Certified File Transfer Professional (CFTP) Official Study Guide

Version 1.3

pro2col
the file transfer experts

Table of Contents

I. Introduction to CFTP

“Certified File Transfer Professional” ("CFTP") is a vendor-independent file transfer certification for IT professionals. A CFTP is qualified to evaluate, deploy, configure, maintain and support secure file transfer, managed file transfer and workflow automation technology such as:

- FTP servers, FTPS servers and SFTP servers
- "Web Transfer" and "Web Client" servers, including WebDAV servers
- Accelerated File Transfer, including "Extreme File Transfer" and UDP-Based solutions
- "Ad Hoc" File Sharing and File Synchronization (or “EFSS”)

CFTP Certifying Body

The certifying body behind the CFTP program is the Certified File Transfer Professional Advisory Board. It is managed and administered by Pro2col, Ltd, a vendor-independent technology firm that encourage file transfer interoperability, education, and best practices through training and professional certification.

The CFTP exam, process and study guide are shaped by a group of experienced advisors with a wide range of managed file transfer and secure file transfer experience. The training and examination was also tested with dozens of early adopters, without whom there would be no curriculum at all. *(See full acknowledgements in Appendix A)*

CFTP Training and Test Procedure

The Certified File Transfer Professional certification exam is an open book test based on material found in this study guide and recorded online training.

Each CFTP exam consists of 60 questions randomly selected from a much larger pool of approved questions. Answers are multiple choice, and all answers are presented in a randomized order. (The chances of any two exams being identical is remote.) Each exam must be completed in 90 minutes or less, and a passing grade is 80% (at least 48 correct questions).

II. File Transfer Concepts

File transfer is used every day by hundreds of millions of people and businesses to share documents, provide remote access to information and automate business process. The data moved by file transfer processes is also often called “bulk data” or “batch data” because files often communicate more than one idea, transaction or receipt.

When File Transfer Should Be Used

File transfer works best when the output from a business workflow needs to be handed off to another person or process, and any response to the output can wait at least a few seconds. File transfer is also

used when information already stored in files (e.g., “documents”, “extracts” or “reports”) simply needs to be shared with other people or processes, and no response is needed.

For example, a workflow that involved posting purchase orders and turning around a response in five minutes would be an excellent candidate for file transfer. A workflow that involved end users sharing spreadsheets with other end users would also be an excellent candidate for file transfer.

When File Transfer Should NOT Be Used

File transfer is NOT the best choice when data must be streamed, or responses to original data are expected back almost immediately. Examples of streamed data that would NOT be appropriate candidates for file transfer include live video broadcasting and audio applications like VOIP. Examples of workflow that would NOT be appropriate candidates for file transfer (because they require exceedingly quick responses) include stock trade orders to be executed within milliseconds, and low-level command-and-control signals within a computer system.

File Transfer’s Role in Collaboration

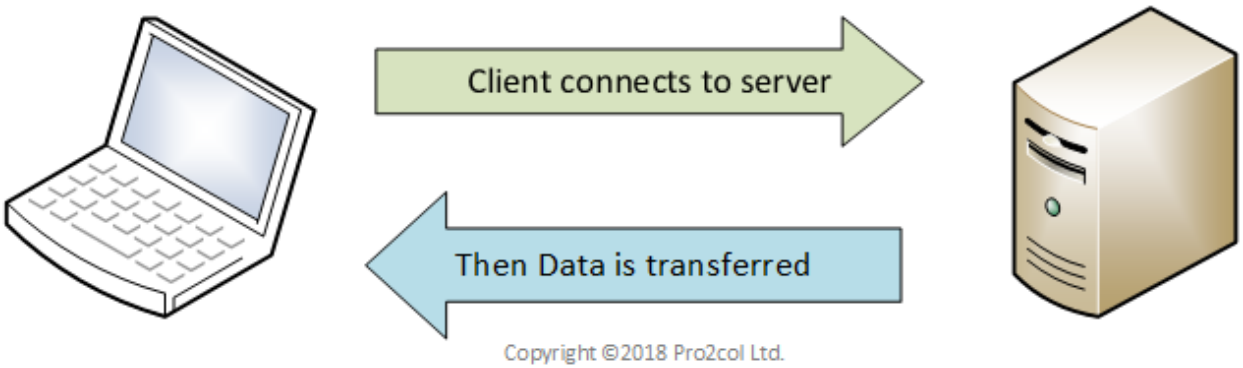
Many organisations use file transfer as a collaboration tool between geographically distant parties. Both “ad hoc” file send and share models and the shared folder models are popular. However, file transfer solutions do not typically offer the file locking or versioning available in “collaboration-first” solutions or in “source control” or “document control” repositories.

File Transfer Client and Server Concepts

Almost all file transfer implementations use a "client/server" computing architecture. In a client/server relationship, the client opens a connection and initiates activity while the server receives the connection and responses to activity requests. When file transfer uses a client/server architecture, the client requests and sends files, and the server provides and receives files. The simplest way to picture this is that servers are reactive (answering requests), while clients are proactive (initiating transfers).

This architecture also allows multiple clients to connect to and perform commands against a single central server or cluster of servers.

A common example of a file transfer client/server architecture is an "FTP client" that requests and sends files, paired with a "FTP server" that provides and receives files. Another example is a "web browser" client that requests and sends files, and a "web server" that provides and receives files.



Somewhat confusingly, some software products perform both a client and server function (most commonly seen with FTP servers that have event driven actions)

Additional functions of file transfer clients and servers are listed below.

File transfer **CLIENTS**:

- connect to servers
- identify the end user or process controlling the client (usually by presenting a username)
- authenticate using credentials such as password, key or certificate
- issue commands to list available files, manage directories and perform other activities
- initiate the transfer (upload and download) of files

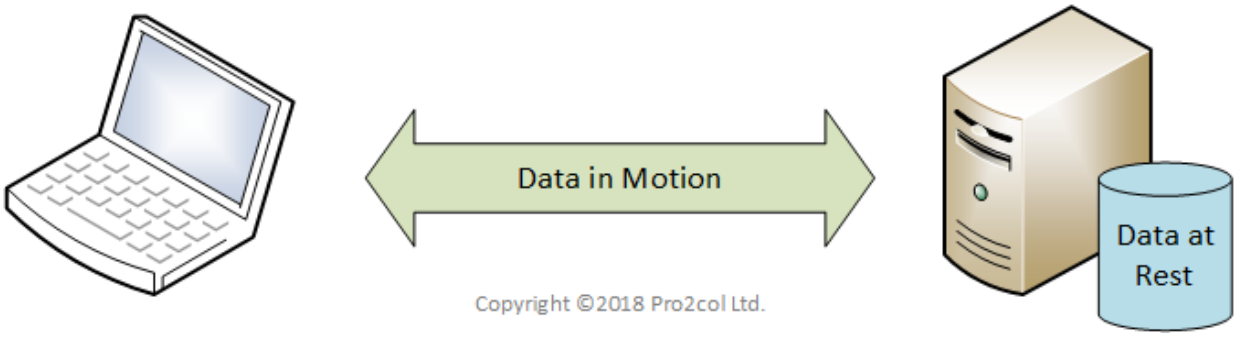
File transfer **SERVERS**:

- listen for connections from clients
- validate credentials for identified users
- grant appropriate permissions to authenticated users ("authorize" them)
- when requested, receive uploaded files, serve up files for download, and perform local file and directory maintenance (such as deleting a file)
- perform other actions when requested, when an event fires, or on a scheduled basis (such as automatically deleting old files from a particular folder at a certain time every night)

"Data in Motion" and "Data at Rest" Encryption Concepts

File transfer workflows often involve a sequence of steps that transfer, store and/or process data, and different encryption concepts apply to each step. When files are:

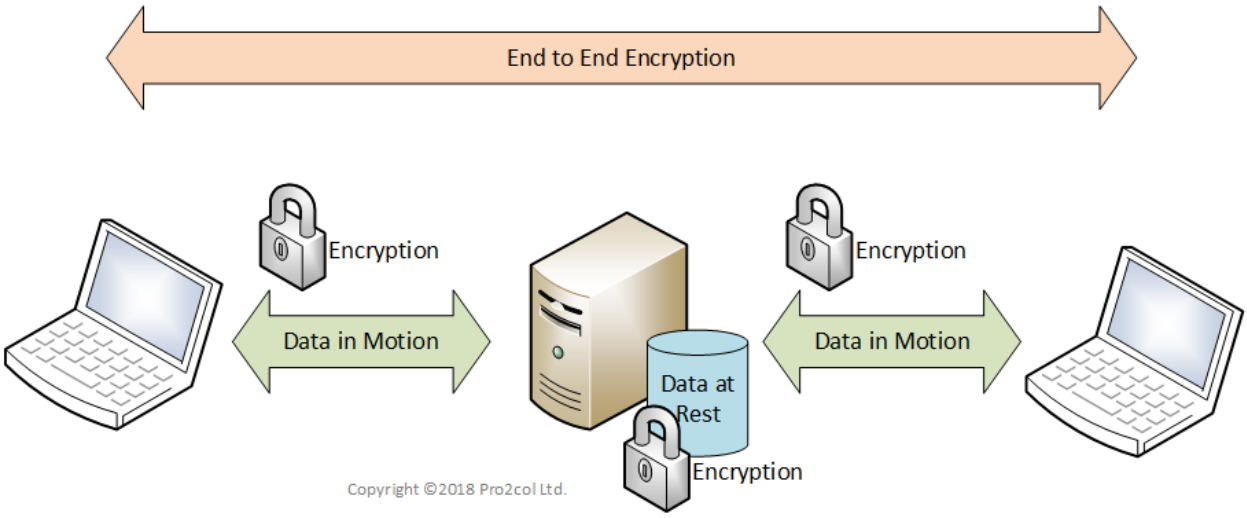
- **Being transferred** between different systems, "**data in motion**" encryption concepts apply.
- **Stored** within a particular system, "**data at rest**" encryption concepts apply.
- **Being processed,** then files are usually unencrypted (or "in clear text").



Similarly, when encryption is applied to hide data and protect it from tampering, it said to:

- "Secure data in motion" when it transferred between two systems.
- "Secure data at rest" when it is stored within a particular system

When data moves through a workflow that uses encryption on every transfer and storage step (i.e. through both "data in motion" and "data at rest"), the workflow is said to offer "end-to-end encryption."



Encryption Concepts

File transfers use encryption to protect credentials and data in transit and at rest. Encryption uses a process called an “encryption algorithm” (or “cipher”) and a secret piece of information called a “key” to transform sensitive data into encrypted versions of the same data. The sensitive data is often referred to as “plaintext” or “cleartext” data, and the encrypted version of the same data is often called the “cipher text.”

