# Global Technical Committee

# Stunnel Implementation Guide for FIX Applications

**July 25, 2017**

**Revision 1.0**

**Proposal Status: Final**

**For Global Technical Committee Governance Internal Use Only**

| | | | |
|---|---|---|---|
| Submission Date | October 28, 2016 | Control Number | |
| Submission Status | Final | Ratified Date | |
| Primary Contact Person | Charles Kilkenny | Release Identifier | |

## DISCLAIMER

# Table of Contents

## Document History

| Revision | Date | Author | Revision Comments |
|---|---|---|---|
| 0.1 | 4/14/2017 | Don Mendelson<br>Silver Flash LLC | DRAFT revision 1 |
| 0.2 | 4/21/2017 | Don Mendelson<br>Silver Flash LLC | DRAFT revision 2 |
| 0.3 | 6/02/2017 | Don Mendelson<br>Silver Flash LLC<br>Vladimir Coxall<br>Itiviti USA | DRAFT revision 3 |
| 0.4 | 7/06/2017 | Don Mendelson<br>Silver Flash LLC | DRAFT revision 4 |
| 0.5 | 7/17/2017 | Don Mendelson<br>Silver Flash LLC | DRAFT revision 5 |
| 0.6 | 7/25/2017 | Don Mendelson<br>Silver Flash LLC | DRAFT revision 6 |
| 1.0 | 1/24/2018 | Alex Pollard<br>GTC PM | Updated to "Final" revision |

# 1 Introduction

## 1.1 Overview

This guide is a supplement to the FIXS Technical Standard for usage of Stunnel. Stunnel is a proxy for Transport Layer Security (TLS) communications. This guide is intended to cover use cases where an application is FIX-protocol aware but either does not have TLS capabilities or for which it is desirable to offload TLS operations to a proxy. One motivation for using a proxy is to terminate external communications in a DMZ subnetwork while protecting applications such as FIX engines within an internal local area network. The communications between the proxy and the FIX engine can then flow over an ordinary TCP transport.

Note: many of the names associated with secure sockets still contain "SSL", but the SSL protocol has been superseded by TLS.

## 1.2 FIX and TLS Roles

In FIX4 and FIXT protocols, each peer plays one of two roles. In the initiator role, a peer initiates the TCP transport connection to a remote host, and once the TCP handshake is complete, it sends a FIX Logon message to the remote side. In the acceptor role, a FIX engine listens on a port for incoming TCP connections. After accepting a new connection, it waits for a FIX Logon and when it is received, it authenticates the user and continues the FIX session.

To add TLS to a FIX session, both peers must implement the protocol in an interoperable way. With or without TLS, the process begins with a TCP handshake between initiator and acceptor. Thereafter, a TLS handshake begins. There are two roles in TLS, client and server. The initiator of the FIX-over-TCP connection acts as a TLS client, and the FIX acceptor acts as a TLS server. TLS always authenticates the server using cryptographic methods. Mutual authentication is optional in TLS. If TLS client authentication is performed, then authentication at the FIX level is redundant since identity is already proven. On the other hand, if one-way authentication is performed in TLS, then the FIX initiator is obligated to send credentials in the FIX Logon message to prove identity at the application layer.

### 1.2.1 Deployment

Stunnel is distributed as an executable for Windows, Android or Linux. Downloads are available at https://www.stunnel.org/downloads.html. The software-only implementation depends on OpenSSL, available in GitHub as source at https://github.com/openssl/openssl. Hardware implementations of encryption engines are also available from product vendors.

A local FIX engine may be configured for one or more FIX sessions. By using Stunnel, the FIX application needs no knowledge of TLS or cryptographic keys, yet its communications to the remote host is secure. All the TLS configuration is in Stunnel.

#### 1.2.1.1 Client Deployment

In the initiator role, a FIX application opens a TCP connection to Stunnel. This triggers Stunnel to open a client-mode TLS session with the remote host. When the TLS handshake is complete, Stunnel forwards any messages received on the local TCP connection to the remote host.

The only possible security configuration in the FIX application consists of FIX credentials to be sent in the Logon message if mutual authentication in TLS is not configured.

The diagrams below show typical deployments of a FIX application connecting to a remote server via Stunnel as a proxy.

