Stronghold version 1.3

User's Guide

©1996 Community Connexion, Inc.

Contents

Chapter 1: Installation 1

Expanding The Archive 1

Installing a New Stronghold Server 1

- Installing Stronghold from Scratch 2
- Upgrading an Apache Server 3

Generating a Key Pair 4

Upgrading an Existing Server 6

Chapter 2: Configuration 7

Editing the PATH Variable 7

Configuring a Unified or Dual Server 7

SSL Configuration Directives 9

- General Directives 9
- Client Authentication 10
- Cipher-Types 12

Configuring an Upgraded Apache Server 13

SSL Utilities 14

- change_pass 14
- decrypt_key 14
- gencert 15
- genkey 15
- genreq 15
- checkcert 15
- getca 16
- makeca 16
- renewal 16

Virtual Hosts 16

- Configuring VirtualHost 17
- Running Genkey 17
- Installing the Certificates 17

Apache Server Extension 18

Chapter 3: Running the Server 19

Starting the Server 19

Restarting the Server 19

Shutting Down the Server 20

Chapter 4: Certificates and Certificate Authorities 21

Using Certificate Authorities 21

VeriSign 22

Choosing a Certificate Authority 22

Installing a Certificate 23

Using a Private Certificate Authority 23

Setting Up the CA 24

Configuring SSLeay 25

Signing Certificates 25

Chapter 5: Troubleshooting 26

Chapter 1 Installation

This manual describes the process of installing Stronghold, including instructions on how to generate a key pair and certificate for SSL-secured transmissions. Once you've installed the server, you'll need to edit certain configuration files. The "Configuration" document describes these files and other topics, including how to use virtual hosts with this server.

This chapter describes how to install version 1.3 of Stronghold, including

- expanding the archive
- installing a new server
- generating a key pair
- upgrading an existing server

Once you've completed these steps, running the server is simple. If you have a problem with the server that is not resolved in this documentation set, the Stronghold FAQ at `http://www.us.apache-ssl.com/faq.phtml` offers additional information and troubleshooting tips.

Expanding The Archive

You download Stronghold as a compressed archive, which you must expand and extract. To expand the compressed file, place the file in a temporary directory, then run `gunzip`:

```
#  gunzip stronghold-version-
allplatforms.tar.gz
```

After `gunzip` expands the file, use `tar` to extract the archive:

```
#  tar -xf stronghold-version-
allplatforms.tar
```

`Tar` extracts the stronghold-version directory, which contains the server components.

Installing a New Stronghold Server

You can install an entire Stronghold server package from scratch, or turn your plain Apache server into an SSL-secured Stronghold server. If you already have a version of Stronghold installed, see "Upgrading an Existing Server" later in this guide.

Installing Stronghold from Scratch

Use the `INSTALL.sh` script to set up a new Stronghold server from scratch. This scripts asks you a few questions, then installs the server according to the parameters you specify.

Note: If you already have an installed Apache server, you can use the `APACHE.sh` script to upgrade it instead. See "Upgrading an Apache Server" later in this section.

---

1. Platform

   The program detects available operating systems that are compatible with Stronghold. Enter the name of the operating system the server will use.

2. SSLeay directory

   By default, the program creates a `/usr/local/ssl` directory and stores SSL security utilities there. Enter a different pathname if you want to store Stronghold elsewhere.

3. Apache directory (ServerRoot)

   The Apache directory is where Stronghold stores the Apache component and items related to its activities, including the CGI-bin, configuration files, and document tree. The default is `/usr/local/apache`; enter a different pathname if you want to store these files elsewhere.

4. Log directory

   The program creates separate logs in this directory for SSL and non-SSL transmissions.

5. Server hostname

   The hostname of the local machine is the default server hostname. Enter a different hostname if the server will use an alias. Stronghold also uses the server hostname as the filename for both your key and certificate files, such as `www.random.net.key` and `www.random.net.cert`, to distinguish between them and keys belonging to virtual hosts.

6. Server administrator's email address

   By default, the server will send notices and alerts to `webmaster@hostname`.

7. Normal server port number

   The default HTTP server port number is 80. This is the port on which the Apache component will send and receive non-SSL transmissions.

8. SSL server port number

   SSL server activity takes place on a separate port to isolate secured access or attempted access. The default SSL server port is 443.

9. Username for server

   The server runs as `nobody` by default, to eliminate security weaknesses through user access accounts.

10. Server group

The server runs as `nogroup` by default.

`INSTALL.sh` completes the installation and displays instructions for editing the PATH variable and other basic configuration files to include Stronghold. You can do this now or later; the program then continues the setup process by invoking `genkey`, which prompts you for information related to key pair generation.

See the next section, "Generating a Key Pair," for instructions on using `genkey`.

Upgrading an Apache Server

With the `APACHE.sh` script, you can upgrade an existing Apache server to implement Stronghold's SSL scheme. If you do this, you'll need to configure its new SSL component and restart the server. See the "Configuring an Upgraded Apache Server" section of Chapter 2, "Configuration."

Like `INSTALL.sh`, the `APACHE.sh` upgrade script asks you a series of questions, then invokes `genkey` to generate a new key pair and certificate request to get you started.

1. Platform

   The program detects available operating systems that are compatible with Stronghold. Enter the name of the operating system the server will use.