Encryption is often divided into two major categories:

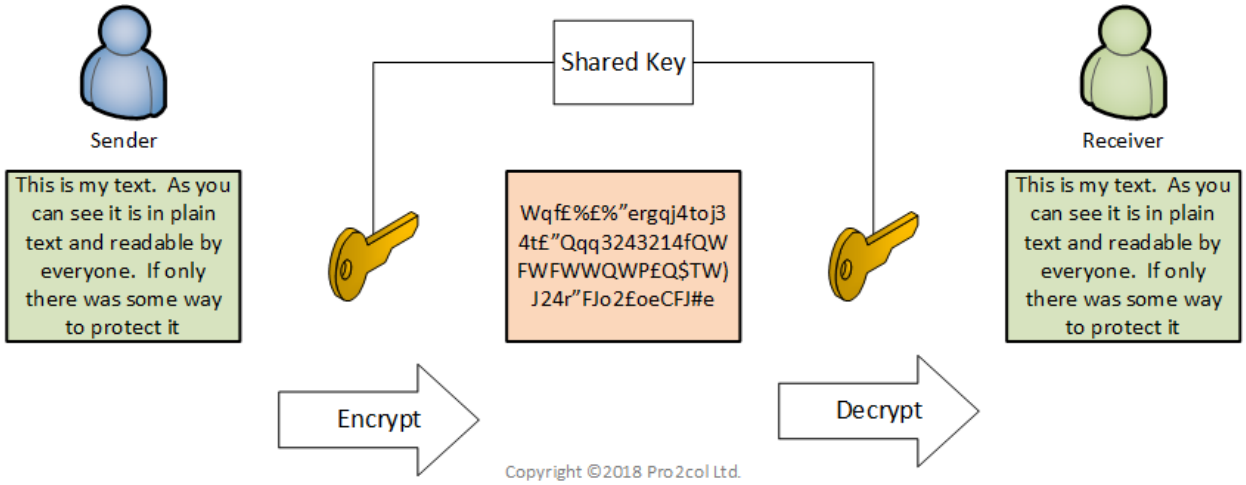
- **Symmetric encryption** that protects the cipher text using a shared key.
- **Asymmetric encryption** that uses the recipients public and private key to encrypt and then decrypt the cipher text. This is also known as PKI

When file transfer professionals simply discuss “encryption” the topic is almost always about symmetric/reversible encryption. Terms like “fingerprints”, “thumbprints”, “one way” and “integrity checks” usually indicate that hashing is being discussed.

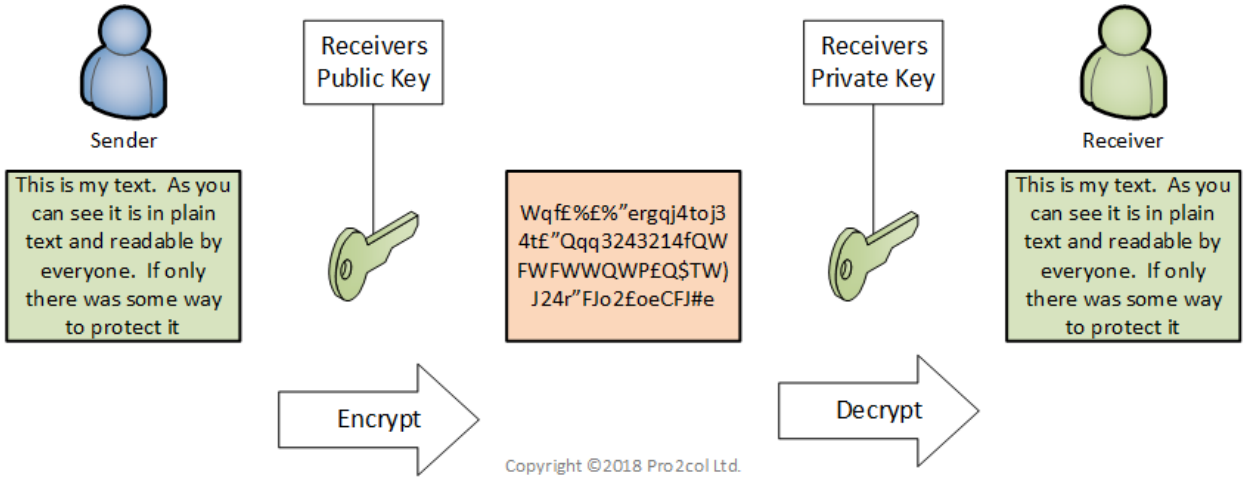
Symmetric encryption uses a particular encryption algorithm and a secret key to convert a sensitive plaintext into an encrypted ciphertext. The same algorithm and key can be then used to decrypt the resulting ciphertext and convert it back into the original plaintext.

Note that encrypted ciphertexts are usually about the same length or longer than their original plaintexts. This is often the result of “padding” operations in encryption algorithms that add extra characters.

Symmetric Encryption



Asymmetric Encryption

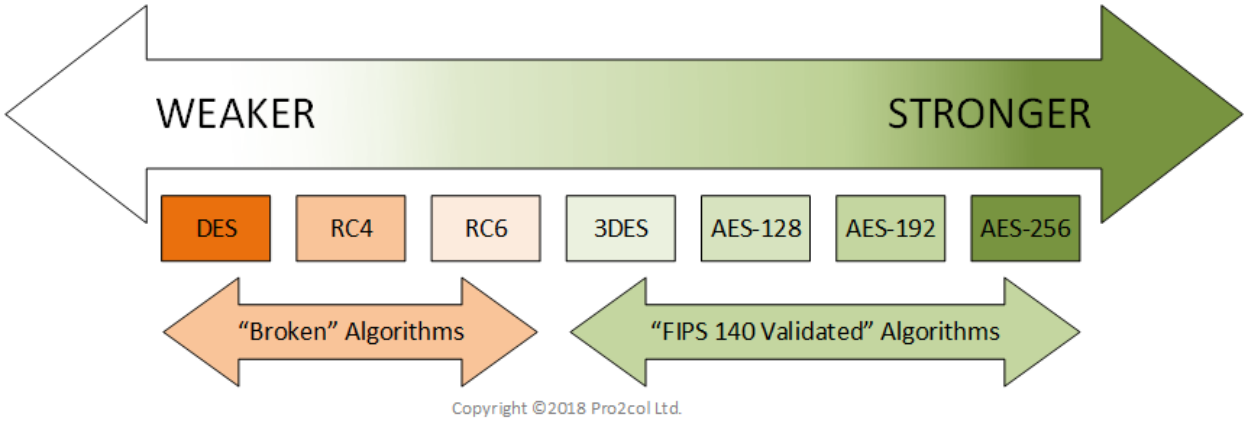


Common Encryption Algorithms

Over the years hundreds of different encryption algorithms have been developed, but file transfer experts typically only focus on a few. First of all, there are weak, or broken algorithms like DES (pronounced "dez") and RC4. Then there are stronger algorithms like RC6 and Twofish. Finally there are strong algorithms that have also been tested, validated and promoted by national governments.

In the United States, Canada and much of the rest of the world, algorithms validated by the US National Institute of Standards and Technology (NIST) and its Canadian counterpart (CSE) are used as a trustworthy set of nationally validated algorithms. These algorithms are called "FIPS-validated algorithms," "FIPS 140 ciphers" or similar names. (FIPS stands for "Federal Information Processing Standard.") The current FIPS standard is 140-2

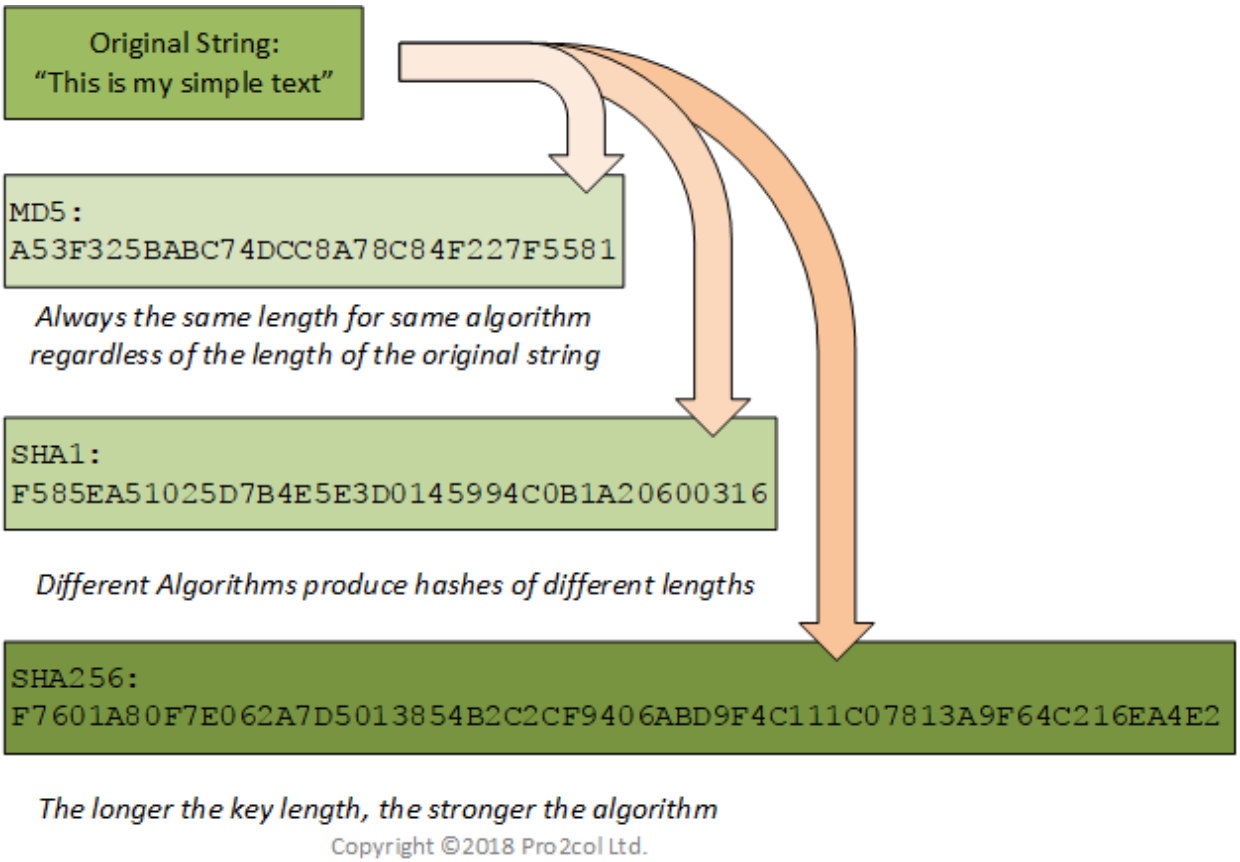
The current set of FIPS 140-2 validated algorithms includes 3DES (pronounced "triple dez") and AES. 3DES supports key lengths of 56, 112 and 168 bits. AES supports key lengths of 128-, 192- and 256-bits.