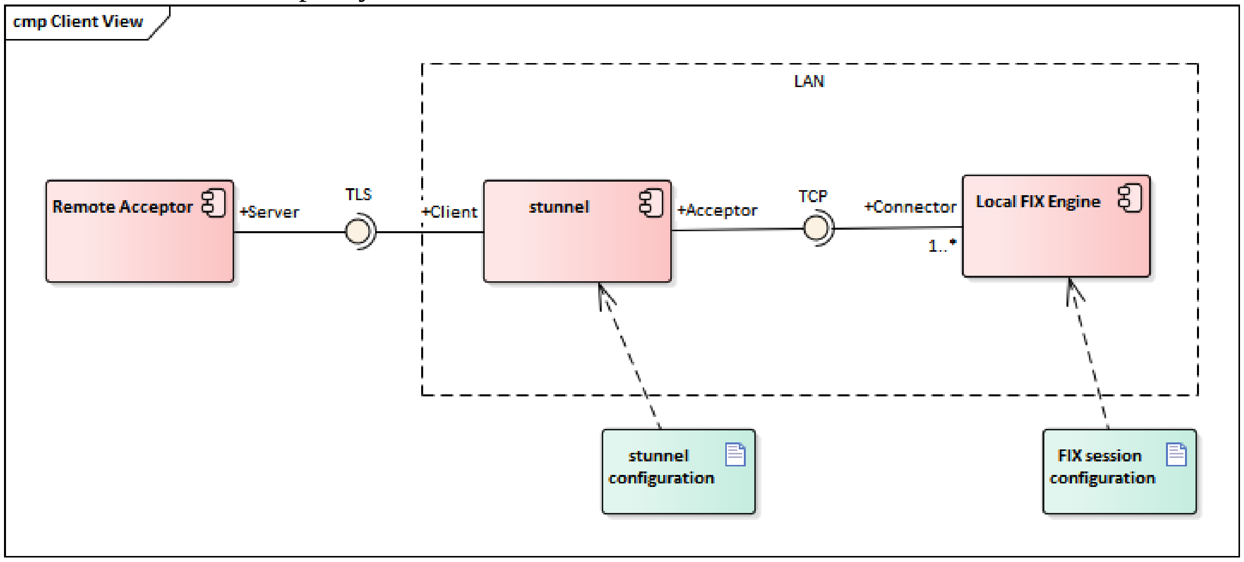

*Figure 1Typical client deployment*

### 1.1.1.1. Server Deployment

When a FIX application acts in the acceptor role, Stunnel acts as a TLS server. When a remote client and opens a TLS session to Stunnel, Stunnel connects to the local FIX engine’s TCP listen port. All messages received from the remote client are forwarded to the local FIX engine, and vice versa.
A typical server configuration is shown below.