2. SSLeay directory

   By default, the program creates a `/usr/local/ssl` directory and stores SSL security utilities there. Enter a different pathname if you want to store Stronghold elsewhere.

3. Apache httpd directory

   Enter the directory where your Apache httpd binary is stored.

4. Server hostname

   The hostname of the local machine is the default server hostname. Enter a different hostname if the server will use an alias. Stronghold also uses the server hostname as the filename for both your key and certificate files, such as `www.random.net.key` and `www.random.net.cert`, to distinguish between them and keys belonging to virtual hosts.

5. SSL server port number

   SSL server activity takes place on a separate port to isolate secured access or attempted access. The default SSL server port is 443.

`APACHE.sh` completes the installation and displays instructions for editing the PATH variable and other basic configuration files to include Stronghold. You can do this now or later; the program then continues the setup process by invoking `genkey`, which prompts you for information related to key pair generation.

The next section describes how to use `genkey`.

Generating a Key Pair

Before you can use Stronghold for secure HTTP transactions, you must generate the encryption key pair using `genkey`. When `INSTALL.sh` or `APACHE.sh` invoke it, `genkey` generates a *public key* and a *private key*. Clients use your public key for encryption. The server passes the public key to the client, which uses it to encrypt the "session key" generated for that session. The session key is sent back to the server and all transmissions are encrypted using the session key. Because the session key was encrypted with the server's public key, only the server and the client know the session key.

Note: When you generate a new key pair, back up any old key pairs by copying them to another directory.

Along with the key pair, `genkey` generates a certificate signing request (CSR) to send to a Certificate Authority (CA), typically VeriSign. The CSR procedure described below corresponds to Step 2 of VeriSign's "Overview of the Process."

Your server's public key is embedded in the certificate you receive from the CA, and the certificate is signed with the CA's private key. The CA acts as a trusted third party who authenticates. When your server sends its signed certificate to a client, the client validates the signature with the CA's public key. Validation assures the client that there is no interposer, or "man-in-the-middle", violating the privacy and integrity of the session.

Successful decryption verifies that the certificate is valid, and the decrypted certificate reveals information about your server and its public key. If your CA is VeriSign, the standard CA, see `http://www.verisign.com/apachessl-us/index.shtml` before you proceed. You can also find more information about using CAs in *Certificate Authorities*.

Like `INSTALL.sh`, `genkey` asks you a series of questions and generates the key pair and certificate-related information based on your parameters:

1. New key generation or existing key conversion

   You can create a new key, test certificate, and CSR, or convert an existing Netscape Commerce certificate and key pair for use with Stronghold. When you enter your choice, `genkey` displays the names of the key file and certificate file, along with the directory in which they are stored.

2. Key size

   This is a crucial point, since the size (and complexity) of the key determines its security level. Smaller keys are easier to crack, while cracking a larger key requires a dedicated supercomputer.

   Note: Decryption times are geometrically proportional to key size. Doubling the key size, for example, multiplies the decryption time by eight for every SSL transmission. Since 512-bit keys are considered innocuous enough for export, choose a larger key size—ideally 1024-bit or as large as your server platform can handle.

Once you have chosen a key size, `genkey` creates some of the random data used to generate the key pair. Depending on the key size, this may require some time.

3. Random keystrokes

   The program then asks you to help complete the random data set by entering random keystrokes while it times the intervals between them.

4. Random file data

   Finally, `genkey` asks for the names of files whose random bits can be used as the final components of the random data set. This step is optional. If you prefer, you can skip it and use only the random data already generated.

5. Passphrase

   The passphrase is an identifying phrase that restricts key access to its owners. Choose your passphrase wisely and *do not lose it*. It's a good idea to write it down and store it in a safe deposit box or a safe.

6. Certificate Signing Request

   Enter Y to send a CSR.

Note: You must send a CSR and install a new certificate to use this key pair for SSL server transmissions. The test certificate generated by genkey is for testing purposes only, and does not authenticate your transmissions. Users will receive an error message when they encounter your test certificate.

---

7. CA

   Select the CA you intend to petition for a certificate.

8. Passphrase

   Enter your passphrase again, this time to verify that you are authorized to enter information related to the generation of your CSR.

9. Country Name

   Enter the two-letter code for the country in which your Stronghold server resides.

10. State Name

    Enter the full name of the state or province in which the server resides.

11. Locality Name

    Enter the name of the city, town, or county in which the server resides.

12. Organization Name

    Your organization name is required information.

13. Organization Unit Name

    This information is optional. To skip this field, enter a period (.).

14. Common Name

    This is typically the hostname of your server, such as `www.random.com`.

The program prints your certificate signing request. Verify the information it contains before you proceed.

15. Webmaster email address
16. Webmaster phone number
17. Certificate Authority

    Enter the email address of the CA to which you want to send the request if it differs from the default.

18. Passphrase

    `Genkey` prompts you again for your passphrase for authentication, then generates and installs a self-signed certificate for test purposes. You can use this certificate until the CA responds to your request, but it does not authenticate your server.

    When `genkey` has finished generating your key pair, be sure to create backups of your key files and store them in a secure directory.

With the new server installed, you are ready to configure it as described in Chapter 2, "Configuration." When you receive the new certificate from your CA, see the "Installing a Certificate" of Chapter 4, "Certificates and Certificate Authorities." For instructions on running the server, see Chapter 3, "Running the Server."