Hashing Concepts

Unlike the encryption we've looked at so far, hashes are mathematic calculations (“algorithms”) that process sets of data and yield fixed-length signatures that describe, but do not reveal, the original data. They are also called “fingerprints,” “thumbprints,” “one-way encryption” and other names.

For example, using the same hashing algorithm, the hash of “toast” might be 4a8b while the hash of “frog legs” might be f36f, where both hashes would be four characters long even though the source data to produce each hash was of different lengths.



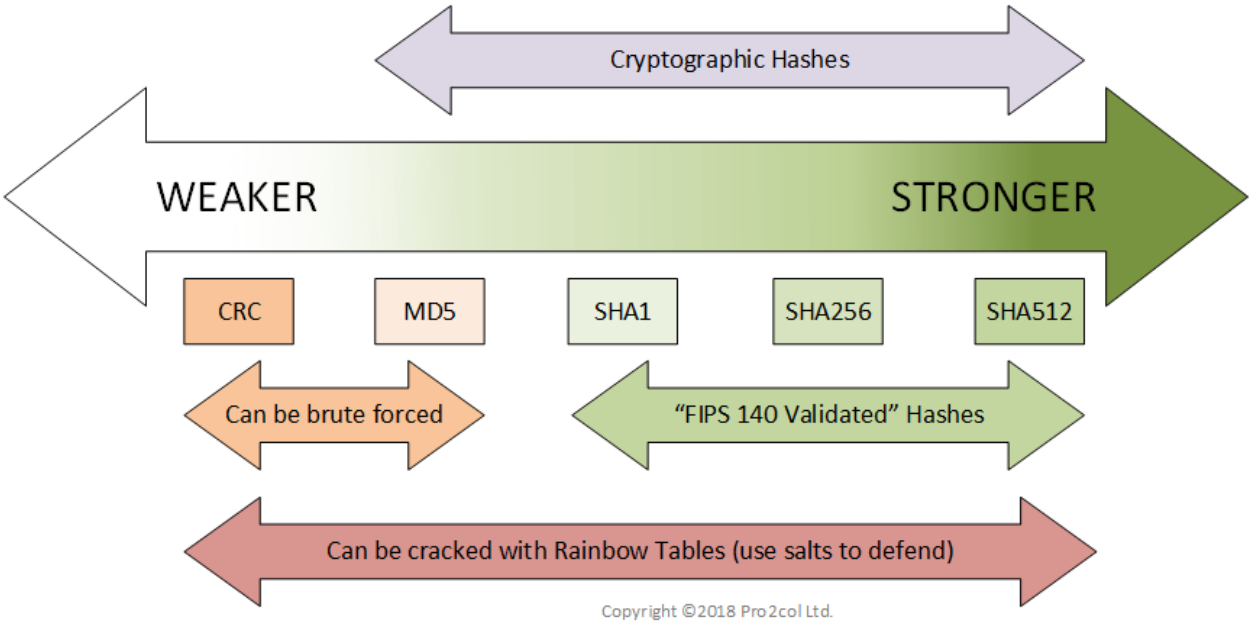
Hashing differs from encryption in many ways. Two important differences are:

- **Length of Result:** Encryption yields ciphertexts whose lengths are often similar to their original plaintexts. Hashing yields signatures whose lengths are always the same for a particular algorithm, regardless of the length of the original plaintexts.
- **Recovery of Plaintext**: Ciphertexts created by symmetric encryption will yield the original plaintext if plied with the correct encryption algorithm and key. Signatures created by hashing cannot reveal the plaintext used to create them.

Common Hash Algorithms

Hashes are often used to provide data integrity (to make sure a transmission is complete and has been altered) and authenticate users (to store signatures of user passwords and keys without actually storing the sensitive credentials).

Data integrity hashes may be either cryptographic (such as MD5, SHA1, SHA256 and SHA512 in increasing strength) or non-cryptographic (such as CRC), while authentication hashes (such as stored passwords) are almost always cryptographic.



Note that "non-repudiation" - the ability to prove the origin, integrity and authenticated delivery of files – usually requires cryptographic hashes (e.g., not CRC).

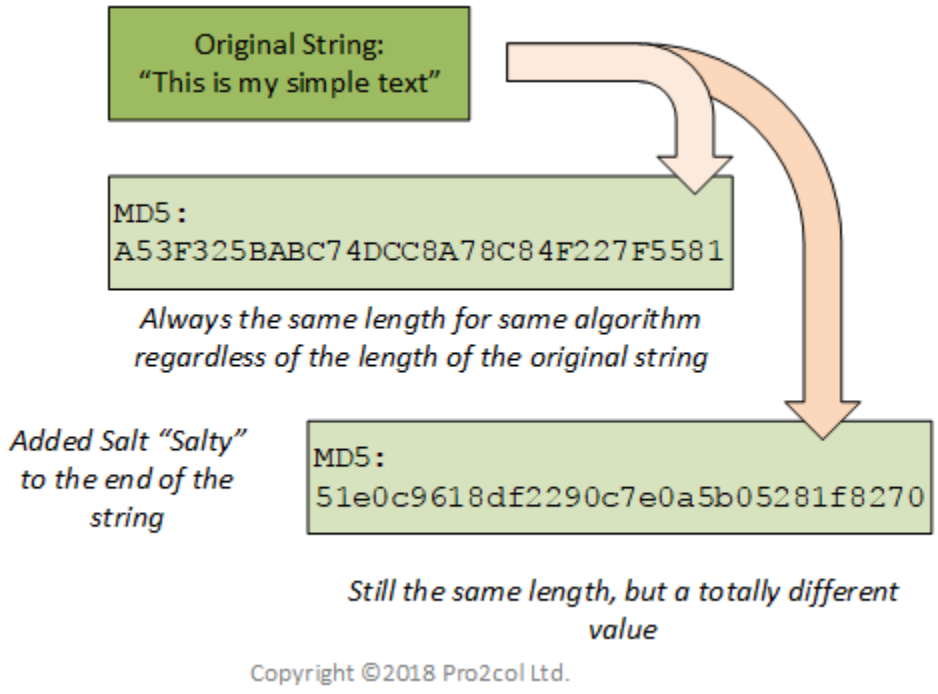
Salting Hashes

Even though hashes do not directly reveal the original cleartext used to create them, their role as unique signatures can be exploited when a limited number of values could be used to generate them. Unfortunately, this is a common occurrence when authentication hashes are used to protect passwords.

For example, if my six-character password of “Fred07” is stored as a hash of “b234”, an attacker who knew what algorithm was used to generate that hash could easily generate a list of ALL possible passwords of six characters or less, and could then perform a “reverse” lookup on his own list to figure out that “b234” means “a password of Fred07.” (A list of generated password/hash pairs like this is called a “rainbow table.”)

Security researchers and hackers have already calculated rainbow tables containing billions of password hashes using popular algorithms like MD5 and SHA1, and many of these tables are online. (e.g., try MD5 hash “5d41402abc4b2a76b9719d911017c592” at http://md5.gromweb.com/ to look up a password of “hello.”) With these tools at hackers’ disposal, it is no longer safe to store passwords or other long term credentials in regular hashed fields.

To increase the safety of hashed storage (and avoid hash table lookups), random “salt” data is now often added to hashes. The extra salt data changes the final value of the hashes and makes them much more resistant to cracking and lookup in rainbow tables.

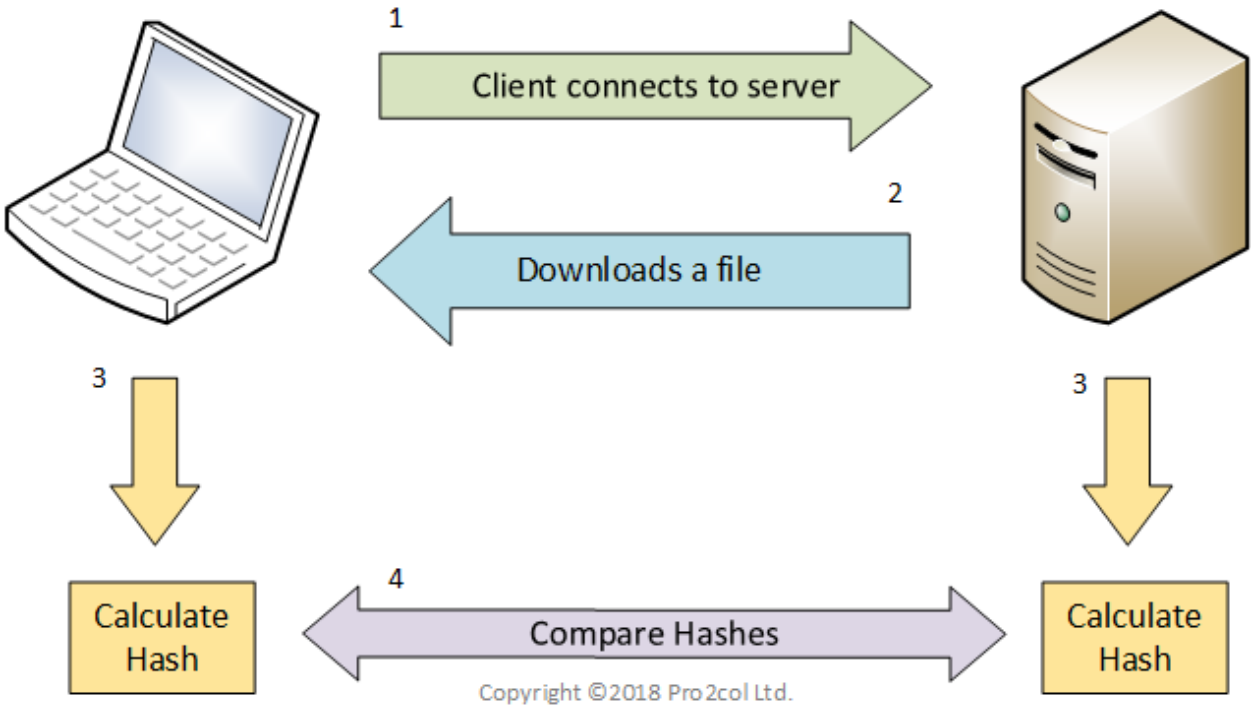


Note that when hashes are used for password storage, the use of salts is almost always recommended. However, when hashes are used for quick file transfer integrity checks (see next section), the use of salts is rare.

Providing Integrity Checks

Hashes are most often used to check whether one set of data matches another set of data without performing a full byte-to-byte comparison.

In the most common example, hashing is performed by a file transfer client and file transfer server immediately after the client has completed its transfer to or from the server. When the client asks the server for the hash of the file just transferred, it will log an error (and may delete a file or retry a transfer) if the client and server hashes disagree. (There is no need to send a second copy of the file to check file integrity – just the hashes will do.)