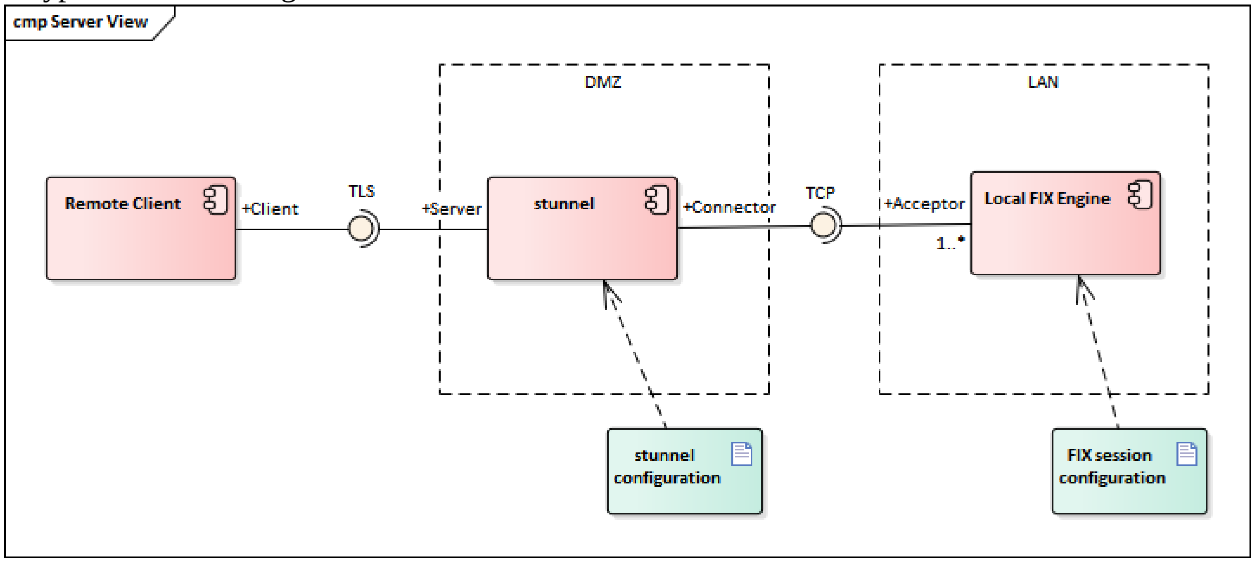

*Figure 2Typical server deployment*

## *1.3 Scope*

The guide is limited to the protocol stack of FIX over TCP/IP with TLS and does not cover every feature of Stunnel. This guide makes recommendations about Stunnel configurations that promote security and interoperability on the wire.
This configuration guide applies mainly to software implementation of TLS mostly commonly used with Stunnel, which is the OpenSSL library.

### 1.3.1 Out of scope

Stunnel also supports hardware TLS engines, but their configuration may be proprietary product-specific, so that is out of scope in this guide.
Network management, firewalls, and perimeter protection are out of scope.

## *1.4 References*

FIX-over-TLS (FIXS) Technical Standard Requirements and Guidance 2016, FIX Protocol Ltd.
Stunnel: Documentation, https://www.stunnel.org/docs.html. Software author: Michał Trojnara. Note that Stunnel is free software but is not open source; the author retains the copyright.
OpenSSL:Documentation, https://www.openssl.org/docs/ Cryptography and SSL/TLS Toolkit used by stunnel. Free and open source. The OpenSSL toolkit stays under a double license, i.e. both the conditions of the OpenSSL License and the original SSLeay license apply to the toolkit.
It is up to individual firms to decide whether software and licenses are suitable for their needs.

### 1.4.1 Versions and Updates

This guide is predicated on current versions of software as of the time of writing, Stunnel version 5.41 and OpenSSL version 1.1.0.
It is incumbent on users to stay current as security vulnerabilities are periodically discovered. Stunnel provides a mailing list for critical announcements and version updates at https://www.stunnel.org/cgi-bin/mailman/listinfo/stunnel-announce.

# 2 Overview of Stunnel Options

The following options apply to all use cases.

| Category | Major options | Section reference |
|---|---|---|
| **Connectivity** | `connect`<br>`accept`<br>`protocol=connect` | Connectivity |
| **TLS** | `sslVersion` | TLS Version |
| **Cipher suites** | `ciphers` | Cipher suites for use with certificates<br>Cipher suites for use with pre-shared keys |
| **Authentication** | `cert`<br>`requireCert`<br>`verifyChain`<br>`verifyPeer` | Public Key Infrastructure (PKI) |
| | `PSKidentity`<br>`PSKsecrets` | Pre-shared Key Authentication |
| **Local administration** | `output` | Local Administration |

## *2.1 Connectivity*

### 2.1.1 Client

The `connect` option is used to specify a remote server address for client connectivity.

```
      connect = <[HOST:]PORT>
```

If host is not supplied, then localhost is used.

#### 2.1.1.2 HTTP Connect

Stunnel supports HTTP Connect (RFC 2817) for client connection only. It makes it possible to tunnel through a firewall to an HTTP proxy that establishes a TCP connection to the ultimate destination. Traffic between Stunnel and the proxy is packaged as HTTP GET and POST requests and responses. HTTP Connect is only needed if the network restricts outgoing connections to HTTP protocol or its well-known port. There is no benefit otherwise since it introduces another network hop to reach the final destination.

```
      ; the address of the HTTP proxy
      connect = <[HOST:]PORT>
      protocol = connect
      ; the address of the remote server
      protocolHost = <HOST:PORT>
      ; username and password for basic authentication to the proxy[1]
      protocolUsername = <USERNAME>
      protocolPassword = <PASSWORD>
```

### 2.1.2 Server

The `accept` option is used listen for connections by clients.

```
      accept = [HOST:]PORT
```

If host is not supplied, then the server listens to the specified port on all network interfaces.

