE and subsequent releases, and now includes a hash-value of the message generating string for example—`user %s not found`. It now contains the message facility and severity.

- **Add the** `debug` **flag to a PAM module to enable diagnostics reporting of that module.**

  Example:

```
# PAM Module Debugging
#
login   auth requisite          pam_authtok_get.so.1
login   auth required           pam_dhkeys.so.1         debug
login   auth required           pam_unix_auth.so.1      debug
login   auth required           pam_dial_auth.so.1
```

  This configuration example enables debugging information from `pam_dhkeys.so.1` and `pam_unix_auth.so.1`.

  What gets logged might vary quite a bit, because there is no standard describing the information that needs to be output in response to this option. It is a good practice for module developers to recognize this `debug` flag and enable some form of debugging when the flag is specified in `/etc/pam.conf`.

PAM LDAP Module

The PAM LDAP module (`pam_ldap`) was introduced in the Solaris 8 OE for use in conjunction with `pam_unix` for authentication and password management with an LDAP server. This module was written to support stronger authentication methods such as CRAM-MD5, in addition to the other UNIX authentication capabilities provided by `pam_unix`.

Note – The pam_ldap module must be used in conjunction with the modules supporting the UNIX authentication, password and account management, because pam_ldap is designed to be stacked directly below these modules.

With the release of Solaris 9 12/02 OE, pam_ldap provides support for authentication, account management, and password management.

The pam_ldap module should be stacked directly below the pam_unix module in the configuration file /etc/pam.conf. If there are other modules that are designed to be stacked in this manner, they could be stacked *under* the pam_ldap module. This design must be followed in order for authentication and password management to work when pam_ldap is used. The following is a sample of /etc/pam.conf file with pam_ldap stacked under pam_unix:

```
# Authentication management for login service is stacked.
# If pam_unix succeeds, pam_ldap is not invoked.
login   auth sufficient /usr/lib/security/pam_unix.so.1
login   auth required /usr/lib/security/pam_ldap.so.1
# Password management
other   password sufficient /usr/lib/security/pam_unix.so.1
other   password required /usr/lib/security/pam_ldap.so.1
```

It is important to note that the control flag for pam_unix is sufficient. This flag means that if authentication through pam_unix succeeds, then pam_ldap is not invoked. Also, other service types, such as dtlogin, su, telnet, and so forth can substitute for login. See FIGURE 3-37.

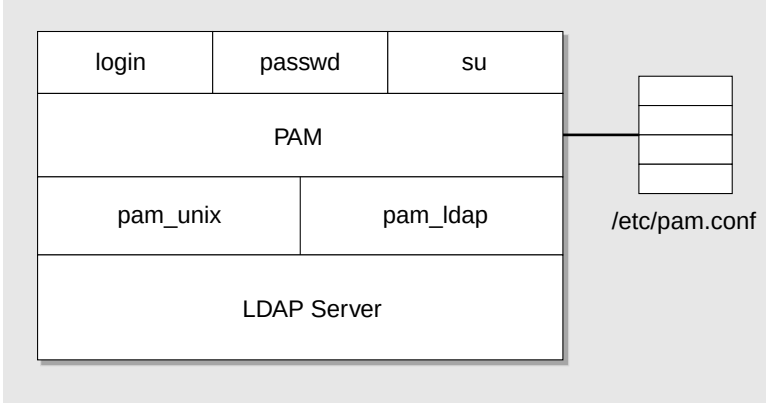

FIGURE 3-37 pam_ldap Structure

The options supported by `pam_ldap` are:

- `debug` – If this option is used with `pam_ldap`, debugging information is output to the `syslog(3C)` files.
- `nowarn` – This option turns off warning messages.

How PAM and LDAP Work

Before discussing the details of how PAM and LDAP work, it is important to provide a quick overview to distinguish between how the password is stored and how the authentication mechanism is used to authenticate to the LDAP server. The password can be stored in a variety of formats in the directory server, such as salted secure hash algorithm (SSHA), secure hash algorithm (SHA), CRYPT, and so forth.