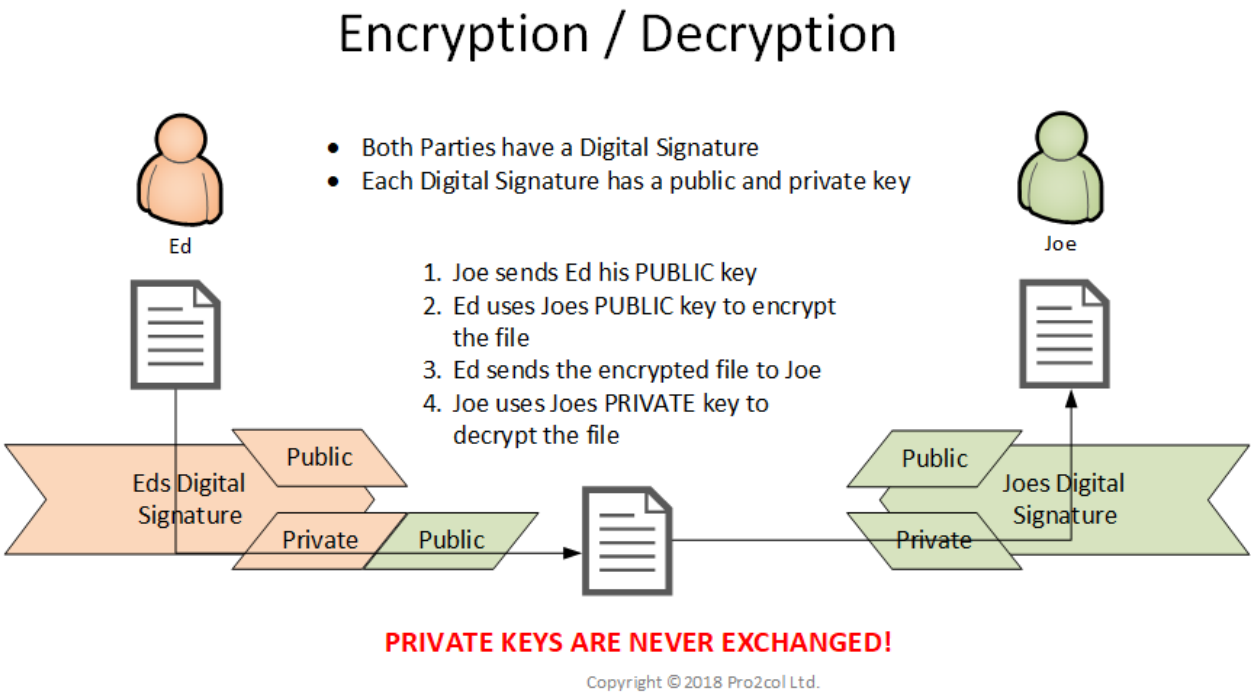
In another common example, hashing is frequently used to “store passwords” as hashes. (The passwords are never actually retained – not even in reversibly encrypted ciphertexts.) When end users attempt to authenticate with a password, each incoming password is hashed and the HASHES of the incoming and saved passwords are compared (instead of performing a cleartext password-to-password comparison).

Finally, hashing is also used in conjunction with PGP,encryption to 'sign' a file; this signature is known as a 'message digest'. Combining the digest with users private key proves the authenticity of the message (That content and sender are legitimate).

PKI (Public-Key Infrastructure) Concepts

One of the most important concepts in modern authentication and encryption is “public-key infrastructure” (PKI). PKI underlies HTTPS, FTPS, SFTP, PGP, AS2 and other protocols and encryption mechanisms commonly used to secure files in transit or at rest. Although PKI can be hard to grasp and sometimes configure, it is nonetheless a popular and powerful encryption technique because it allows two parties to securely exchange data without ever exposing their full keys. *(PKI advantages and disadvantages are covered in more depth in the "At Rest" Section)*

In PKI each party has a two-part key (or certificate). One of the two parts is a “public key”, which is shared freely with each and every person or process that needs to communicate securely and authenticate communications with its owner. The second part is the “private key”, which, as its name suggests, is kept private by the owner and should never be shared with other users.

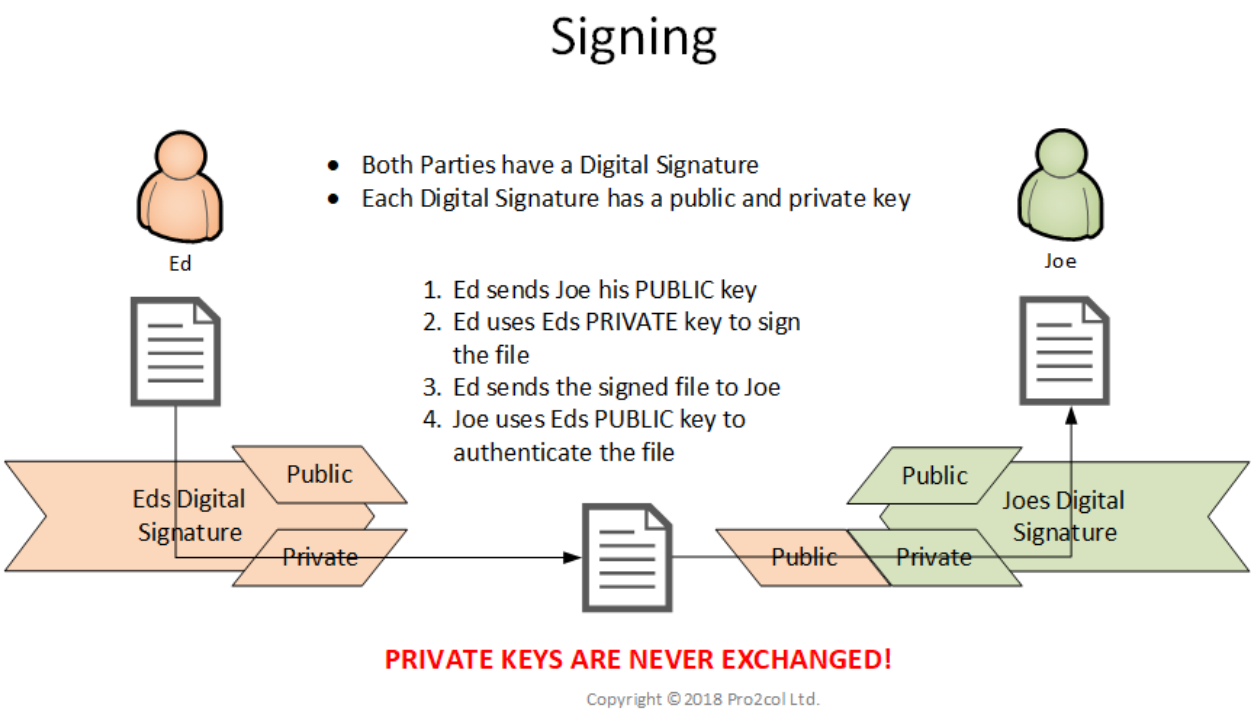


To safely transfer a file, each party involved in the transfer sends his PUBLIC key to the other party. This step is called “key exchange.” (PRIVATE keys are NOT exchanged!) Then, the sender encrypts a file using the recipient’s PUBLIC key, and sends it to the recipient. Finally, the recipient decrypts the file using his own PRIVATE key.

This technique works because the mathematics behind PKI ensure that at least two different keys can be used to encrypt and decrypt a file, and each of these keys is made up of public and private components from both the sender and recipient.

When the sender encrypts a file, the sender could also “sign” the file with his PRIVATE key. (The signature is essentially a hash of the unencrypted file combined with the key.) This act allows the recipient (or anyone else with access to the sender’s public key) to verify the file’s signature using the

sender's PUBLIC key.



Role in Security in Transit

In PKI-secured protocols such as HTTPS or SFTP, a server shares its public key/cert with each and every end user who connects. This allows each end user to validate the identity of the server, and is the basis for the red/green status code you see in your HTTPS URLs in web browsers today.

Immediately after a server is verified, clients can also (but often do not) send a specific public key/cert back to the server to allow the server to validate the identity of the client. In practice, web browsers often generate a cert just to set up quick SSL/TLS connections, and servers often ignore these provided certificates, but high security applications often use "client certificate authentication" or "client key authentication".

Whenever both sides (two end user or a client and server) both identify themselves with their own certificate, it is called "mutual authentication" or "strong authentication."

Whenever client certificate or client key authentication is used in addition to a password, it is called "two factor authentication."

Role in Security at Rest

In PKI-secured encryption schemes such as PGP or SMIME, each party shares his or her public key with the other party, sometimes through a centrally-managed, publicly-accessible "key store". *(More information about these protocols is provided in the "At Rest" encryption section)*

Obtaining a Certificate

System administrators need to be able to apply for and apply certificates to servers.

The process starts with the generation of a public/private key-pair on the requesting server. This key-pair may already exist (as in the case of Microsoft IIS which uses the machine RSA keys by default), or may need to be generated. Programs like OpenSSL can achieve this.

Once a key pair exists, a Certificate Signing Request (CSR) needs to be generated to allow a Certificate Authority (CA) to vouch for it. The CSR is generated by a certificate application (such as OpenSSL or IIS) and contains two vital pieces of information – the public key (from the key-pair), and the fully qualified domain name that the certificate is to used with – for example www.example.com. Other information can be included during the generation process to identify the organization that is requesting the certificate (Country, State, Company etc.). The request is signed using the private key from the key-pair and encoded as a Base64 PKCS#10 text file. (PKCS#10 is a standard defining the encoding format of CSRs). The CSR is then passed to a CA who will generate and issue an X.509 digital certificate signed by the CA's own certificate and containing the content of the CSR.

Recently it has become more common to request a "wildcard" certificate. This is a special kind of certificate that can be used to protect many different resources in a given domain, so for example a wildcard certificate named *.example.com can be used to protect files.example.com, www.example.com and mail.example.com. Commonly, a single private key is generated at the CA and used on all of the various servers that the certificate protects, however for a higher level of security each servers private key may be used instead.

IPv4 and IPv6

Familiarity with Internet Protocol (IP), TCP and UDP is needed to design, maintain or troubleshoot file transfer systems. A full discussion of those topics is beyond the scope of the CFTP program, but familiarity with a few IP concepts is necessary to be an effective file transfer professional.

