

FTPS vs. SFTP: What to Choose

Pros and Cons of two competitive protocols

Eugene Mayevski

October 11, 2007

File transfer over the network using FTP protocol (defined by RFC 959 and later additions) has its roots in 1980, when the first RFC for FTP protocol was published. FTP provides functions to upload, download, and delete files; create and delete directories; and read directory contents. Although FTP is very popular, it has certain disadvantages that make it harder to use. The major drawbacks are lack of the uniform format for directory listing (this problem has been partially solved by introducing the MLST command, but it's not supported by some servers) and the presence of the secondary connection (DATA connection). Security in FTP is provided by employing SSL/TLS protocol for channel encryption as defined in RFC 2228. The secured version of FTP is called FTPS.

In UNIX systems, another security standard has grown. It was the SSH family of protocols. The primary function of SSH was to secure remote shell access to UNIX systems. Later, SSH was extended with file transfer protocol—first SCP (in SSH 1.x), and then SFTP (in SSH2). Version 1 of the SSH protocol is outdated, unsecure, and generally not recommended for use. Consequently, SCP is not used anymore and SFTP gains popularity day by day.

The "SFTP" abbreviation is often mistakenly used to specify some kind of Secure FTP, by which people most often mean FTPS. Another (similar) mistake is that SFTP is thought to be some kind of FTP over SSL. In fact, SFTP is an abbreviation of "SSH File Transfer Protocol." This is not FTP over SSL and not FTP over SSH (which is also technically possible, but very rare).

SFTP is a binary protocol, the latest version of which is standardized in RFC 4253. All commands (requests) are packed to binary messages and sent to the server, which replies with binary reply packets. In later versions, SFTP has been extended to provide not just file upload/download operations, but also some file-system operations, such as file lock, symbolic link, creation, and so forth.

Both FTPS and SFTP use a combination of an asymmetric algorithm (RSA, DSA), a symmetric algorithm (DES/3DES, AES, Twofish and so on), and a key-exchange algorithm. For authentication, FTPS (or, to be more precise, SSL/TLS protocol under FTP) uses X.509 certificates, whereas SFTP (SSH protocol) uses SSH keys.

X.509 certificates include the public key and certain information about the certificate owner. This information lets the other side verify the integrity of the certificate itself and authenticity of the certificate owner. Verification can be done both by computer and to some extent by the human. An X.509 certificate has an associated private key that is usually stored separately from the certificate for security reasons.

A SSH key contains only a public key (the associated private key is stored separately). It doesn't contain any information about the owner of the key. Neither does it contain information that lets one reliably validate the integrity and authenticity. Some SSH software implementations use X.509 certificates for authentication, but in fact they don't validate the whole certificate chain—only the public key is used (which makes such authentication incomplete and similar to SSH key authentication).

Here's the brief list of pros and cons of the two protocols:

FTPS

Pros:

- Widely known and used
- The communication can be read and understood by humans
- Provides services for server-to-server file transfer
- SSL/TLS has good authentication mechanisms (X.509 certificate features)

- FTP and SSL/TLS support is built into many Internet communication frameworks

Cons:

- Doesn't have a uniform directory listing format
- Requires a secondary DATA channel, which makes it hard to use behind the firewalls
- Doesn't define a standard for file name character sets (encodings)
- Not all FTP servers support SSL/TLS
- Doesn't have a standard way to get and change file and directory attributes

SFTP

Pros:

- Has good standards background that strictly defines most (if not all) aspects of operations
- Has only one connection (no need for DATA connection)
- The connection is always secured
- The directory listing is uniform and machine-readable
- The protocol includes operations for permission and attribute manipulation, file locking, and more functionality

Cons:

- The communication is binary and can't be logged "as is" for human reading
- SSH keys are harder to manage and validate
- The standards define certain things as optional or recommended, which leads to certain compatibility problems between different software titles from different vendors
- No server-to-server copy and recursive directory removal operations
- No built-in SSH/SFTP support in VCL and .NET frameworks

What to Choose

As usual, the answer depends on what your goals and requirements are. In general, SFTP is technologically superior to FTPS. Of course, it's a good idea to implement support for both protocols, but they are different in concepts, in supported commands, and in many other things.

It's a good idea to use FTPS when you have a server that needs to be accessed from personal devices (smartphones, PDAs, and the like) or from some specific operating systems that have FTP support but don't have SSH/SFTP clients. If you are building a custom security solution, SFTP is probably the better option.

As for the client side, the requirements are defined by the server(s) that you plan to connect to. When connecting to Internet servers, SFTP is more popular because it's supported by Linux and UNIX servers by default.

For private host-to-host transfer, you can use both SFTP and FTPS. For FTPS, you would need to search for a free FTPS client and server software or purchase a license for commercial one. For SFTP support, you can install an OpenSSH package that provides free client and server software.

Developer Tools

If you are a software developer and need to implement file transfer capability in your application, you will be searching for the components to do the job.

In .NET, you have built-in support for FTPS in the .NET Framework (see the `FtpWebRequest` class). However, functionality of this class is severely limited, especially in the SSL/TLS control aspect. The .NET Framework doesn't include any support for SSH or SFTP.

In VCL, you have a selection of free components and libraries that provide FTP functionality. When you add OpenSSL to them, you can get FTPS for free. If you don't want to deal with OpenSSL DLLs, you can use one of the commercially available libraries for SSL and FTPS support. Again, there are no freeware SFTP components available for .NET.

If you use a tool with which you have to use ActiveX controls, you need to search for commercial FTPS or SFTP controls. No free controls are available. SecureBlackbox library provides both FTPS and SFTP support for .NET, VCL and ActiveX technologies.