### 2.1.3 IP Address or Domain Name

A host may be specified to clients either as an IP address or a domain name may be provided. Some private networks do not provide DNS. A domain name has the possible advantage of location independence, but on the other hand, DNS can be an attack vector for denial-of-service attacks.

### 2.1.4 Socket Options

For FIX, it is recommended to disable the Nagle algorithm of TCP to reduce latency.

```
      ; global options for accept/local/remote sockets
      socket=a:TCP_NODELAY=1
      socket=l:TCP_NODELAY=1
      socket=r:TCP_NODELAY=1
```

## *2.2 TLS Version*

Stunnel configuration should conform to the minimum TLS version recommended by the FIXS Technical Standard. Currently, only TLS 1.2 is recommended.

```
      sslVersion = TLSv1.2
```

## *2.3 Cipher Suites*

Use the cipher suites recommended by the FIXS Technical Standard. This is configured in Stunnel with a service-level option, where CIPHER_LIST is a colon-delimited list of cipher suite names:

```
      ciphers = <CIPHER_LIST>
```

The list of enabled cipher suites is transmitted from client to server in the order that they are configured in Stunnel, and in accordance with the TLS protocol, the list is in order of preference. The server will accept the most preferred cipher suite that it has in common with the client. OpenSSL uses non-standard cipher suite names for configuration. Each name is a combination key exchange, encryption and message digest algorithms. (Within the TLS protocol, cipher suites are designated on the wire by a two-byte code, not the names listed below. Although its configuration names are non-standard, OpenSSL translates its names to the standard codes. See

[1] NTLM authentication is also supported on Windows only.

https://tools.ietf.org/html/rfc5246#page-75 or https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml for the codes used in TLS 1.2.)

### 2.3.1 Cipher suites for use with certificates

| Standard cipher suite name | OpenSSL name | Code |
|---|---|---|
| TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 | DHE-RSA-AES128-SHA256 | 0x00,0x67 |
| TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 | DHE-RSA-AES128-GCM-SHA256 | 0x00,0x9E |
| TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 | DHE-RSA-AES256-SHA256 | 0x00,0x6B |
| TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 | DHE-RSA-AES256-GCM-SHA384 | 0x00,0x9F |
| TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 | ECDHE-ECDSA-AES128-SHA256 | 0xC0,0x23 |
| TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 | ECDHE-ECDSA-AES128-GCM-SHA256 | 0xC0,0x2B |
| TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 | ECDHE-ECDSA-AES256-SHA384 | 0xC0,0x24 |
| TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 | ECDHE-ECDSA-AES256-GCM-SHA384 | 0xC0,0x2C |
| TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 | ECDHE-RSA-AES128-SHA256 | 0xC0,0x27 |
| TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 | ECDHE-RSA-AES128-GCM-SHA256 | 0xC0,0x2F |
| TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 | ECDHE-RSA-AES256-SHA384 | 0xC0,0x28 |
| TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 | ECDHE-RSA-AES256-GCM-SHA384 | 0xC0,0x30 |

### 2.3.1 Cipher suites for use with pre-shared keys

| Standard cipher suite name | OpenSSL name | Code |
|---|---|---|
| TLS_DHE_PSK_WITH_AES_128_CBC_SHA | PSK-AES128-CBC-SHA | 0x00,0x90 |
| TLS_DHE_PSK_WITH_AES_256_CBC_SHA | PSK-AES256-CBC-SHA | 0x00,0x91 |

## 2.4 Authentication Methods

### 2.4.1 Public Key Infrastructure (PKI)

PKI is a form of asymmetric encryption used in the authentication of peers in TLS. It uses public and private keys to encrypt and decrypt data. Keys are issued by certificate authorities in the form of certificates. X.509 is the prevailing standard for public key certificates.
The remainder of this section pertains only to PKI authentication. See below for PSK as an alternative.