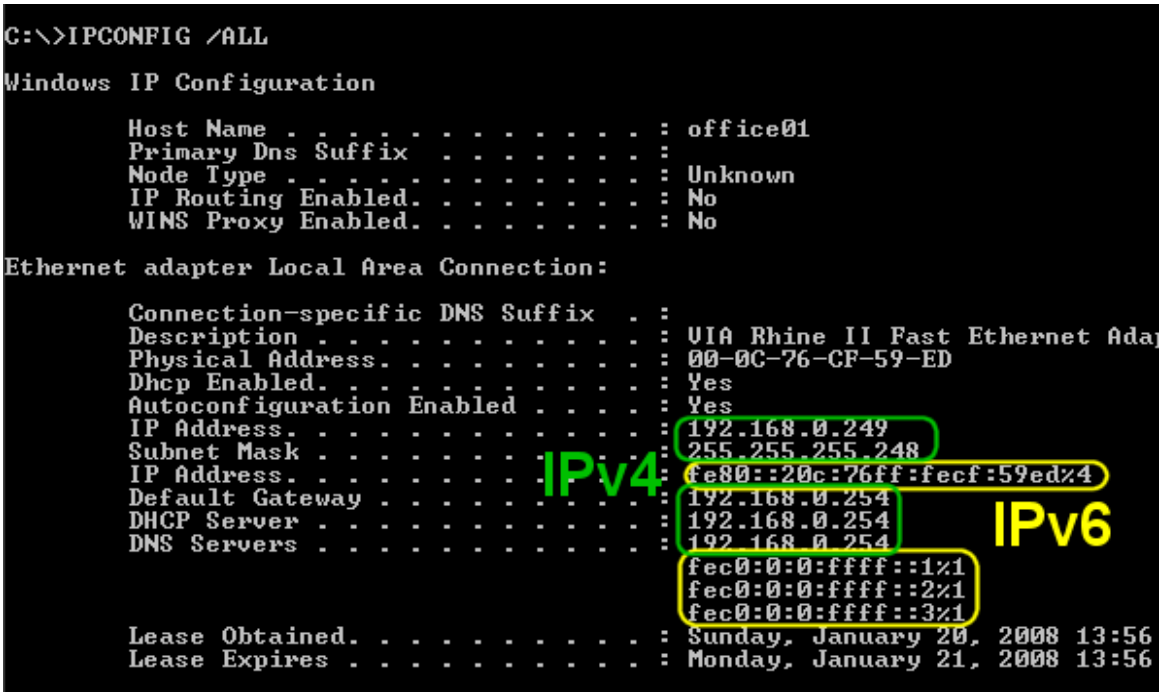
Private vs. Public Addresses

First, knowing which "private" (or "internal") IPv4 addresses are never routed over the Internet helps to quickly identify whether clients, servers or other resources are internal or external.

The list of private IPv4 addresses that usually identify internal resources includes 127.0.0.1 (or "localhost"), all 10.*.*.* addresses, all 192.168.*.* address and all addresses from 172.16.*.* through 172.31.*.*.

Second, a basic understanding of IPv6 addressing is necessary. Since the world has largely used up the pool of available IPv4 addresses on the Internet, IPv6's much larger pool of available addresses is getting more interesting, and it continues to be rolled out to combat worldwide IPv4 number exhaustion.

Valid IPv6 addresses include up to 8 hexadecimal octets (as opposed to IPv4's 4 octets) such as 11:00:00:00:00:00:77:DD. However, IPv6 addresses are often abbreviated with "double colons," when a series of 00 octets can be condensed, such as "11::77:DD" In fact, the "localhost" IPv6 address (or "00:00:00:00:00:00:00:01") is usually just expressed as "::1."



In the screenshot above, IPv4 private addresses and IPv6 addresses are being used together. IPv4 provides the default gateway through which traffic to the Internet and other non-local networks will flow. However, the machine also has an IPv6 address and has been instructed to try three IPv6-listening DNS servers using IPv6 if the primary IPv4-listening DNS cannot resolve a hostname.

Network Address Translation

Network Address Translation (NAT) is a method of allowing connections a range of IP addresses to use a single common IP address instead. This is most commonly employed at the exit point from a private network, where all packets leaving the network will receive the same common 'gateway' IP address.

Use of NAT has become increasingly popular in recent times, leading to an extension in the life expectancy of IPv4 (as users in an organization effectively 'share' the same IP address, fewer addresses are required). NAT is also frequently used at the firewall level to simplify external access to published resources like web or file transfer servers. However, NAT can also complicate correct identification of the end users source IP address.

Proxies

Proxy servers have an increasing use in networks, providing intermediary services to clients. In essence, the client make a request to a proxy server, which then routes it to a server which will handle it. Normally, the existence of a proxy server should be transparent to the client.

Reverse Proxy

Reverse proxy servers are most commonly employed to protect internet facing application servers from attack by internet users. The reverse proxy server is placed within the organisations DMZ; the application server (inside the network) then opens a channel to the reverse proxy server on a dedicated port. Internet based users access the proxy server which to all intents and purposes appears to be the application server instead. As long as they use the application in the way intended, everything works as expected. However, if the proxy server is compromised in some way, the intruder will not be able to access the network (no outbound ports are opened from the reverse proxy server to any other location).

Forward Proxy

Forward Proxies provide NAT for clients needing to exit the network when transferring data across the internet. Additionally, it negates the need for store-and-forward designs when file transfer clients are not permitted to directly exit the network. Dedicated forward proxies for file transfer solutions are not as common as dedicated reverse proxies.

III. Basic File Transfer Protocols

File transfer protocols are the languages file transfer clients use to exchange files with file transfer servers. At least one file transfer protocol has been built into almost every operating system, router and smart phone available today, but different protocols excel in different situations.

This section covers the basic protocols that every file transfer professional needs: FTP, FTPS and SFTP. It also briefly covers SCP, which is rarely used in managed file transfer applications but is common on *nix systems.

FTP

FTP (literally “file transfer protocol”) is the oldest file transfer protocol, and used to be the most popular of all file transfer protocols for many years. (FTP lost its “most popular” designation to HTTP in the mid-1990’s with the rise of the World Wide Web.)

FTP is implemented natively on most modern operating systems and many web browsers, but the protocol lacks the transport encryption necessary to secure credentials and data. *(A secure variation of FTP called* FTPS (SSL/TLS) *– described below - addresses the encryption issue.)*

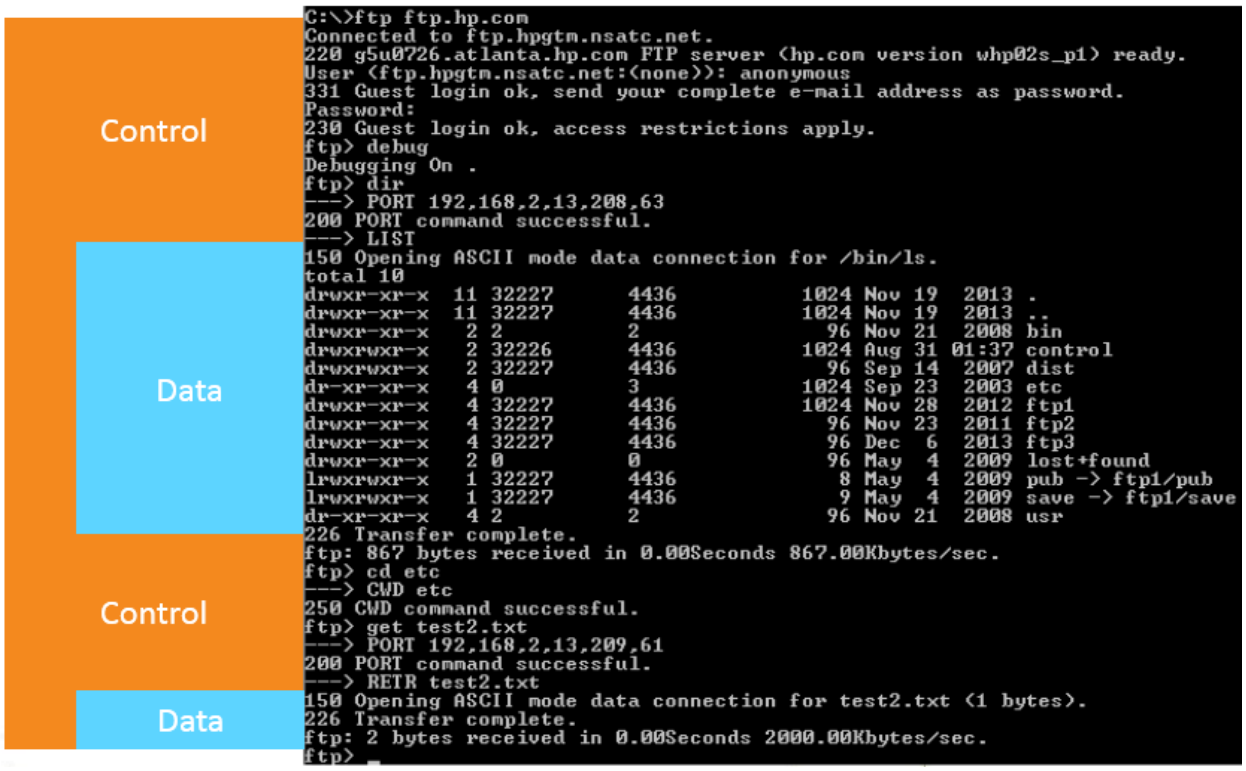
```
C:\>ftp ftp.hp.com
Connected to ftp.hpgtm.nsatc.net.
220 g5u0726.atlanta.hp.com FTP server (hp.com version whp02s_p1) ready.
User (ftp.hpgtm.nsatc.net:(none)): anonymous
331 Guest login ok, send your complete e-mail address as password.
Password:
230 Guest login ok, access restrictions apply.
ftp> debug
Debugging On .
ftp> dir
---> PORT 192,168,2,13,208,63
200 PORT command successful.
---> LIST
150 Opening ASCII mode data connection for /bin/ls.
total 10
drwxr-xr-x  11 32227      4436          1024 Nov 19  2013 .
drwxr-xr-x  11 32227      4436          1024 Nov 19  2013 ..
drwxr-xr-x   2 2          2               96 Nov 21  2008 bin
drwxrwxr-x   2 32226      4436          1024 Aug 31 01:37 control
drwxrwxr-x   2 32227      4436            96 Sep 14  2007 dist
dr-xr-xr-x   4 0          3             1024 Sep 23  2003 etc
drwxr-xr-x   4 32227      4436          1024 Nov 28  2012 ftp1
drwxr-xr-x   4 32227      4436            96 Nov 23  2011 ftp2
drwxr-xr-x   4 32227      4436            96 Dec  6  2013 ftp3
drwxr-xr-x   2 0          0               96 May  4  2009 lost+found
lrwxrwxr-x   1 32227      4436             8 May  4  2009 pub -> ftp1/pub
lrwxrwxr-x   1 32227      4436             9 May  4  2009 save -> ftp1/save
dr-xr-xr-x   4 2          2               96 Nov 21  2008 usr
226 Transfer complete.
ftp: 867 bytes received in 0.00Seconds 867.00Kbytes/sec.
ftp> cd etc
---> CWD etc
250 CWD command successful.
ftp> get test2.txt
---> PORT 192,168,2,13,209,61
200 PORT command successful.
---> RETR test2.txt
150 Opening ASCII mode data connection for test2.txt (1 bytes).
226 Transfer complete.
ftp: 2 bytes received in 0.00Seconds 2000.00Kbytes/sec.
ftp> _
```

The screenshot above shows FTP being used to list files on a remote server, and then download a file called “test2.txt.”