Upgrading an Existing Server

If you already have a version of Stronghold, you can quickly upgrade it using the `UPGRADE.sh` script. The script asks you a few questions, then upgrades your server according to the parameters you specify:

1. Old version number

   Select the version number of your existing Stronghold server.

2. Apache directory (ServerRoot)

   Indicate the directory where your existing Apache server component is stored. The default is `/usr/local/apache`; enter a different pathname if Apache is installed elsewhere.

3. SSLeay directory

   The default SSLeay directory is `/usr/local/ssl`. Enter a different pathname if you installed the SSL component elsewhere.

4. Platform

   Enter the platform on which your Stronghold server runs.

   The script upgrades the server, edits configuration and certificate files, and restarts the new server version.

Your existing configuration files remain intact. For instructions on running the upgraded server, see Chapter 4, "Running the Server."

Chapter 2 Configuration

Once you've installed Stronghold, you must configure the server to support your virtual hosts and provide SSL security that meets your organization's security policy. This document describes

- how to edit the PATH variable
- how to configure a unified server
- how to configure a dual server
- SSL configuration directives
- how to configure an upgraded Apache server
- SSL utilities
- VirtualHost configuration

Editing the PATH Variable

The PATH variable behaves differently depending on which type of UNIX shell you are using.

From the Bourne shell, Korn shell, or bash, edit the PATH variable in your `.profile` as follows, where *SSLTOP* is the pathname of your top-level SSL directory:

```
# SSLTOP=${SSLTOP}
# PATH=${SSLTOP}/bin:$PATH
# export PATH SSLTOP
```

If you are using the C-shell or tsh, edit the PATH variable in your `.cshrc` as follows:

```
# setenv SSLTOP ${SSLTOP}
# set path=(${SSLTOP}/bin $path)
```

PATH variable configuration works the same whether you choose the unified or dual server configuration, described in the next section.

Configuring a Unified or Dual Server

You can configure Stronghold to carry out both SSL and non-SSL transmissions as a single server, instead of separate SSL and non-SSL servers. The `httpd.conf` file supports separate VirtualHost configurations for separate ports. When you enter a set of VirtualHost directives, you can specify a port number, either the port for SSL or non-SSL transmissions. In this way, a single configuration file stores both SSL and non-SSL directives for a single server.

TO CONFIGURE A UNIFIED SERVER

1. Configure the `ServerRoot/conf/httpd.conf` file as described in the Apache Configuration Directives guide.
2. Add Listen directives for both the SSL port and the non-SSL port, so that the server listens on both:

```
Listen 80
Listen 443
```

3. For each virtual host, add directives for port 80, the default non-SSL port. For example:

```
<VirtualHost 204.17.33.11:80>
ServerAdmin webmaster@www.random.com
DocumentRoot /www/random
ServerName www.random.com
UserDir disabled
ErrorLog /www/log/random/error_log
TransferLog /www/log/random/access_log
</VirtualHost>
```

4. For each virtual host, add directives for port 443, the default SSL port:

```
<VirtualHost 204.17.33.11:443>
...
</VirtualHost>
```

   Use the directives described in "SSL Configuration Directives" later in this guide.

5. Add SSL directives for the main host as if it were a virtual host:

```
<VirtualHost 204.17.33.10:443>
...
</VirtualHost>
```

   Use the directives described in the next section, "SSL Configuration Directives."

You can also configure Stronghold as as dual server with separate SSL and non-SSL components. To do so, use two separate configuration files: `httpd.conf` for the normal server and `httpsd.conf` for the SSL server.

TO CONFIGURE A DUAL SERVER

1. Configure `ServerRoot/conf/httpd.conf` for normal transmissions, according to the Apache Configuration Directives manual.
2. Configure `ServerRoot/conf/httpsd.conf` for SSL-secured transmissions, using the directives described in the next section.

SSL Configuration Directives

The fields described here configure the SSL portion of Stronghold. For information on configuring the Apache server component, including VirtualHost directives for port 80, see the Apache Configuration manual.

You need to use these directives differently depending on whether you've installed a unified SSL/non-SSL server or separate SSL and non-SSL servers:

- To configure a unified SSL/non-SSL Stronghold server, add these directives to `httpd.conf` as described in the previous section.
- To configure SSL as a separate server, add these directives to the `httpsd.conf` file. If you've upgraded your plain Apache server to Stronghold, you must create an `httpsd.conf` file by copying the `httpd.conf` file to `httpsd.conf`, then add the directives listed here.

General Directives

These directives set the basic parameters used by the SSL server, including certificates, logs, and keys.

`SSLFlag`

Syntax: SSLFlag on

"SSLFlag on" is required to activate the SSL server functionality.

`SSLRoot`

Syntax: SSLRoot *path*

`SSLRoot` is the root directory for SSL files. SSL file directives (except logging directives) can be relative to `SSLRoot`.

Note: All other paths and filenames relating to SSL must be either absolute or relative to `SSLRoot`.

