A

Configuring Samba with SSL

This appendix describes how to set up Samba to use secure connections between the Samba server and its clients. The protocol used here is Netscape's Secure Sockets Layer (SSL). For this example, we will establish a secure connection between a Samba server and a Windows NT workstation.

Before we begin, we will assume that you are familiar with the fundamentals of public-key cryptography and X.509 certificates. If not, we highly recommend Bruce Schneier's *Applied Cryptography, 2nd Edition* (Wiley) as the premiere source for learning the many secret faces of cryptography.

If you would like more information on Samba and SSL, be sure to look at the document *SSLeay.txt* in the *docs/textdocs* directory of the Samba distribution, which is the basis for this appendix.

About Certificates

Here are a few quick questions and answers from the *SSLeay.txt* file in the Samba documentation, regarding the benefits of SSL and certificates. This text was written by Christian Starkjohann for the Samba projects.

What is a Certificate?

A certifcate is issued by an issuer, usually a *Certification Authority* (CA), who confirms something by issuing the certificate. The subject of this confirmation depends on the CA's policy. CAs for secure web servers (used for shopping malls, etc.) usually attest only that the given public key belongs the given domain name. Com-

pany-wide CAs might attest that you are an employee of the company, that you have permissions to use a server, and so on.

What is an X.509 certificate, technically?

Technically, the certificate is a block of data signed by the certificate issuer (the CA). The relevant fields are:

- Unique identifier (name) of the certificate issuer
- Time range during which the certificate is valid
- Unique identifier (name) of the certified object
- Public key of the certified object
- The issuer's signature over all the above

If this certificate is to be verified, the verifier must have a table of the names and public keys of trusted CAs. For simplicity, these tables should list certificates issued by the respective CAs for themselves (self-signed certificates).

What are the implications of this certificate structure?

Four implications follow:

- Because the certificate contains the subjects's public key, the certificate and the private key together are all that is needed to encrypt and decrypt.
- To verify certificates, you need the certificates of all CAs you trust.
- The simplest form of a dummy-certificate is one that is signed by the subject.
- A CA is needed. The client can't simply issue local certificates for servers it trusts because the server determines which certificate it presents.

Requirements

To set up SSL connections, you will need to download two programs in addition to Samba:

SSLeay
: Eric Young's implementation of the Secure Socket's Layer (SSL) protocol as a series of Unix programming libraries

SSL Proxy
: A freeware SSL application from Objective Development, which can be used to proxy a secure link on Unix or Windows NT platforms

These two products assist with the server and client side of the encrypted SSL connection. The SSLeay libraries are compiled and installed directly on the Unix system. SSL Proxy, on the other hand, can be downloaded and compiled (or downloaded in binary format) and located on the client side. If you intend to have a Windows NT client or a Samba client on the other end of the SSL connection, you will not require a special setup.

SSL Proxy, however, does not work on Windows 95/98 machines. Therefore, if you want to have a secure connection between a Samba server and Windows 95/98 client, you will need to place either a Unix server or a Windows NT machine on the same subnet with the Windows 9*x* clients and route all network connections through the SSL-Proxy-enabled machine. See Figure A-1.