All FTP transfers use two different TCP connections:

- **Control** channel: established first, carries commands and brief responses (not data)
- **Data** channel: established as needed, carries file data and directory listings

The screenshot used to show FTP operations on the previous page is show again below, with the use of control and data channels broken out on the left.



(Notice the "Opening…data connection…" statements when data channels are opened.)

FTP connections always start by establishing a control channel from FTP client to FTP server. There is only ever one control channel for an FTP connection. Data channels are established as needed to transfer files or directory listings then closed when they are no longer required. A single FTP connection can use zero, one or many data channels as needed.

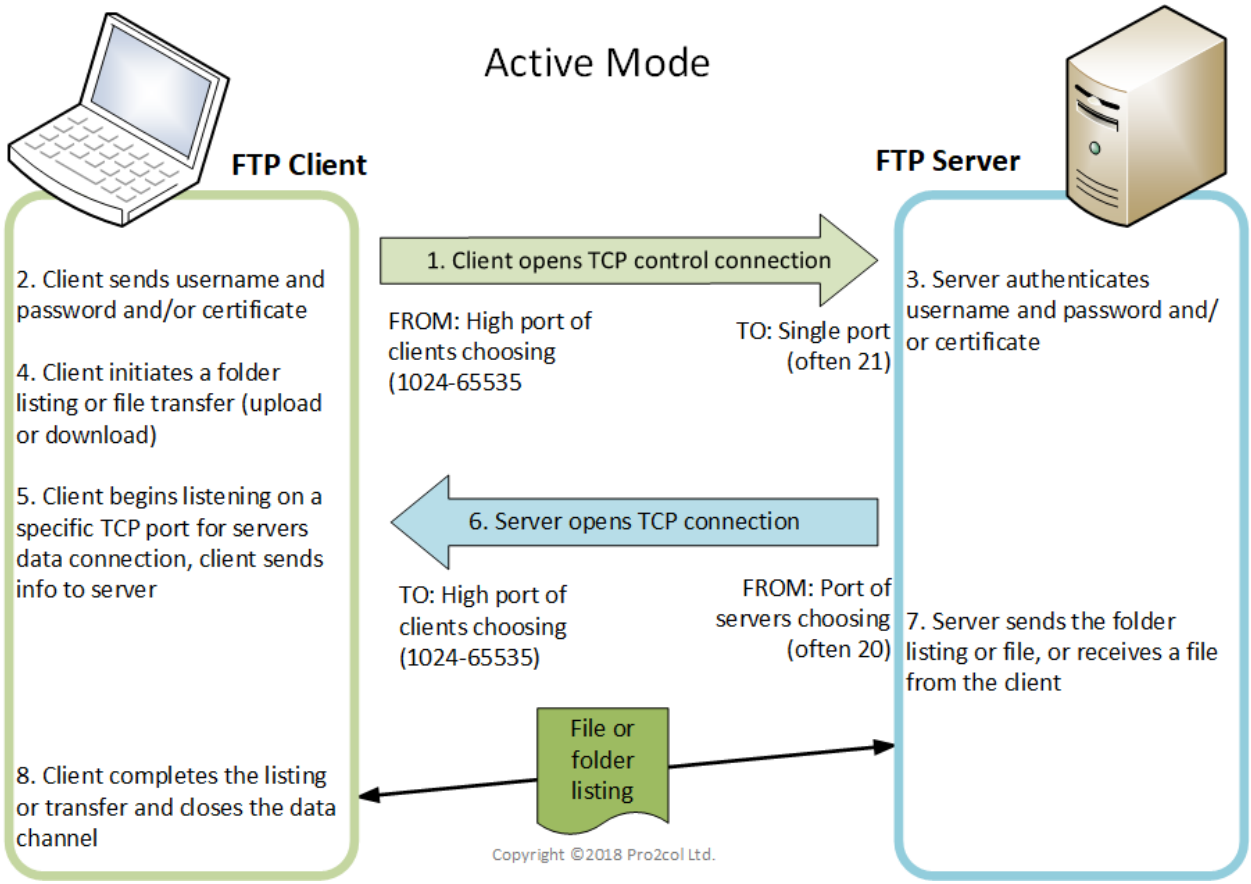
Originally, data channels connected from the FTP server back to the FTP client. *(Yes, from the server back to the client; this was before the Internet had firewalls.)* This mode is still supported and is called "active" mode. After firewalls between clients and servers became more popular, a new FTP mode that connected data channels from client to server became popular. This mode is called "passive" or "firewall friendly" mode and is usually preferred today. *(Passive mode is the default mode on almost every modern FTP client.)*

FTP and its secure FTPS variant *(covered below)* both support active and passive mode. Active and passive mode are also both supported when FTP and FTPS are used over IPv6. *(See EPSV and EPRT below.)*

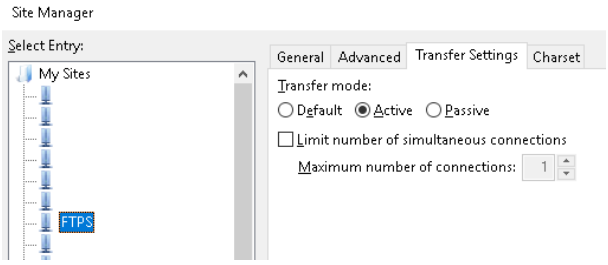
More information about both active and passive modes is found in the next two sections.

Active Mode

Active mode is the FTP file transfer mode in which the FTP server opens a connection back to the FTP client to support data and directory list transfers. This is the original mechanism behind FTP transfers and is still widely supported today. In an active connection, the client instructs the server to use a specific port on the client to open a data port to.



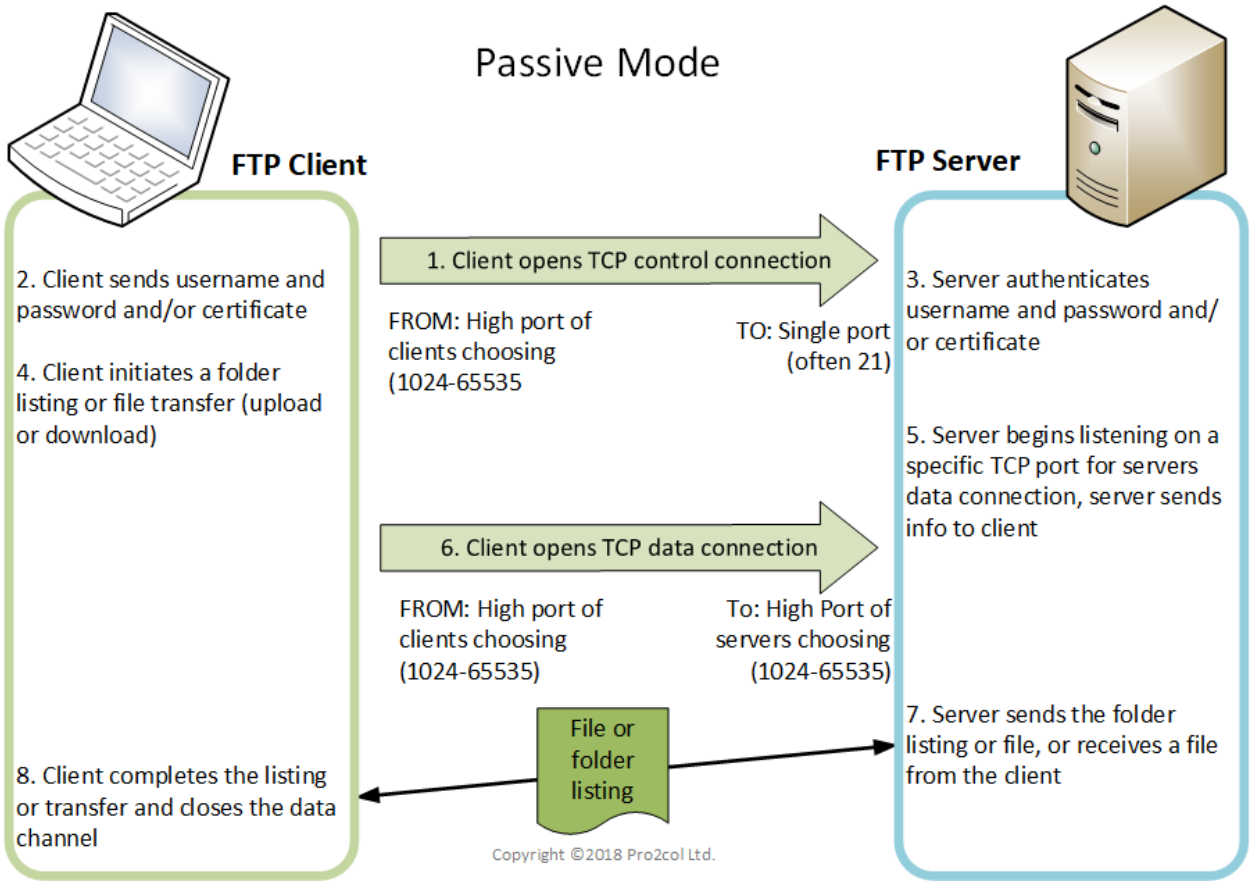
An example of an FTP client configuration that allows Active mode is provided below. An example of a command-line session using Active mode (note the "PORT" command) is also provided below.



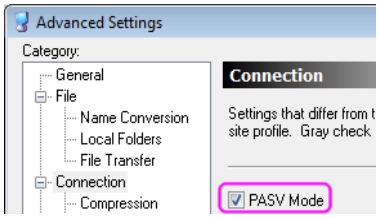

```
ftp> get test2.txt
---> PORT 192,168,2,13,209,61
200 PORT command successful.
---> RETR test2.txt
150 Opening ASCII mode data connection for test2.txt (1 bytes).
226 Transfer complete.
ftp: 2 bytes received in 0.00Seconds 2000.00Kbytes/sec.
```

Passive Mode (or "Firewall Friendly")

Passive mode is the FTP file transfer mode in which the FTP client opens a connection to the FTP server to support data and directory list transfers. This mechanism gained favor over active mode with the rise of firewalls and is also called "firewall friendly" mode. It is also often abbreviated "PASV".