`SSLCACertificatePath`

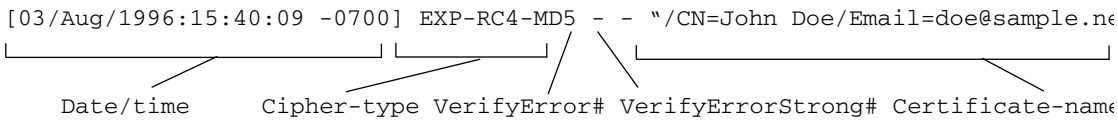
Syntax: SSLCACertificatePath *path*

This is the path for trusted CA root certificates. Store the CA root certificates for client certification here.

`SSLLogFile`

Syntax: SSLLogFile *filename*

Stronghold stores information about each SSL-enabled connection—such as cipher and client-authentication data—in the `SSLLogFile`. This sample log entry shows the log file fields:



`SSLErrorFile`

Syntax: SSLErrorFile *filename*

`SSLErrorFile` is where error messages and SSL diagnostics are stored. If this directive is unspecified, Stronghold stores this data in the `SSLLogFile` instead.

`SSLSessionServer`

Syntax: SSLSessionServer *filename*

In order for Stronghold to track client sessions and session keys, you must set the `SSLSessionServer` directive. If you do not specify a UNIX socket file, the default will be used.

`SSLSessionTimeout`

Syntax: SSLSessionTimeout *seconds*

Specify the SSL session timeout in seconds. Any session that is inactive for the period you specify will be closed.

Client Authentication

Stronghold uses these parameters to authenticate clients using client certificates.

`SSLVerifyClient`

Syntax: SSLVerifyClient [*0* |*1* |*2*]

This sets the X.509 Client Authentication option:

```
0 = No
1 = X.509 certificate optional
2 = X.509 certificate required
```

`SSLFakeBasicAuth`

Syntax: SSLFakeBasicAuth

Include this directive to enable access-controls based on client authentication, then enter access-control directives for each directory that requires authentication. For example:

```
<Location path>
AuthUserFile UserFilePath
AuthType Basic
AuthName RealmName
require valid-user
</Location>
```

AuthUserFile denotes the file that contains the list of accepted certificate names and corresponding, encrypted passwords, such as

```
/CN=Sameer Parekh/Email=sameer@c2.net:ieEc4N05EI
```

The encrypted password is the client passphrase for the key pair associated with the client certificate.

AuthType should always be set to Basic, because browsers currently do not support other values. RealmName is an arbitrary label for this authentication realm; use it to distinguish between different authentication schemes for different virtual hosts or directories.

The require field can take the following values:

- valid-user, to require authentication for any user who connects to this authentication realm
- user *username*, to require authentication only for specific users
- group *groupname*, to require authentication only for specific user groups

Verisign's Digital ID Center at `http://digitalid.verisign.com/brw_pick.htm` offers client certificate services.

Note: If this option is set, `SSLVerifyClient` *must* be set to 2 in order to avoid a security weakness. If `SSLVerifyClient` is set to 1, client authentication will only be implemented if the client *offers* a certificate.

`SSLVerifyDepth`

Syntax: SSLVerifyDepth *n*

Certificate Authorities are certified by other CAs, forming a chain of certificates that the server can follow for progressively decisive verification. `SSLVerifyDepth` sets the number of CAs the server will follow in this chain when verifying client certificates.

`SSLCACertificateFile`

Syntax: SSLCACertificateFile *filename*

This is the trusted CA root certificate file. Unlike `SSLCACertificatePath`, it specifies a certificate file rather than a certificate directory.

`SSLCertificateKeyFile`

Syntax: SSLCertificateKeyFile *filename*

This is the key pair file. Make sure it is stored securely.

Cipher-Types

Encryption directives set the acceptable encryption parameters for communication with your server. For these directives, use this list of cipher-types:

Cipher-Type	Description
`NULL-MD5`	No encryption, MD5 hash
`RC4-MD5`	Export-controlled RC4-compatible cipher, MD5 hash
`EXP-RC4-MD5`	Weakened for export, RC4-compatible cipher, MD5 hash
`RC2-CBC-MD5`	Export-controlled RC2-compatible cipher in CBC mode, MD5 hash
`EXP-RC2-CBC-MD5`	Weakened for export RC2-compatible cipher in CBC mode, MD5 hash
`IDEA-CBC-MD5`	IDEA cipher in CBC mode, MD5 hash
`DES-CBC-MD5`	DES in CBC, MD5 hash
`DES-CBC-SHA`	DES in CBC, SHA hash
`DES-CBC3-MD5`	3DES in CBC, MD5 hash
`DES-CBC3-SHA`	3DES in CDC, SHA hash
`DES-CFB-M1`	DES in CFB, M1 hash

`SSLRequiredCiphers`

Syntax: SSLRequiredCiphers *cipher1*:*cipher2*:*cipher3*....

This is a list of permitted cipher-types for initial SSL transmissions, in order of preference. This list applies only when the accessed directory does not have its own `SSLRequireCipher` or `SSLBanCipher` directives. If the client does not support any of the specified cipher-types, the server returns an error message.

`SSLRequireCipher`

Syntax: <Directory *path*>
SSLRequireCipher *cipher*
</Directory>

`SSLRequireCipher` specifies a cipher-type that is required for access to the specified directory. Each `SSLRequireCipher` directive takes only one argument; enter multiple `SSLRequireCipher` directives on separate lines for multiple required cipher-types. If the client does not support any of the required cipher-types, the server returns an error message.