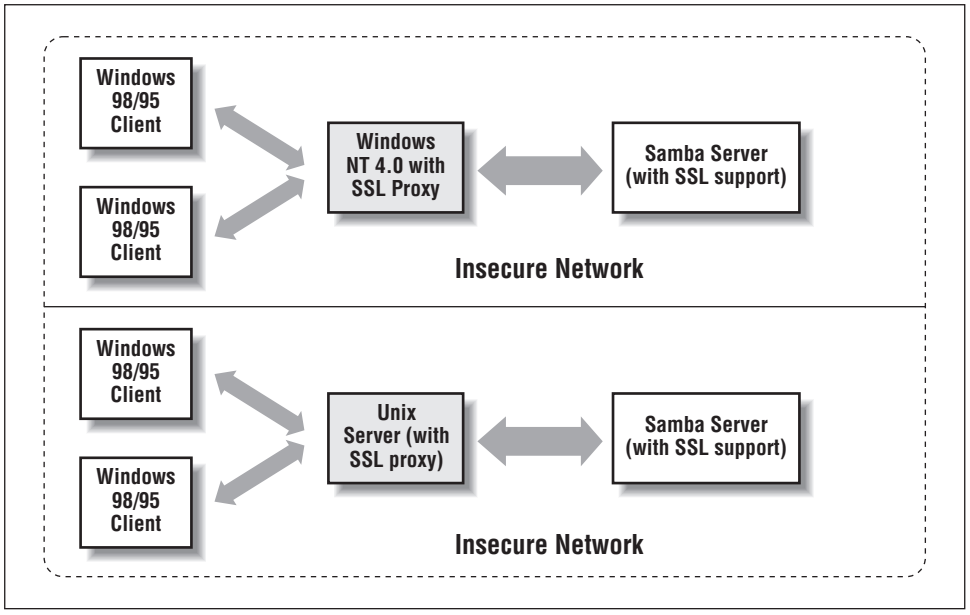

Figure A-1. Two possible ways of proxying Windows 95/98 clients

For the purposes of this chapter, we will create a simple SSL connection between the Samba server and a Windows NT client. This configuration can be used to set up more complex networks at the administrator's discretion.

Installing SSLeay

Samba uses the SSLeay package, written by Eric Young, to provide Secure Sockets Layer support on the server side. Because of U.S. export law, however, the SSLeay package cannot be shipped with Samba distributions that are based in the United States. For that reason, the Samba creators decided to leave it as a separate

package entirely. You can download the SSLeay distribution from any of the following sites:

- *ftp://ftp.psy.uq.oz.au/pub/Crypto/SSL/*
- *ftp://ftp.uni-mainz.de/pub/internet/security/ssl*
- *ftp://ftp.cert.dfn.de/pub/tools/crypt/sslapps*
- *ftp://ftp.funet.fi/pub/crypt/mirrors/ftp.psy.uq.oz.au*
- *ftp://ftp.sunet.se/ftp/pub/security/tools/crypt/ssleay*

The latest version as of this printing is 0.9.0b. Download it to the same server as the Samba distribution, then uncompress and untar it. You should be left with a directory entitled *SSLeay-0.9.0b*. After changing to that directory, you will need to configure and build the SSL encryption package in the same way that you did with Samba.

SSLeay uses a Perl-based *configure* script. This script modifies the Makefile that constructs the utilities and libraries of the SSLeay package. However, the default script is hardcoded to find Perl at */usr/local/bin/perl*. You may need to change the *configure* script to point to the location of the Perl executable file on your Unix system. For example, you can type the following to locate the Perl executable:

```
# which perl
/usr/bin/perl
```

Then modify the first line of the *configure* script to force it to use the correct Perl executable. For example, on our Red Hat Linux system:

```
#!/usr/bin/perl
#
# see PROBLEMS for instructions on what sort of things to do
# when tracking a bug -tjh
...
```

After that, you need to run the *configure* script by specifying a target platform for the distribution. This target platform can be any of the following:

```
BC-16              BC-32              FreeBSD            NetBSD-m86
NetBSD-sparc       NetBSD-x86         SINIX-N            VC-MSDOS
VC-NT              VC-W31-16          VC-W31-32          VC-WIN16
VC-WIN32           aix-cc             aix-gcc            alpha-cc
alpha-gcc          alpha400-cc        cc                 cray-t90-cc
debug              debug-irix-cc      debug-linux-elf    dgux-R3-gcc
dgux-R4-gcc        dgux-R4-x86-gcc    dist               gcc
hpux-cc            hpux-gcc           hpux-kr-cc         irix-cc
irix-gcc           linux-aout         linux-elf          ncr-scde
nextstep           purify             sco5-cc            solaris-sparc-cc
solaris-sparc-gcc  solaris-sparc-sc4  solaris-usparc-sc4 solaris-x86-gcc
sunos-cc           sunos-gcc          unixware-2.0       unixware
```