#### 2.4.1.3 Providing a certificate

A certificate file provides the public key of a party and its chain of certificate authorities. It is transmitted to a peer during TLS negotiation. A server certificate is always required, while a client certificate is only needed for mutual authentication.
An X.509 certificate file is configured in Stunnel as follows.

```
      cert = <CERT_FILE>
```

The private key may be stored in the same file as the certificate. Alternatively, the corresponding private key may be stored in its own file:

```
      key = <KEY_FILE>
```

#### 2.4.1.4 Require a peer certificate

An X.509 certificate should always be required by clients from servers. A client certificate is only required by servers to implement mutual authentication in TLS.
In Stunnel, this is configured with a service-level option:

```
      requireCert = yes
```

This option is implied by `verifyChain` or `verifyPeer` as described below, but it is not an error if `requireCert` is also present in those cases.

#### 2.4.1.5 Verify a peer certificate

Two mutually exclusive options are available for certificate verification. One of these options should always be specified for server certificate verification (client configuration), and for client verification in the case of mutual authentication (server configuration).

##### *2.4.1.5.1 Verify a chain of trust*

This option verifies an entire chain of trust including the peer and its chain of certificate authorities:

```
      verifyChain = yes
```

Additionally, one of the following options must be provided with `verifyChain` to tell the location of Certificate Authority certificates. The directory `/etc/ssl/certs` is the conventional location of a hashed directory containing trusted CA certificates.

```
      CApath = <CA_DIRECTORY>
```

Or

```
      CAfile = <CA_FILE>
```

The `verifyChain` option implies `requireCert = yes.`
One of the checks performed is whether any of the certificates in the chain has expired; no configuration is needed to activate that check. The `verifyChain` option should also be used with one of the methods described below to activate a check for revoked certificates.
Verifying an entire chain has a significant performance cost for a server, so it is generally recommended to use certificate pinning instead (see below).

##### *2.4.1.5.2 Leaf certificate pinning*

This option verifies a peer by storing its certificate on first use or ahead of time and considering it trusted on subsequent uses. It is less costly than verifying the entire chain each time and avoids certain vulnerabilities regarding certificate authorities. Subsequent uses match the public key to a stored value. However, it requires a check to make sure that the subject of the certificate matches the stored one. The configuration option is:

```
      verifyPeer = yes
```

Optionally, one or more of the following settings may be added to verify the subject of the certificate, either by host name, IP address or email address:

```
      checkEmail = <EMAIL>
      checkHost = <HOST>
      checkIP = <IP>
```

More than one address may be configured. A certificate is verified successfully if there is at least one subject match.
The `verifyPeer` option implies `requireCert = yes.` The `verifyPeer` option should be used with one of the methods described below to activate a check for revoked certificates.

#### 2.4.1.6 Certificate Revocation List (CRL)

Certificates are occasionally revoked by a certificate authority so it is necessary to check whether a peer certificate is still valid. One technique is to search for a certificate in a locally stored Certificate Revocation List. Two configuration parameters are available in Stunnel to tell locations of local CRLs.
To configure the path to locally stored CRL files, set:

```
      CRLpath = <CRL_DIRECTORY>
```

Alternatively, the path to single CRL file may be given:

```
      CRLfile = <CRL_FILE>
```

Due to the difficulty of maintain current CRLs for all possible peers and their CAs, it is recommended that OCSP be used instead (see below).

#### 2.4.1.7 Online Certificate Status Protocol (OCSP)

The relieve users of maintaining CRLs locally, the Online Certificate Status Protocol provides a means to use an external service to check for certificate revocation. Such a service is called an OCSP responder.
To configure a specific OCSP responder address:

```
      OCSP = <Responder_URL>
```

The OCSP responder addresses may be derived from AIA (Authority Information Access) extension of a peer certificate, if present, with this configuration option:

```
      OCSPaia = yes
```

The primary use of AIA extension is to fetch missing intermediate certificates in a chain of trust.