An example of an FTP client configuration that allows Passive ("PASV") mode is provided below. An example of a command-line session using Passive mode (note the "PASV" command) is also provided.




```
ftp> get test2.txt
---> PASV
227 Entering Passive Mode (15,192,45,28,172,230)
---> RETR test2.txt
150 Opening ASCII mode data connection for test2.txt (1 bytes).
226 Transfer complete.
ftp: 2 bytes received in 0.39Seconds 0.01Kbytes/sec.
```

ASCII / Binary / EBCDIC Formatting

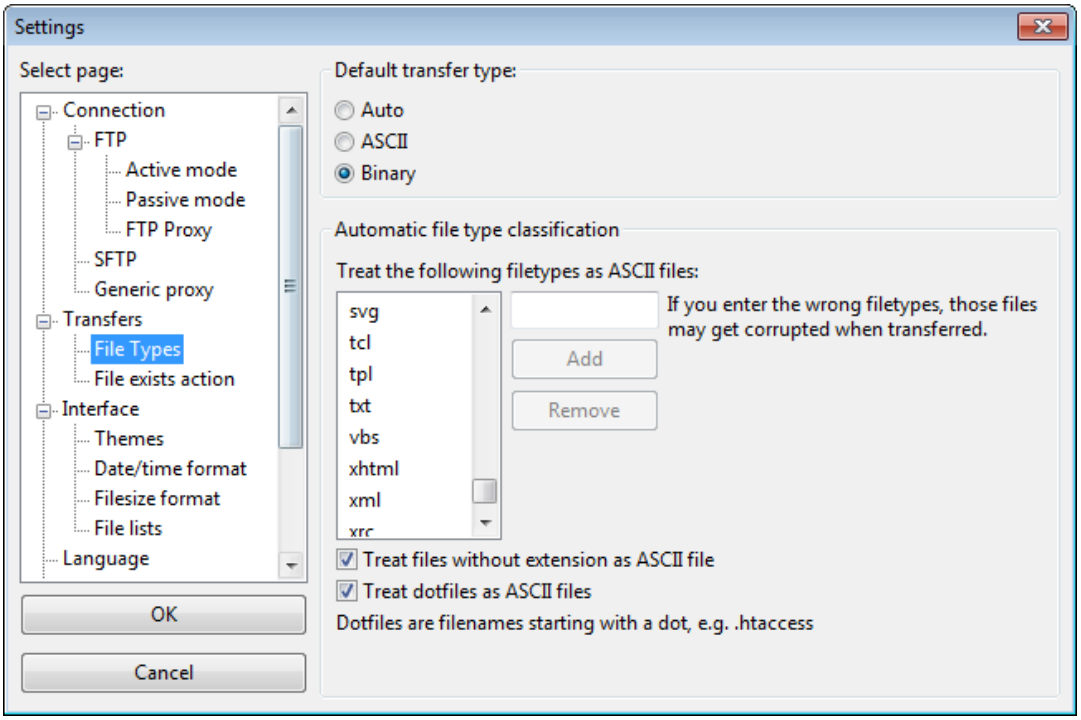
Some operating systems, such as Windows and Linux, use different character sequences to separate lines in their text files. Some systems, such as Unix/Window systems and mainframes, use different byte sequences to represent the same letters. FTP uses the "ASCII" and "EBCDIC" commands to handle these differences by automatically adding or removing line breaks and translating byte sequences as necessary.

The "ASCII" command is more common when PC systems are involved (Windows/Linux/Mac/mobile), and "EBCDIC" is often seen when mainframe and minicomputer systems are involved.

FTP also uses the "BINARY" command to indicate that a file transfer should not perform any translation or text formatting on a file. In all cases, these formatting commands are issued after authentication and before file transfer.

```
ftp> ascii
200 Type set to A.
ftp> binary
200 Type set to I.
```

Modern file transfer clients often use file extensions (e.g., ".txt" or ".png") to automatically set the appropriate text formatting type before each transfer. An example of this type of automatic configuration is provided below.



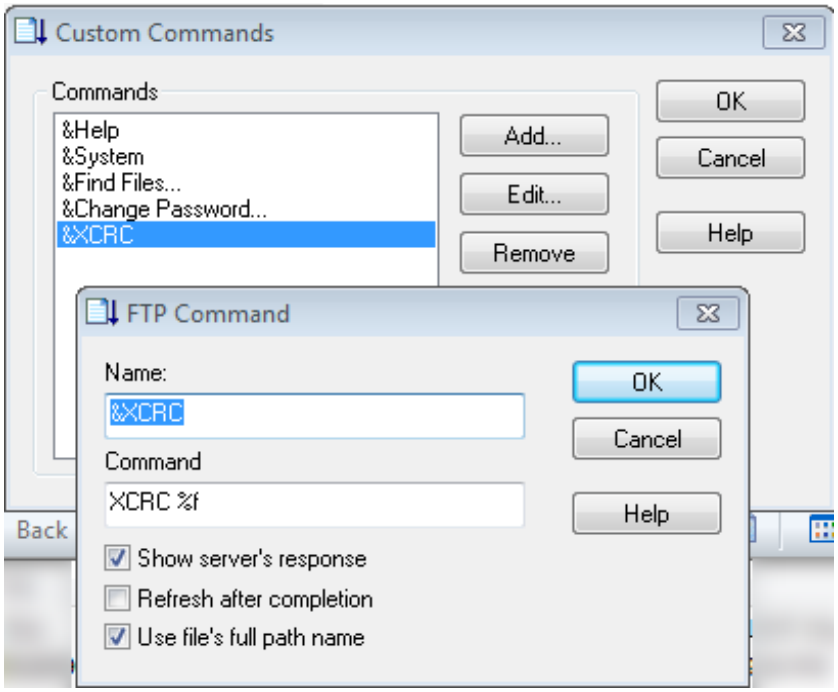
Custom "Quote" Commands

FTP servers often support commands beyond those offered by the basic FTP protocol, such as commands to change passwords, perform integrity checks and set blocking factors on storage systems that require explicit record lengths for each file. FTP "quote" commands fulfill this need by allowing arbitrary, per-server commands to be submitted by clients.

While it is quite possible to issue custom commands by hand…

```
ftp> quote noop
200 NOOP command successful.
ftp> quote syst
215 UNIX Type: L8
```

…many modern file transfer clients include configuration options that allow you to assign these to menu commands (as pictured here), run them silently as part of file transfer scripts, or automatically detect which commands are supported after connecting to each server.



EPSV and EPRT

EPSV is the modern version of the original FTP “PASV” command used to establish passive data channels. EPRT is the modern version of the original FTP “PORT” command used to establish active data channels. EPSV and EPRT are required when transferring files over IPv6 networks.

```
EPSV
229  Entering Extended Passive Mode (|||34347|)
```

EPSV and EPRT are currently supported by only a few servers. However, if EPSV and EPRT commands are supported by a particular server, that is often a sign that the server technology is kept up to date.

```
COMMAND>  FEAT
          211-Features:
           EPRT
           EPSV
```

Integrity Checks

Many FTP clients and servers support commands to perform integrity checks against files before and after transfers. These commands include “XCRC” (the non-cryptographic CRC check), “XMD5” (the cryptographic MD5 check), “XSHA1”, “XSHA256”, “XSHA512” and similar commands.

FTP integrity check commands are typically issued by FTP clients on individual files. For example, a client may issue the following command and receive the following response.

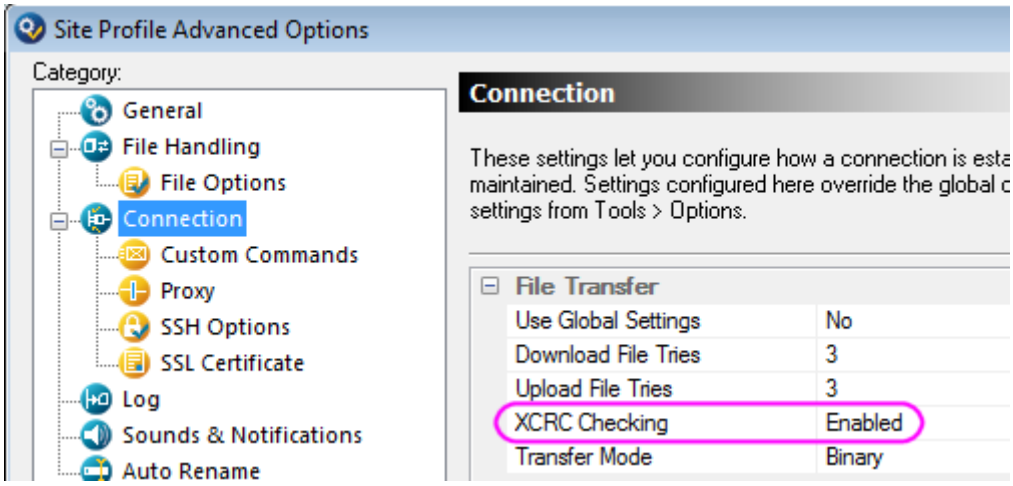
```
XMD5 filename.ext
250 5eb63bbbe01eeed093cb22bb8f5acdc3.
```

(“250” is a common FTP response code that indicates success and the rest of the response is the MD5 hash of the contents of “filename.ext”.)

Not all FTP servers support integrity checks, and very few support all the common variants (XCRC, XMD5 and XSHA1).

```
COMMAND>    XCRC "/robots.txt"
            500 XCRC not understood
ERROR>      XCRC command failed. The command may not be supported by this server.
```

FTP clients may try to use a particular type of integrity check by default (typically CRC or MD5), or the type of integrity check may be configurable item. Unfortunately, support for different integrity checks in any particular client is also rare. The following example is from an FTP client that supports integrity checks, but only non-cryptographic checks because only XCRC checks are supported.