For our system, we would enter the following:

```
# ./Configure linux-elf
CC            =gcc
CFLAG         =-DL_ENDIAN -DTERMIO -DBN_ASM -O3 -fomit-frame-pointer
EX_LIBS       =
BN_MULW       =asm/bn86-elf.o
DES_ENC       =asm/dx86-elf.o asm/yx86-elf.o
BF_ENC        =asm/bx86-elf.o
CAST_ENC      =asm/cx86-elf.o
RC4_ENC       =asm/rx86-elf.o
RC5_ENC       =asm/r586-elf.o
MD5_OBJ_ASM   =asm/mx86-elf.o
SHA1_OBJ_ASM  =asm/sx86-elf.o
RMD160_OBJ_ASM=asm/rm86-elf.o
THIRTY_TWO_BIT mode
DES_PTR used
DES_RISC1 used
DES_UNROLL used
BN_LLONG mode
RC4_INDEX mode
```

After the package has been configured, you can build it by typing `make`. If the build did not successfully complete, consult the documentation that comes with the distribution or the FAQ at *http://www.cryptsoft.com/ssleay/* for more information on what may have happened. If the build did complete, type `make install` to install the libraries on the system. Note that the makefile installs the package in */usr/local/ssl* by default. If you decide to install it in another directory, remember the directory when configuring Samba to use SSL.

Configuring SSLeay for Your System

The first thing you need to do is to set the `PATH` environment variable on your system to include the */bin* directory of the SSL distribution. This can be done with the following statement:

```
PATH=$PATH:/usr/local/ssl/bin
```

That's the easy part. Following that, you will need to create a random series of characters that will be used to prime SSLeay's random number generator. The random number generator will be used to create key pairs for both the clients and the server. You can create this random series by filling a text file of a long series of random characters. For example, you can use your favorite editor to create a text file with random characters, or use this command and enter arbitrary characters at the standard input:

```
cat >/tmp/private.txt
```

The Samba documentation recommends that you type characters for longer than a minute before interrupting the input stream by hitting Control-D. Try not to type

only the characters that are under your fingers on the keyboard; throw in some symbols and numbers as well. Once you've completed the random file, you can prime the random number generator with the following command:

```
# ssleay genrsa -rand /tmp/private.txt >/dev/null
2451 semi-random bytes loaded
Generating RSA private key, 512 bit long modulus
..+++++
.............................+++++
e is 65537 (0x10001)
```

You can safely ignore the output of this command. After it has completed, remove the series of characters used to create the key because this could be used to recreate any private keys that were generated from this random number generator:

```
rm -f /tmp/private.txt
```

The result of this command is the hidden file *.rnd*, which is stored in your home directory. SSLeay will use this file when creating key pairs in the future.

Configuring Samba to use SSL

At this point, you can compile Samba to use SSL. Recall that in Chapter 2, *Installing Samba on a Unix System*, we said you have to first run the configure script, which initializes the makefile, before you compile Samba. In order to use SSL with Samba, you will need to reconfigure the makefile:

```
./configure --with-ssl
```

After that, you can compile Samba with the following commands:

```
# make clean
# make all
```

If you encounter an error that says the *smbd* executable is missing the file *ssl.h*, you probably didn't install SSLeay in the default directory. Use the configure option `--with-sslinc` to point to the base directory of the SSL distribution—in this case, the directory that contains *include/ssl.h*.

On the other hand, if you have a clean compile, you're ready to move on to the next step: creating certificates.

Becoming a Certificate Authority

The SSL protocol requires the use of X.509 certificates in the protocol handshake to ensure that either one or both parties involved in the communication are indeed who they say they are. Certificates in real life, such as those use for SSL connections on public web sites, can cost in the arena of $300 a year. This is because the certificate must have a digital signature placed on it by a *certificate authority*. A

certificate authority is an entity that vouches for the authenticity of a digital certificate by signing it with its own private key. This way, anyone who wishes to check the authenticity of the certificate can simply use the certificate authority's public key to check the signature.

You are allowed to use a public certificate authority with SSLeay. However, you don't have to. Instead, SSLeay will allow you to declare yourself a trusted certificate authority—specifying which clients you choose to trust and which clients you do not. In order to do this, you will need to perform several tasks with the SSLeay distribution.