### 2.4.2 Pre-shared Key Authentication

A client may be authenticated by a pre-shared secret key (PSK), or shared secret, instead of installing a certificate on the client side. PSK authentication is implemented with symmetric key encryption. Both peers must possess the key.
Both client and server installations of Stunnel would enter the same configuration as follows.
Leave `requireCert = no` by default, and set these options instead:

```
      PSKidentity = <IDENTITY>
      PSKsecrets = <KEY_FILE>
```

Each line of the file should have the following format:

```
        IDENTITY:KEY
```

The key is required to be at least 20 characters long. The file should not be world-readable nor world-writable.

## *2.5 Platform-specific configuration*

### 2.5.1 Windows

Microsoft CryptoAPI engine allows for authentication with private keys stored in the Windows certificate store. Each section using this feature needs this option:

```
      engine = capi
```

### 2.6 Local Administration

#### 2.6.1 Configuration file

Conventionally, the configuration file is named `/etc/stunnel/stunnel.conf`, but this may be overridden on the command line. Alternative configuration files are presented below for each use case.
On UNIX when the 'chroot' option is used, Stunnel will look for all its files (including the configuration file, certificates, the log file and the pid file) within the chroot jail.

#### 2.6.2 Local security

It is recommended to drop root privileges if Stunnel is started by root.

```
setuid = nobody
setgid = nogroup

chroot = <DIRECTORY>

; PID file is created inside the chroot jail (if enabled)
pid = /usr/local/var/run/stunnel.pid
```

#### 2.6.3 Logging

On UNIX, Stunnel logs to Syslog by default. The default log level is “notice”, but it may be overridden with `debug = <LEVEL>`.
Logging may be directed to a file as follows (required on Windows):

```
output = /usr/local/var/log/stunnel.log
```

# 3 Use Cases

See the main FIX-over-TLS document for a full explanation of the use cases described below.

## 3.1 Service defaults

The following options apply to all configurations.

### 3.1.1 TLS version

Leave the global options and service defaults listed in the Stunnel sample configuration file, except for the following.
Change the TLS version configuration to enable only the recommended TLS version.

```
sslVersion = TLSv1.2
```

### 3.1.2 Cipher suites

By default, all cipher suites enabled in the underlying engine are available. TLS negotiates between client and server to select one of the suites that are supported by both peers. Therefore, the list is common to client and server configurations.
Override the default cipher suites by explicitly listing suites that are recommended by the current FIXS Technical Standard and are supported by the Stunnel engine. With the default engine,

OpenSSL names must be used. The list of cipher names is colon delimited. The order that cipher suites are listed determines their priority in negotiation between client and server.

```
ciphers = DHE-RSA-AES128-SHA256:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-
AES256-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-
SHA256:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-
ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES128-GCM-
SHA256:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES256-GCM-SHA384
```

## 3.2 Mutual authentication using certificates

In these use cases, both the server and client are authenticated by a certificate in TLS negotiation.

### 3.2.1 Server Configuration

**Cases 1a and 1b:** **Mutual TLS with Leaf Certificate Pinning**

```
[fix-server]
; Listen port for TLS connection from client
accept  = 0.0.0.0:7777
; TCP connection to local FIX engine in the form host:port
connect = 127.0.0.1:3003
cert = /etc/stunnel/stunnel-5.41/stunnel.pem
verifyPeer = yes
CAfile = /etc/ssl/certs/stunnel2.pem
ciphers = DHE-RSA-AES128-SHA256:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-
AES256-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-
SHA256:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-
ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES128-GCM-
SHA256:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES256-GCM-SHA384
sslVersion = TLSv1.2
```

### 3.2.2 Client Configuration

**Case 1a:** **Mutual TLS with Leaf Certificate Pinning**

```
[fix-client]
client = yes
accept = 127.0.0.1:2002
connect = 192.168.153.32:7777
ciphers = DHE-RSA-AES128-SHA256:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-
AES256-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-
SHA256:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-
ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES128-GCM-
SHA256:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES256-GCM-SHA384
cert = /etc/stunnel/stunnel2.pem
verifyPeer = yes
CAfile = /etc/ssl/certs/stunnel.pem
```

**Case 1b:** **Mutual TLS with CA Pinning**

```
[fix-client]
client = yes
accept = 127.0.0.1:2002
connect = 192.168.153.32:7777
ciphers = DHE-RSA-AES128-SHA256:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-
AES256-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-
SHA256:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-
ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES128-GCM-
SHA256:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES256-GCM-SHA384
cert = /etc/stunnel/stunnel2.pem
verifyChain = yes
CAfile = /etc/ssl/certs/stunnel.pem
```

## *3.3 Using Pre-Shared keys*

In this use case, mutual authentication is achieved via a pre-shared key rather than with certificates.