`SSLBanCipher`

Syntax: <Directory *path*>
SSLBanCipher *cipher*
</Directory>

Some cipher-types may be not meet the security requirements for certain server directories. This directive specifies ciphers that are unacceptable to the directory specified by *path*. Each `SSLBanCipher` directive takes only one argument; enter multiple `SSLBanCipher` directives on separate lines for multiple cipher-types that are not acceptable.

Configuring an Upgraded Apache Server

If you've upgraded your existing Apache server to Stronghold using `APACHE.sh`, you'll need to set up a separate configuration file for the new SSL component. An ordinary Apache server that has been upgraded to Stronghold must run as a dual server; it cannot run as a unified server.

To configure an upgraded Apache server

1. Copy the `httpd.conf` file to `httpsd.conf`.
2. Edit the following directives in the `httpsd.conf` file:

```
Port 443
ScoreBoardFile logs/ssl/apache_runtime_status
PidFile logs/httpsd.pid
TransferLog logs/ssl/access_log
ErrorLog logs/ssl/error_log
SSLLogFile logs/ssl/cipher_log
SSLErrorFile logs/ssl/ssl_error_log
```

3. Add the following directives to the file:

```
SSLFlag on
SSLRoot path
SSLCertificateKeyFile private/filename
SSLCertificateFile certs/filename
SSLVerifyDepth n
SSLVerifyClient [0 |1 |2 ]
```

4. Add the `start` script to the `conf/` directory:

```
#!/bin/sh
SSLTOP=path
export SSLTOP
ServerRoot/httpsd -d ServerRoot -f conf/
httpd.conf
ServerRoot -d ServerRoot -f conf/httpsd.conf
```

5. Add the `stop` script to the `conf/` directory:

```
#!/bin/sh
kill 'cat ServerRoot/logs/httpd.pid
ServerRoot/logs/httpsd.pid'
```

6. Add the `reload` script to the `conf/` directory:

```
#!/bin/sh
kill -HUP 'cat ServerRoot/logs/httpd.pid
ServerRoot/logs/httpsd
```

7. Run the `stop` script to stop the old Apache server.
8. Run the `start` script to start the new SSL-enabled Stronghold server.

SSL Utilities

Stronghold provides server maintenance utilities for its SSL component. The utilities listed here are stored in `${SSLTOP}/bin`, and they control keys, certificates, and other SSL objects. Since you can use them with virtual hosts, use *hostname* as the command line argument for all utilities. The argument distinguishes the different hosts and their separate key pairs and certificate files.

change_pass

Sets the passphrase used to encrypt the private key for a host:

```
#  change_pass hostname
```

Use this utility periodically to keep your passphrase dynamic and prevent intruders from guessing it.

decrypt_key

Decrypts the private key for a host:

```
#  decrypt_key hostname
```

With `decrypt_key`, you can store your unencrypted private key on disk so that the server can access it when restarting unattended. If you use `decrypt_key`, be sure you store your unencrypted key securely to avoid a major security weakness. An unsecured and unencrypted private key exposes your server to attack.

gencert

Generates a test certificate for a host:

```
# gencert hostname
```

The test certificate does not authenticate your server. It generates an error message when users encounter it, and it exposes your server to possible man-in-the-middle attacks. Use the certificate generated by `gencert` for testing purposes *only*. It expires in 30 days.

genkey

Generates a key pair, then calls `genreq` and `gencert` to generate a certificate signing request and a temporary test certificate for a host:

```
# genkey hostname
```

`Genkey` creates a new, unique key pair for a new certificate. To generate a new certificate based on an existing key pair, use `gencert` alone.

genreq

Generates a certificate signing request based on the existing key pair for a host:

```
# genreq hostname
```

To generate a new key pair for the new certificate, plus a test certificate, use `genkey` instead.

Note: When you use `genreq`, be sure you specify the correct key pair file, so that your current public key is embedded in the certificate you receive from a CA. If you specify an incorrect key pair file, the new certificate will not work properly.

---

checkcert

Checks a certificate against the current key pair for a host:

```
# checkcert hostname
```

Since the certificate passes a public key to the client for use during encrypted sessions, the current certificate for any host or virtual host must match the current key pair. If they do not match, either

- locate the correct key pair, save it as `${SSLTOP}/private/host-name.key`, and run `genreq` to send a new CSR based on this key pair,
- run `genreq` if the correct key pair is already saved with the correct filename, or
- run genkey to generate a new key pair and matching CSR.

getca

Installs a certificate. When you receive your signed certificate from a CA, save it in a temporary file, such as `tmp/cert`. Run `getca` and specify both the name of the server that owns the certificate and the name of the temporary certificate file:

```
# getca hostname < tmp/cert
```

This saves the certificate to the file `${SSLTOP}/certs/servername.cert`. After you run `getca`, be sure to remove the temporary certificate file as a security precaution.

makeca

Sets up a private certificate authority on the Stronghold server platform by generating a key pair and self-signed certificate for a host:

```
#  makeca hostname
```

Use the key pair to sign certificates in response to CSRs.