The first thing you need to do is specify a secure location where the certificates of the clients and potentially the server will be stored. We have chosen */etc/certificates* as our default. Execute the following commands as `root`:

```
# cd /etc
# mkdir certificates
# chmod 700 certificates
```

Note that we shut out all access to users other than `root` for this directory. This is very important.

Next, you need to set up the SSLeay scripts and configuration files to use the certificates stored in this directory. In order to do this, first modify the *CA.sh* script located at */usr/local/ssl/bin/CA.sh* to specify the location of the directory you just created. Find the line that contains the following entry:

```
CATOP=./demoCA
```

Then change it to:

```
CATOP=/etc/certificates
```

Next, you need to modify the */usr/local/ssl/lib/ssleay.cnf* file to specify the same directory. Find the entry:

```
[ CA_default ]
dir     = ./demoCA              # Where everything is kept
```

Then change it to:

```
[ CA_default ]
dir     =  /etc/certificates    # Where everything is kept
```

Next, run the certificate authority setup script, *CA.sh*, in order to create the certificates. Be sure to do this as the same user that you used to prime the random number generator above:

```
/usr/local/ssl/bin/CA.sh -newca
mkdir: cannot make directory '/etc/certificates': File exists
CA certificate filename (or enter to create)
```

Press the Enter key to create a certificate for the CA. You should then see:

```
Making CA certificate ...
Using configuration from /usr/local/ssl/lib/ssleay.cnf
Generating a 1024 bit RSA private key
..............................+++++
.....................+++++
writing new private key to /etc/certificates/private/cakey.pem
Enter PEM pass phrase:
```

Enter a new pass phrase for your certificate. You will need to enter it twice correctly before SSLeay will accept it:

```
Enter PEM pass phrase:
Verifying password - Enter PEM pass phrase:
```

Be sure to remember this pass phrase. You will need it to sign the client certificates in the future. Once SSLeay has accepted the pass phrase, it will continue on with a series of questions for each of the fields in the X509 certificate:

```
You are about to be asked to enter information that will be
incorporated into your certificate request.
What you are about to enter is what is called a Distinguished
Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
```

Fill out the remainder of the fields with information about your organization. For example, our certificate looks like this:

```
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:California
Locality Name (eg, city) []:Sebastopol
Organization Name (eg, company) []:O'Reilly
Organizational Unit Name (eg, section) []:Books
Common Name (eg, YOUR name) []:John Doe
Email Address []:doe@ora.com
```

After that, SSLeay will be configured as a certificate authority and can be used to sign certificates for client machines that will be connecting to the Samba server.

Creating Certificates for Clients

It's simple to create a certificate for a client machine. First, you need to generate a public/private key pair for each entity, create a certificate request file, and then use *SSLeay* to sign the file as a trusted authority.

For our example client `phoenix`, this boils down to three SSLeay commands. The first generates a key pair for the client and places it in the file *phoenix.key*. The

private key will be encrypted, in this case using triple DES. Enter a pass phrase when requested below—you'll need it for the next step:

```
# ssleay genrsa -des3 1024 >phoenix.key
1112 semi-random bytes loaded
Generating RSA private key, 1024 bit long modulus
.....................................+++++
.............+++++
e is 65537 (0x10001)
Enter PEM pass phrase:
Verifying password - Enter PEM pass phrase:
```

After that command has completed, type in the following command:

```
# ssleay req -new -key phoenix.key -out phoenix-csr
Enter PEM pass phrase:
```

Enter the pass phrase for the client certificate you just created (not the certificate authority). At this point, you will need to answer the questionnaire again, this time for the client machine. In addition, you must type in a challenge password and an optional company name—those do not matter here. When the command completes, you will have a certificate request in the file *phoenix-csr*.

Then, you must sign the certificate request as the trusted certificate authority. Type in the following command:

```
# ssleay ca -days 1000 -inflies phoenix-csr >phoenix.pem
```

This command will prompt you to enter the PEM pass phrase of the *certificate authority*. Be sure that you do not enter the PEM pass phrase of the client certificate that you just created. After entering the correct pass phrase, you should see the following:

```
Check that the request matches the signature
Signature ok
The Subjects Distinguished Name is as follows:
...
```

This will be followed by the information that you just entered for the client certificate. If there is an error in the fields, the program will notify you. On the other hand, if everything is fine, SSLeay will confirm that it should sign the certificate and commit it to the database. This adds a record of the certificate to the */etc/certificates/newcerts* directory.