The authentication mechanisms currently used and supported in the Solaris 8 OE LDAP Client, are NONE, SIMPLE, and CRAM-MD5 authentication. Simple authentication requires the client to pass a distinguished name (DN) and password to the server in clear text. Currently, the Sun ONE Directory Server 5.x software does not support the authentication mechanism CRAM-MD5, which sends only the digest over the wire. CRAM-MD5 is implemented as a Simple authentication and security layer (SASL) mechanism, and both the client and server must use it. What happens is the client request authentication is based on SASL/CRAM-MD5 and the server must support this to complete the authentication. In general, very few clients use CRAM-MD5, now that RFC 2829 mandates the use of DIGEST-MD5, which is intended to be an improvement over CRAM-MD5.

Note – DIGEST-MD5 as an authentication mechanism for LDAPv3 directory servers is mandated in RFC 2829. RFC 2831 provides information about DIGEST-MD5 as a SASL mechanism, but is not LDAP specific.

With the introduction of the Sun ONE Directory Server 5.2 software, support for SASL/DIGEST-MD5 has also been added as an authentication mechanism. This feature was initially introduced in the Sun ONE Directory Server 5.1 software release.

With SASL/DIGEST-MD5, a digest is created and sent across the wire to authenticate to the directory server. The directory server then compares the digest that was sent with the digest created by itself with the stored password and returns success if it matches. In this case, the password is *not* sent in clear text. To address the absence of a security model in the Solaris 8 OE LDAP Client, the Solaris 9 OE now incorporates the Sun ONE Directory Server 5.1 software and Solaris 9 OE Secured LDAP Client, addressing the security issues found in the LDAP Client.

To use SASL/DIGEST-MD5, the Sun ONE Directory Server software requires that the password is stored in the directory in the clear. In the Sun ONE Directory Server 5.2 software, you need to make sure that you enable the SASL mechanism that you

wish to use. Also there is support for identity mapping which was covered previously. The identity mapping allows for quite a bit of flexibility. For the Sun ONE Directory Server 5.1 release, two forms are supported, which are `dn:` and `u:` as specified in the RFC. This has built-in rules to handle the identity mapping.

Note – With identity mapping, you must map to one, and only one, identity.

Note – In the current release of the Solaris 9 OE, the extended Start TLS operation is not supported.

Authentication With `pam_unix`

In authentication with `pam_unix`, depending on how the client is configured, the client retrieves the password that is stored in the server by making a call to the `getspnam` function. This function binds to the LDAP server with the proxy agent account (the reason the proxy `passwd` is sent across the wire in clear text). The proxy agent password is stored in the `userPassword` attribute in the directory server. This proxy agent account can reside anywhere in the directory server, but must contain the `userPassword` attribute.

Note that the ACIs of the proxy agent allow this account to have read access to all user passwords, which you may not want to do if you are using `pam_ldap`. ACIs are instructions that are stored in the directory server itself. Every entry can have a set of rules that define an ACI for that entry. An ACI appears as an attribute in the entry so it can be retrieved by using LDAP search, or it can be added, updated, or deleted with an LDAP modify operation.

An entry may have one ACI, many ACIs, or none. ACIs allow or deny permissions to entries. When the directory server processes an incoming request for that entry, the server uses the ACIs specific to that entry to determine whether or not the LDAP client has permission to perform the requested operation.

Note – LDAP stores data as entries. An entry has a distinguished name (DN) to uniquely identify it within the directory server

---

The encrypted password is sent to the client side and compared with the encrypted password supplied by the user at the password prompt. If there is a match, `pam_unix` returns `success`. The following tables illustrate the authentication mechanisms currently used.

TABLE 3-11 lists the PAM abbreviations used in this section.

TABLE 3-11 PAM Abbreviations

Abbreviation	Description
UP	User password
PP	Proxy agent password
NP	New password
NO*	Not applicable (at present)

TABLE 3-12 illustrates if the user password and proxy password are transmitted in the clear during PAM authentications.

TABLE 3-12 PAM Authentication

Authentication Mechanisms	pam_unix		pam_ldap	
SIMPLE	UP-No	PP-Yes	UP-Yes	PP-Yes
DIGEST-MD5	UP-NO*	PP-No	UP-No	PP-No
TLS: SIMPLE	UP-No	PP-No	UP-No	PP-No
TLS: DIGEST-MD5	UP-No	PP-No	UP-No	PP-No

Note – In TABLE 3-11 and TABLE 3-12 the reason for "NO*" as the value of the DIGEST-MD5 UP column is because the Sun ONE Directory Server version 5.1 software requires that passwords be stored in the server in clear text for DIGEST-MD5 to work.