Before you run `makeca`, make sure that `${SSLTOP}` is set to the correct path for your SSL directory. After you run `makeca`, edit the `${SSLTOP}/lib/ssleay.conf` file to configure the certificate authority according to your organization's parameters. For more detailed instructions, see the "Using a Private Certificate Authority" section of Chapter 4, "Certificates and Certificate Authorities."

renewal

Submits a certificate renewal request to your CA for a host:

```
#  renewal hostname
```

Stronghold notifies you when a certificate is about to expire, and automatically prompts you to run `renewal`. Run `renewal` promptly; an expired certificate prevents users from connecting to secured sites on your server.

Virtual Hosts

Because Netscape Navigator version 2.0 and later check the URL against the hostname on the server certificate, each virtual host requires a separate certificate. To run Stronghold with virtual hosts and multiple certificates, you need to configure VirtualHost, run `genkey`, and install the certificates.

Configuring VirtualHost

The example below illustrates a VirtualHost configuration for port 443, the default SSL port:

```
<VirtualHost 140.174.185.37:443>
ServerAdmin admin@alpha.c2.org
DocumentRoot /ul/ghio/alpha_html
ServerName alpha.c2.org
UserDir disabled
ErrorLog /var/log/www/alpha/eror_log
TransferLog /var/log/www/alpha/access_log
SSLRoot /usr/local/ssl
SSLLogFile /var/log/www/alpha/ssl_log
SSLErrorFile /var/log/www/alpha/ssl_error_log
SSLCertificateKeyFile private/
alpha.c2.org.key
SSLCertificateFile certs/alpha.c2.org.cert
SSLCACertificateFile CA/rootcerts.pemn
SSLCACertificatePath certs
SSLVerifyClient 0
SSLVerifyDepth 10
</VirtualHost>
```

Running Genkey

Run the `genkey` program just as you did for the original host, specifying a virtual host instead:

```
#  genkey alpha.c2.org
```

Along with the key pair, `genkey` will generate a certificate signing request (CSR) to send to a Certificate Authority (CA), typically VeriSign. For more detailed information about `genkey`, see "Server Installation."

Installing the Certificates

Run `getca` for each certificate and each virtual host, as follows:

1. Save the certificate on disk, in a temporary file such as `/tmp/cert`.
2. Run `getca`:

```
#  getca hostname < /tmp/cert
#  rm /tmp/cert
```

The certificate includes your server's public key and other identifying information about the server. It is also signed with the CA's private key. Since the client can obtain the matching public key directly from the CA, successfully decrypting the certificate shows the client that the certificate is valid and authorized by a trusted third party—the CA. It can therefore trust that the information in your certificate is correct and the public key belongs to your server. In this way, certificates prevent malicious parties from intercepting data packets and substituting their own encryption keys, which would allow them to decrypt all of your transmissions.

Apache Server Extension

As you optimize your server platform, you may want to install modules or patches to improve or extend the server's capabilities. To do so, compile them as follows:

1. Switch to the *ServerRoot*`/src` directory:

```
#  cd /usr/local/apache/src
```

   Note: This is the directory created during server installation.

2. Edit the configuration file to reflect your platform.
3. Run `configure`:

```
#  ./Configure
```

4. Build HTTPSD:

```
#  make
```

5. Stop the server:

```
#  ../stop
```

6. Install the new binaries:

```
#  mv ../httpsd ../httpsd-bak
#  cp httpsd ..
```

7. Restart the server:

```
#  ../start
```

Chapter 3 Running the Server

This brief reference describes

- how to start the server
- how to restart the server
- how to shut down the server

Starting the Server

Your ServerRoot directory contains the `start` script, which sets the appropriate SSL configuration options and runs the server. Always use `start`, and read the script carefully before you decide to build your own.

For secure transmissions, you must direct clients to the SSL part of Stronghold by using a URL that specifies the HTTPS protocol instead of HTTP, such as `https://www.random.com/private.html`.

Normally, the server will not start until you enter your pass phrase. In order to restart the server automatically—when the machine reboots unattended, for example—you need to store your *unencrypted* private key on disk. By default, Stronghold stores the key encrypted for higher security. In order to remove the encryption, use `decrypt_key`:

```
#  decrypt_key hostname
```

If you want to change the passphrase for your encrypted key, use `change_pass`. This script decrypts your key and then re-encrypts it with a new passphrase:

```
# change_pass hostname
```

Note: The `change_pass` script re-encrypts the key. If you were storing the key unencrypted, run `decrypt_key` again after you change the pass-phrase.

Restarting the Server

The `reload` script lets you restart the Stronghold server in one step. You can also restart the server by running the `stop` and `start` scripts in sequence.

As with the `start` script, `reload` does not restart the server until you enter your pass phrase. In order to run `reload` unattended, you need to store your unencrypted private key on disk. To do so, use `decrypt_key`:

```
#  decrypt_key hostname
```

If you want to change the pass phrase for your encrypted key, use `change_pass` to decrypt it and then re-encrypt with a new passphrase:

```
# change_pass hostname
```

Shutting Down the Server

To shut down the server, simply run the `stop` script. Always use `stop`, and read the script carefully before you create your own, specialized shutdown script.

Chapter 4 Certificates and Certificate Authorities

SSL requires the use of a certificate authority (CA). The CA prevents "man-in-the-middle" attacks on your Web server. You need to be certain that no third party is intercepting communications and violating the integrity or privacy of your transmissions. Users who access your site also need to be certain the no such interception can compromise their security.