The operative files at the end of this exercise are the *phoenix.key* and *phoenix.pem* files, which reside in the current directory. These files will be passed off to the client with whom the SSL-enabled Samba server will interact, and will be used by SSL Proxy.

Configuring the Samba Server

The next step is to modify the Samba configuration file to include the following setup options. These options assume that you created the certificates directory for the certificate authority at */etc/certificates*:

```
[global]
    ssl = yes
    ssl server cert = /etc/certificates/cacert.pem
    ssl server key = /etc/certificates/private/cakey.pem
    ssl CA certDir = /etc/certificates
```

At this point, you will need to kill the Samba daemons and restart them manually:

```
# nmbd -D
# smbd -D
Enter PEM pass phrase:
```

You will need to enter the PEM pass phrase of the certificate authority to start up the Samba daemons. Note that this may present a problem in terms of starting the program using ordinary means. However, you can get around this using advanced scripting languages, such as Expect or Python.

Testing with smbclient

A good way to test whether Samba is working properly is to use the *smbclient* program. On the Samba server, enter the following command, substituting the appropriate share and user for a connection:

```
# smbclient //hydra/data -U tom
```

You should see several debugging statements followed by a line indicating the negotiated cipher, such as:

```
SSL: negotiated cipher: DES-CBC3-SHA
```

After that, you can enter your password and connect to the share normally. If this works, you can be sure that Samba is correctly supporting SSL connections. Now, on to the client setup.

Setting Up SSL Proxy

The SSL Proxy program is available as a standalone binary or as source code. You can download it from *http://obdev.at/Products/sslproxy.html*.

Once it is downloaded, you can configure and compile it like Samba. We will configure it on a Windows NT system. However, setting it up for a Unix system involves a nearly identical series of steps. Be sure that you are the superuser (administrator) for the next series of steps.

If you downloaded the binary for Windows NT, you should have the following files in a directory:

- *cygwinb19.dll*
- *README.TXT*
- *sslproxy.exe*
- *dummyCert.pem*

The only one that you will be interested in is the SSL Proxy executable. Copy over the *phoenix.pem* and *phoenix.key* files that you generated earlier for the client to the same directory as the SSL proxy executable. Make sure that the directory is secure from the prying eyes of other users.

The next step is to ensure that the Windows NT machine can resolve the NetBIOS name of the Samba server. This means that you should either have a WINS server up and running (the Samba server can perform this task with the `wins support = yes` option) or have it listed in the appropriate *hosts* file of the system. See Chapter 7, *Printing and Name Resolution*, for more information on WINS server.*

Finally, start up SSL Proxy with the following command. Here, we assume that `hydra` is the name of the Samba server:

```
# C:\SSLProxy>sslproxy -l 139 -R hydra -r 139 -n -c phoenix.pem -k phoenix.key
```

This tells SSL Proxy to listen for connections to port 139 and relay those requests to port 139 on the NetBIOS machine `hydra`. It also instructs SSL Proxy to use the *phoenix.pem* and *phoenix.key* files to generate the certificate and keys necessary to initiate the SSL connection. SSL Proxy responds with:

```
Enter PEM pass phrase:
```

Enter the PEM pass phrase of the client keypair that you generated, *not* the certificate authority. You should then see the following output:

```
SSL: No verify locations, trying default
proxy ready, listening for connections
```

That should take care of the client. You can place this command in a startup sequence on either Unix or Windows NT if you want this functionality available at all times. Be sure to set any clients you have connecting to the NT server (including the NT server itself) to point to this server instead of the Samba server.

After you've completed setting this up, try to connect using clients that proxy through the NT server. You should find that it works almost transparently.

---

* If you are running SSL Proxy on a Unix server, you should ensure that the DNS name of the Samba server can be resolved.