For updating passwords in pam_unix, the same comparison as for authentication takes place (because the user has to bind as the dn); then the new password is encrypted and *not* passed over the wire in clear text (TABLE 3-13).

TABLE 3-13 PAM Update of Password

Authentication Mechanisms	pam_unix			pam_ldap		
SIMPLE	UP-No	PP-Yes	NP-No	UP-Yes	PP-Yes	NP-Yes
DIGEST-MD5	UP-NO*	PP-No	NP-NO*	UP-No	PP-No	NP-Yes
TLS: SIMPLE	UP-No	PP-No	NP-No	UP-No	PP-No	NP-No
TLS: DIGEST-MD5	UP-No	PP-No	NP-No	UP-No	PP-No	NP-No

The matrices are easier to understand when you distinguish between how the password is stored and how the authentication mechanism is used to authenticate to the LDAP server. The password can be stored in a variety of formats, such as SSHA, SHA, crypt, clear text, and so forth. The authentication mechanisms that are currently supported are NONE, SIMPLE, SASL/CRAM-MD5, SASL/DIGEST-MD5, TLS:NONE, TLS:SIMPLE, TLS:SASL/CRAM-MD5, and TLS:SASL/DIGEST-MD5.

pam_ldap *Authentication*

In authentication that uses pam_ldap, the user password is passed to the server in an auth structure in clear text because authentication is being attempted with the user dn and password. If Simple authentication is used, and the password matches, then success is returned. Using pam_ldap in the Solaris 9 OE Secured LDAP Client now provides SASL/DIGEST-MD5 authentication, privacy, and data integrity with SSL/TLS. If you require stronger authentication mechanisms such as DIGEST-MD5; then you must use pam_ldap. In addition, pam_ldap is designed to be extended for future authentication mechanisms that will be supported in future Solaris OE releases. One of the benefits of using pam_ldap, is that it does not require passwords to be stored in any specific format, so you can store passwords using SSHA, SHA, or CRYPT formats.

For additional information, see the pam_ldap man page for the correct way to stack the authentication management for login service, and password management modules in the /etc/pam.conf configuration file.

Note – CRAM-MD5 is supported by the Secured LDAP Client, but not by the Sun ONE Directory Server software. However, DIGEST-MD5 is supported by both.

Secured LDAP Client Backport to the Solaris 8 OE

Now that we have touched on the Solaris 9 OE Secured LDAP clients, which have the option to use TLSv1 and SASL/DIGEST-MD5 for authentication, we can discuss what has been done with the Solaris 8 OE LDAP clients. Initially as previously discussed, the Solaris 8 OE LDAP clients relied on clear text passwords or the less secure SASL/CRAM-MD5 for authentication. This is obviously not desirable for customers that wanted to deploy a secure naming service, and also maintain equal and matching functionality in both the Solaris 8 and 9 OE.

Note – The Sun ONE directory server does not support SASL/CRAM-MD5.

With this in mind, Sun backported the Secured LDAP Client found in the Solaris 9 OE to the Solaris 8 OE to provide TLSv1/ and SASL/DIGEST-MD5 support for the LDAP client. The following lists what functionality has been backported:

- The configuration of the directory server (LDAP) setup has been simplified with the use of `idsconfig`.
- A more robust security model that supports strong authentication and Transport Layer Security (TLS) encrypted sessions. A client's proxy credentials are no longer stored in a client s profile on the directory server.
- The `ldapaddent` command allows you to populate and dump data onto the server.
- Service search descriptors and attribute mapping
- New profile schema
- PAM Framework including account management
- Updated man pages include:
  - `ldaplist(1)`
  - `ldapaddent(1)`
  - `pam_authok_check(5)`
  - `pam_authok_get(5)`
  - `pam_authok_store(5)`
  - `pam_passwd_auth(5)`
  - `pam_unix_auth(5)`
  - `pam_conf(4)`

You can obtain the Secured LDAP Client Backport for the Solaris 8 OE from:

`http://sunsolve.sun.com/pub-cgi/show.pl?target=patches/patch-access`

In the Enter a Patch ID field, enter one of the following patches:

- 108993-xx (SPARC™ systems)
- 108994-xx (x86 systems)