This document discusses

- using certificate authorities
- Verisign, the default certificate authority
- choosing a certificate authority
- installing a certificate
- using a private certificate authority

Using Certificate Authorities

In order to encrypt transmissions to your server, the client requires the server's public key. The integrity (not the privacy) of the key is vital to the security of the subsequent SSL session. If a third party replaced it with his own public key, that "man-in-the-middle" could view all traffic, or even modify the data in transit. Neither the client nor the server would detect the intrusion.

To prevent this, the server sends the public key in a certificate signed by a certificate authority. The client checks that digital signature. If the signature is valid, the client knows that the CA has certified that this is this server's authentic certificate, not a certificate forged by a "man-in-the-middle."

The CA must be a *trusted* third party in order to provide meaningful authentication.

If you are using Stronghold for Internet-wide transactions, choose a CA trusted by as many people as possible. It is important that users feel safe connecting to a Web site that claims to be secure. It is important that they trust that the data they are transmitting and receiving has not been modified or viewed by unauthorized third parties.

VeriSign

Currently, there are only a few Internet-wide certificate authorities. The most popular of these is VeriSign. VeriSign realizes the importance of trust in the certification industry and has taken many steps to become the trusted name on the Internet. Being the first in the field, they have made mistakes and improved with experience, though the rapid innovation of network burglars makes future missteps unavoidable for any CA.

Prior to version 2.0, Netscape Navigator only trusted VeriSign. Other CAs were acknowledged, but VeriSign was the only one used. With the release of Navigator 2.0, clients were given the option of trusting certificate authorities other than VeriSign. With Navigator 3.0 (still in beta as of this writing, and subject to change), more CAs are built into the client as default-trusted CAs, and the ability to trust additional CAs remains.

If you install Stronghold for access by a limited number of people, such as within a corporate firewall, the issues become much more complex. Under no circumstances, however, should the server be installed with a "self-signed certificate." The self-signed certificate installed with Stronghold for testing purposes only. A server running with a self-signed certificate provides no protection against the man-in-the-middle attack.

Choosing a Certificate Authority

Depending on your security needs, you can choose to use an external CA, such as VeriSign, or an internal CA controlled by your own organization. Each has benefits and drawbacks.

Using an external CA is the simplest option. The Stronghold installation script automatically generates a certificate signing request for you to fill out, sends it for you, and you receive a signed certificate in a few days. No expensive training or startup fee is required. Be aware, however, that different certificate authorities have different policies. Investigate CAs before you choose one that is appropriate for your needs.

The largest drawback of this approach is that is sacrifices local control and confidentiality in favor of usability. By involving an external agency in the installation of your SSL-secured Web server, you may be subjecting sensitive data to external eyes or inadequate security measures. In some cases, this violates a site's security policy. By keeping a private CA, there is little risk of such a violation. A private CA can also be more responsive to the needs of people requesting certificates within the organization. Furthermore, since the CA is local, the organization has absolute control over the security policy. The security surrounding the CA can be as strong or as weak as required.

The drawbacks of the local CA lie in the expense of setting one up. Setting up a high-security CA requires dedicated, RF-shielded hardware, for example. Highly trusted personnel must be trained to operate the CA. Unless the organization is very large, setting up an internal CA may cost more than using an external CA—even though it may appear cheaper at first.

Another option combines the local control of the internal CA with the outsourcing benefit of the external CA. VeriSign offers "private label services" which allow people to build internal CAs while gaining leverage from VeriSign's existing investment in hardware, people, and knowledge. Contact VeriSign for more information.

Installing a Certificate

You should receive your signed certificate within a few days after you submit the CSR using `genkey` or `genreq`. Use `getca` to install it:

1. Save the certificate on disk, in a temporary file such as `/tmp/cert`.
2. Run `getca`, as follows:

```
# getca hostname < /tmp/cert
# rm /tmp/cert
```

The certificate includes your server's public key and other identifying information about the server. It is also signed with the CA's private key. Since the client can obtain the matching public key directly from the CA, successfully decrypting the certificate shows the client that the certificate is valid and authorized by a trusted third party—the CA. It can therefore trust that the information in your certificate is correct and the public key belongs to your server. In this way, certificates prevent malicious parties from intercepting data packets and substituting their own encryption keys, which would allow them to decrypt all of your transmissions.

Using a Private Certificate Authority

You can create a private certificate authority for your Intranet to certify internal web servers. Since these servers are typically accessed only by internal clients, a private CA can provide a sufficient level of trust within certain security policies.

It is your responsibility to follow an adequate security policy. Choose your CA platform carefully, and maintain it under the strongest security:

- Consult a security expert before building your own CA.
- Do not connect the CA to a network.
- Restrict login access to those who absolutely require it.
- Restrict physical access to the CA by keeping it in a tightly locked room.
- Equip the CA with an RF-shield to protect it from radio-frequency surveillance.
- Instruct your users to generate key pairs of adequate sizes, preferably 1024-bit.

Stronghold provides tools for setting up a basic certificate authority. To do this, you must

- Create the CA
- Configure SSLeay
- Distribute the certificate
- Set up a CSR service

Keep in mind that you can also use XCert Sentry to set up a more complex CA, which includes an Apache module and LDAP support, among other things.