SSL Configuration Options

Table A-1 summarizes the configuration options introduced in the previous section for using SSL. Note that all of these options are global in scope; in other words, they must appear in the `[global]` section of the configuration file.

Table A-1. SSL Configuration Options

Option	Parameters	Function	Default	Scope
`ssl`	boolean	Indicates whether SSL mode is enabled with Samba.	`no`	Global
`ssl hosts`	string (list of addresses)	Specifies a list of hosts that must always connect using SSL.	None	Global
`ssl hosts resign`	string (list of addresses)	Specifies a list of hosts that never connect using SS.	None	Global
`ssl CA certDir`	string (fully-qualified pathname)	Specifies the directory where the certificates are stored.	None	Global
`ssl CA certFile`	string (fully-qualified pathname)	Specifies a file that contains all of the certificates for Samba.	None	Global
`ssl server cert`	string (fully-qualified pathname)	Specifies the location of the server's certificate.	None	Global
`ssl server key`	string (fully-qualified pathname)	Specifies the location of the server's private key.	None	Global
`ssl client cert`	string (fully-qualified pathname)	Specifies the location of the client's certificate.	None	Global
`ssl client key`	string (fully-qualified pathname)	Specifies the location of the client's private key.	None	Global
`ssl require clientcert`	boolean	Indicates whether Samba should require each client to have a certificate.	`no`	Global
`ssl require servercert`	boolean	Indicates whether the server itself should have a certificate.	`no`	Global
`ssl ciphers`	String	Specifies the cipher suite to use during protocol negotiation.	None	Global
`ssl version`	`ssl2or3`, `ssl3`, or `tls1`	Specifies the version of SSL to use.	`ssl2or3`	Global

Table A-1. SSL Configuration Options (continued)

Option	Parameters	Function	Default	Scope
`ssl compatibility`	boolean	Indicates whether compatibility with other implementations of SSL should be activated.	`no`	Global

ssl

This global option configures Samba to use SSL for communication between itself and clients. The default value of this option is `no`. You can reset it as follows:

```
[global]
    ssl = yes
```

Note that in order to use this option, you must have a proxy for Windows 95/98 clients, such as in the model presented earlier in this chapter.

ssl hosts

This option specifies the hosts that will be forced into using SSL. The syntax for specifying hosts and addresses is the same as the `hosts allow` and the `hosts deny` configuration options. For example:

```
[global]
    ssl = yes
    ssl hosts = 192.168.220.
```

This example specifies that all hosts that fall into the 192.168.220 subnet must use SSL connections with the client. This type of structure is useful if you know that various connections will be made by a subnet that lies across an untrusted network, such as the Internet. If neither this option nor the `ssl hosts resign` option has been specified, and `ssl` is set to `yes`, Samba will allow only SSL connections from all clients.

ssl hosts resign

This option specifies the hosts that will *not* be forced into SSL mode. The syntax for specifying hosts and addresses is the same as the `hosts allow` and the `hosts deny` configuration options. For example:

```
[global]
    ssl = yes
    ssl hosts resign = 160.2.310. 160.2.320.
```

This example specifies that all hosts that fall into the 160.2.310 or 160.2.320 subnets will not use SSL connections with the client. If neither this option nor the `ssl hosts` option has been specified, and `ssl` is set to `yes`, Samba will allow only SSL connections from all clients.

ssl CA certDir

This option specifies the directory containing the certificate authority's certificates that Samba will use to authenticate clients. There must be one file in this directory for each certificate authority, named as specified earlier in this chapter. Any other files in this directory are ignored. For example:

```
[global]
    ssl = yes
    ssl hosts = 192.168.220.
    ssl CA certDir = /usr/local/samba/cert
```

There is no default for this option. You can alternatively use the option `ssl CA certFile` if you wish to place all the certificate authority information in the same file.

ssl CA certFile

This option specifies a file that contains the certificate authority's certificates that Samba will use to authenticate clients. This option differs from `ssl CA certDir` in that there is only one file used for all the certificate authorities. An example of its usage follows:

```
[global]
    ssl = yes
    ssl hosts = 192.168.220.
    ssl CA certFile = /usr/local/samba/cert/certFile
```