### 3.3.1 Server Configuration

**Case 2: Mutual TLS with Pre-Shared Keys**

```
[fix-server]
; Listen port for TLS connection from client
accept  = 0.0.0.0:7777
; TCP connection to local FIX engine in the form host:port
connect = 127.0.0.1:3003
PSKsecrets = /home/orc/stunnel/stunnel-5.41/psk.txt
ciphers = PSK-AES256-CBC-SHA:PSK-AES128-CBC-SHA
sslVersion = TLSv1.2
requireCert = no
```

### 3.3.2 Client Configuration

**Case 2: Mutual TLS with Pre-Shared Keys**

```
[fix-client]
client = yes
accept = 127.0.0.1:2002
connect = 192.168.153.32:7777
ciphers = PSK-AES128-CBC-SHA:PSK-AES256-CBC-SHA
PSKsecrets = psk.txt
PSKidentity = clientQA1
sslVersion = TLSv1.2
requireCert = no
```

## *3.4 Using FIX credentials and server certificates*

In this use case, the server is authenticated by a certificate in TLS negotiation while the client is authenticated by credentials passed in the FIX Logon message after the TLS session has been established. Therefore, server authentication is configured in Stunnel, while client authentication is configured in FIX engines. FIX engine configuration is not shown here since it is different for each engine.

### 3.4.1 Server Configuration

**Cases 3a and 3b: Simple TLS with FIX Credentials for Client**

```
[fix-server]
; Listen port for TLS connection from client
accept  = 0.0.0.0:7777
; TCP connection to local FIX engine in the form host:port
connect = 127.0.0.1:3003
cert = /etc/stunnel/stunnel.pem
ciphers = DHE-RSA-AES128-SHA256:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-
AES256-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-
SHA256:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-
ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES128-GCM-
SHA256:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES256-GCM-SHA384
sslVersion = TLSv1.2
requireCert = no
```

### 3.4.2 Client Configuration

**Case 3a: Simple TLS with Leaf Certificate Pinning**

```
[fix-client]
client = yes
accept = 127.0.0.1:2002
connect = 192.168.153.32:7777
ciphers = DHE-RSA-AES128-SHA256:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-
AES256-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-
SHA256:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-
ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES128-GCM-
SHA256:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES256-GCM-SHA384
verifyPeer = yes
CAfile = /etc/ssl/certs/stunnel.pem
```

**Case 3b: Simple TLS with CA Pinning**

```
[fix-client]
client = yes
accept = 127.0.0.1:2002
connect = 192.168.153.32:7777
ciphers = DHE-RSA-AES128-SHA256:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-
AES256-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-
SHA256:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-
ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES128-GCM-
SHA256:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES256-GCM-SHA384
verifyChain = yes
CAfile = /etc/ssl/certs/ca-chain.cert.pem
checkIP = 192.168.153.32
```

# 4 Appendix

The following utilities are helpful to manage certificates used by Stunnel.

## 4.1 Generating a self-signed certificate

A self-signed server certificate may be generated with this command line as shown below. OpenSSL prompts for qualified subject name. It is important that the Common Name (CN) matches the host as clients will access it, not by a local domain.
This command example places the certificate and private key in separate files. In the example, the certificate is valid for 365 days.

```
openssl req -new -x509 -days 365 -nodes -out cert.pem -keyout key.pem
```

A certificate may be displayed with this command:

```
   openssl x509 -text -in stunnel.pem
```

## 4.2 Listing supported ciphers

Command to show a verbose list of ciphers with high strength:

```
      openssl ciphers -v -s HIGH
```

## 4.3 Downloading a certificate

A certificate may be downloaded from a remote server for purposes of certificate pinning. Portions from the command output can be copied to a certificate file in .pem format.

```
   openssl s_client -showcerts -connect www.example.com:443
```