Setting Up the CA

The `makeca` script sets up the CA by generating a key pair and self-signed certificate for the CA. Be sure `${SSLTOP}` is set appropriately.

Once the CA is built using the `makeca` script, you should edit the `${SSLTOP}/lib/ssleay.conf` file to fit the policy for the Certificate Authority that you've built. Pay special attention to the policy section. This is the section you need to edit in order to limit the fields for requests which your CA will sign. (In most cases, your CA will only sign keys within in the same organization as the CA. In this case you should set the policy to `policy_match` which requires that the countryName, stateOrProvinceName, and organizationName of the certificate request match the respective information in the Certificate Authority's information.)

Now that the Certificate Authority has been built and the policy configured, you need to publicize within your organization the CA's certificate, so that you user's client software accepts server's certificates signed with your private CA. The file `${SSLTOP}/CA/cacert.pem` is where the `makeca` script stored your CA's certificate. Netscape Navigator 2.0 uses a MIME type of `application/x-x509-ca-cert` to denote a Certificate Authority certificate. See the Netscape site for how Navigator deals with certificates.

Make sure your users install your new certificate in Netscape Navigator and their other client programs.

The CA certificate must be in DER format when downloaded to Navigator with the `application/x-x509-ca-cert` MIME type. In order to convert the PEM certificate format to DER format, you just use the `x509` program:

```
# x509 -outform DER \
< /usr/local/ssl/CA/cacert.pem > /usr/local/
apache/htdocs/myca.cacert
```

Now you need to setup a mechanism by which people will submit Certificate Signing Requests (CSRs) to your Certificate Authority. You perhaps should setup an email address to which people can mail their CSRs. You also need to define your organization's policy regarding the level of documentation before your CA will issue a signed certificate for a given CSR. You need to establish a policy such that you can be sure that the information in the certificate is the correct information for the person

or group within your organization submitting the certificate request. When someone in your organization then installs an SSL application (such as Stronghold or Netscape Commerce), they can send the CSR to your private CA address.

When you receive a CSR, make sure you first verify` the origin of the CSR according to your policies as you've defined them above. Then, after you've determined that you desire to issue a signed certificate, pass the CSR through the `ca` program. This program will print out information regarding the CSR. Verify that the information is compliant with your organization's policy again, and then answer "y" to both questions to issue a certificate. Send the signed certificate back to the requestor.

Configuring SSLeay

The default SSLeay settings are recommended. However, in order to use the CA according to your organization's security policy, you may need to edit the `${SSLTOP}/lib/ssleay.conf` file.

Signing Certificates

Once you have set up the private certificate authority and publicized it, establish a procedure for collecting and processing CSRs. Depending on the number of CSRs you need to accommodate, you can construct an automatic mechanism for signing certificates or sign them one by one through email or file transfer.

TO SIGN A CERTIFICATE

1. Save the CSR to a temporary file such as `/tmp/csr`.
2. Run the following:

```
#  ca -config ${SSLTOP}/lib/ssleay.conf -in /
tmp/csr
```

3. Return the signed certificate to the applicant.

Chapter 5 Troubleshooting

This document provides a list of common errors and their meanings. If you are having a problem that is not covered in this document, contact Stronghold technical support at `Stronghold-support@c2.net`.

`Error setting default verify locations`

`SSLCACertificatePath` and/or `SSLCACertificateFile` are set incorrectly in the configuration file. If `SSLCACertificateFile` is set, the file need to include PEM-encoded certificate filenames. If `SSLCACertficate-Path` is set, the directory it specifies needs to include PEM certificate files, with appropriate hash symlinks.

`Can't open certificate file "`*`X`*`."`

The certificate file specified with `SSLCertificateFile` is unreadable.

`Error reading server certificate file "`*`X`*`."`

The certificate file specified with `SSLCertificateFile` is corrupt. Check to make sure it is a PEM-encoded certificate file. Check to make sure the file is not empty.

`Can't open key file "`*`X`*`."`

The key file specified with `SSLCertificateKeyFile` is unreadable.

`Error reading private key file "`*`X`*`."`

The key file is corrupt, or the password you are using to decrypt the file is incorrect.

`No SSLRoot set for server "`*`X`*`."`

The SSLRoot directive is missing from the SSL configuration file.

`No SSL Certificate set for server `*`X`*`.`

The `SSLCertificateFile` directive is missing from the SSL configuration file.

`Bad value for SSLVerifyClient (`*`X`*`).`

`SSLVerifyClient` is less than 0 or greater than 2. Set its value to 0, 1, or 2.

`Required SSLLogFile missing.`

The `SSLLogFile` directive is missing from the SSL configuration file.

`bad base64 decode`

The PEM-encoded file's BASE64 coding is corrupt.

`bad decrypt`

The passphrase for the encrypted PEM file is incorrect.

`bad end line`

The PEM file has no end line. Insert the endline `--END X509 CERTIFICATE--`.

`no start line`

The PEM file has no start line (maybe the file is empty)

`stderr shows i=-1 enc_bits=5`

The key file and certificate file do not match. Use the `checkcert` utility to make sure that the filenames for `&{SSLTOP}/certs/`*`servername`*`.cert` and `&{SSLTOP}/private/`*`servername`*`.key` match:

```
#  checkcert servername
```