There is no default for this option. You can also use the option `ssl CA certDir` if you wish to have a separate file for each certificate authority that Samba trusts.

ssl server cert

This option specifies the location of the server's certificate. This option is mandatory; the server must have a certificate in order to use SSL. For example:

```
[global]
    ssl = yes
    ssl hosts = 192.168.220.
    ssl CA certFile = /usr/local/samba/cert/certFile
    ssl server cert = /usr/local/samba/private/server.pem
```

There is no default for this option. Note that the certificate may contain the private key for the server.

ssl server key

This option specifies the location of the server's private key. You should ensure that the location of the file cannot be accessed by anyone other than `root`. For example:

```
[global]
    ssl = yes
    ssl hosts = 192.168.220.
    ssl CA certFile = /usr/local/samba/cert/certFile
    ssl server key = /usr/local/samba/private/samba.pem
```

There is no default for this option. Note that the private key may be contained in the certificate for the server.

ssl client cert

This option specifies the location of the client's certificate. The certificate may be requested by the Samba server with the `ssl require clientcert` option; the certificate is also used by *smbclient*. For example:

```
[global]
    ssl = yes
    ssl hosts = 192.168.220.
    ssl CA certFile = /usr/local/samba/cert/certFile
    ssl server cert = /usr/local/ssl/private/server.pem
    ssl client cert= /usr/local/ssl/private/clientcert.pem
```

There is no default for this option.

ssl client key

This option specifies the location of the client's private key. You should ensure that the location of the file cannot be accessed by anyone other than `root`. For example:

```
[global]
    ssl = yes
    ssl hosts = 192.168.220.
    ssl CA certDir = /usr/local/samba/cert/
    ssl server key = /usr/local/ssl/private/samba.pem
    ssl client key = /usr/local/ssl/private/clients.pem
```

There is no default for this option. This option is only needed if the client has a certificate.

ssl require clientcert

This option specifies whether the client is required to have a certificate. The certificates listed with either the `ssl CA certDir` or the `ssl CA certFile` will be searched to confirm that the client has a valid certificate and is authorized to connect to the Samba server. The value of this option is a simple boolean. For example:

```
[global]
    ssl = yes
    ssl hosts = 192.168.220.
    ssl CA certFile = /usr/local/samba/cert/certFile
```

```
    ssl require clientcert = yes
```

We recommend that you require certificates from all clients that could be connecting to the Samba server. The default value for this option is `no`.

ssl require servercert

This option specifies whether the server is required to have a certificate. Again, this will be used by the *smbclient* program. The value of this option is a simple boolean. For example:

```
[global]
    ssl = yes
    ssl hosts = 192.168.220.
    ssl CA certFile = /usr/local/samba/cert/certFile
    ssl require clientcert = yes
    ssl require servercert = yes
```

Although we recommend that you require certificates from all clients that could be connecting to the Samba server, a server certificate is not required. It is, however, recommended. The default value for this option is `no`.

ssl ciphers

This option sets the ciphers on which SSL will decide during the negotiation phase of the SSL connection. Samba can use any of the following ciphers:

```
DEFAULT
DES-CFB-M1
NULL-MD5
RC4-MD5
EXP-RC4-MD5
RC2-CBC-MD5
EXP-RC2-CBC-MD5
IDEA-CBC-MD5
DES-CBC-MD5
DES-CBC-SHA
DES-CBC3-MD5
DES-CBC3-SHA
RC4-64-MD5
NULL
```

It is best not to set this option unless you are familiar with the SSL protocol and want to mandate a specific cipher suite.

ssl version

This global option specifies the version of SSL that Samba will use when handling encrypted connections. The default value is `ssl2or3`, which specifies that either version 2 or 3 of the SSL protocol can be used, depending on which version is negotiated in the handshake between the server and the client. However,

if you want Samba to use only a specific version of the protocol, you can specify the following:

```
[global]
    ssl version = ssl3
```

Again, it is best not to set this option unless you are familiar with the SSL protocol and want to mandate a specific version.

ssl compatibility

This global option specifies whether Samba should be configured to use other versions of SSL. However, because no other versions exist at this writing, the issue is moot and the variable should always be left at the default.