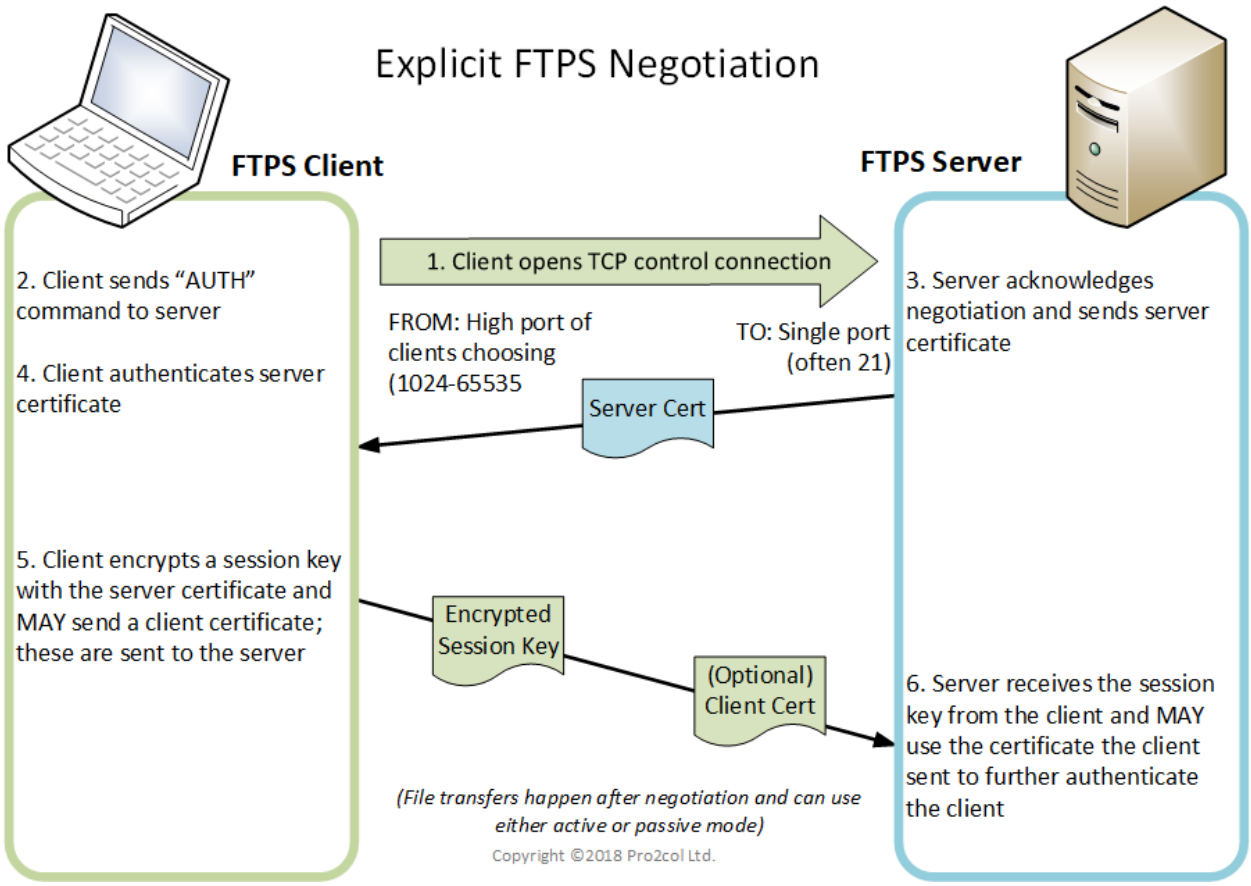
FTPS (SSL/TLS)

FTPS is the secure implementation of FTP. FTPS uses the same control-and-data channel mechanism used by FTP, but offers certificate-based PKI to encrypt both channels with SSL/TLS to secure authentication (especially passwords), commands and data in motion. SSL/TLS is, of course, the same technology used to secure HTTP as HTTPS.

FTPS "Explicit" or "RFC-Compliant" Mode

FTPS requires the use of one of two different secure negotiation modes, called "explicit" and "implicit," when an FTPS client establishes its control channel connection to an FTPS server.

In explicit mode, an FTP client connects its control channel to the normal FTP port (usually TCP 21), passes negotiation commands in cleartext, and then establishes an SSL/TLS connection. Explicit mode is also known as "RFC compliant" mode because there is an official FTPS RFC (RFC 2228), and the RFC only specifies how to set up explicit connections (not implicit connections).

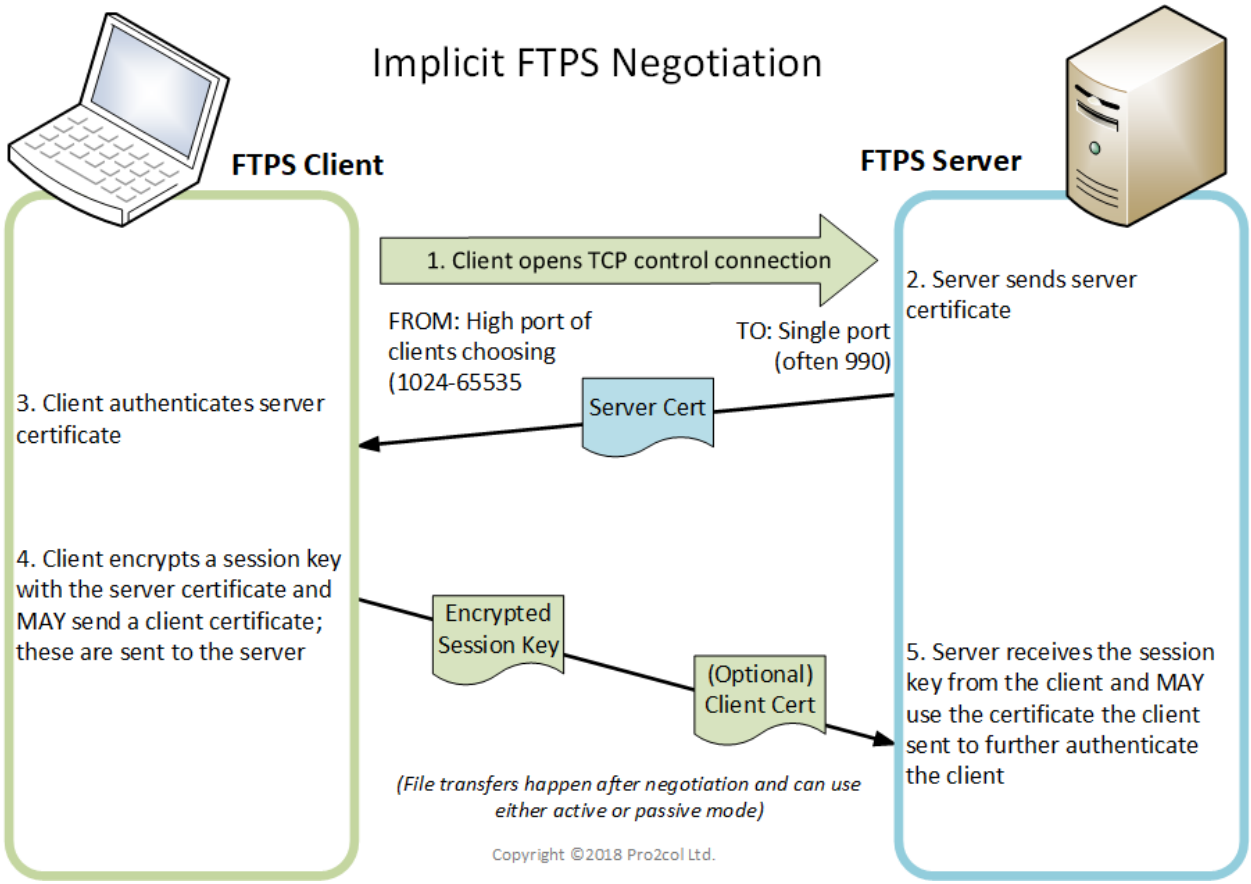


FTPS "Implicit" Mode

FTPS implicit mode works like HTTPS. In HTTP/S it is common to use separate TCP ports for insecure HTTP connections (often port 80) and secure HTTPS connections (often port 443). In implicit FTP, separate TCP ports are used for insecure FTP connections (often port 21) and secure FTPS connections (often port 990). As with HTTPS, implicit FTP channel encryption automatically happens right after the connection succeeds.

Since none of the encryption negotiation takes place using ordinary cleartext FTP commands, implicit FTPS can be easier to use, but this popular variant is not universally supported because it violates the official FTPS RFC (RFC 2228).

Note that it is still necessary to use a second channel for passing of data (transfers, directory listings)



FTPS vs. Firewalls and NAT

FTP was developed before firewalls and NAT were common, and many firewalls still have problems handling FTPS. Many firewalls automatically handle cleartext FTP connections by snooping FTP's unencrypted control channel to translate internal/external IP addresses and open data ports on the fly. However, firewalls' FTP snooping features are useless when FTPS encrypts its command channel.

To make FTPS work in firewall and NAT'ed environments, a variety of features are now common in modern FTPS implementations. Most FTPS servers implement at least one of these features, and many FTPS servers implement all four. FTPS servers that implementing more of these features than others may be better choices because they will work with more FTPS clients across multiple firewalls.

- **"Require passive and require a fixed range of passive ports"** - a combination of settings designed to force clients to always connect into the server ("require passive") and allow firewall technicians to keep the same, small range of high ports (e.g., "5000-5005") open from the Internet to the FTP server. Most successful FTPS implementations use this configuration, and it is by far the most popular way FTPS server administrators deal with firewalls and NAT routing.

- **CCC ("Clear Command Channel")** - a special FTPS command used to temporarily "un-encrypt" the command channel so an FTP-aware firewall can snoop the command channel. CCC is commonly supported but not often used in practice. *(Note that some professionals and auditors shy away from CCC because it "lowers the shields" on a previously encrypted FTPS control channel.)*

- **"Inject external IP address into data channel responses"** - a common configuration used to replace unreachable internal IP addresses with external IP addresses in the server's responses to passive data channel negotiation. This type of configuration also often requires an administrator to define groups of internal and external addresses by IP and/or hostname so the right address is returned to each connecting FTPS client.

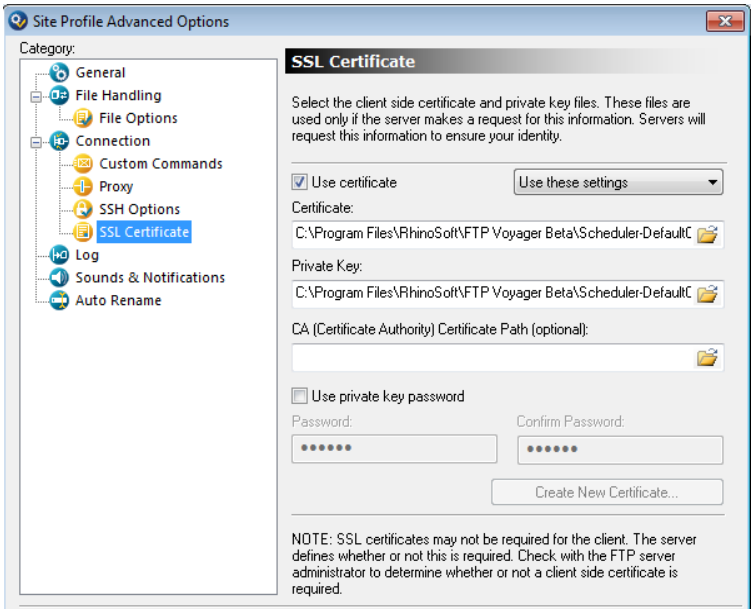
Strong Authentication (via Certificates)

FTPS supports strong authentication using X.509 certificates.

Server authentication allows FTPS clients to confirm that an FTPS server is what it claims to be. Since FTPS uses X.509 certificates, which are the same type of certificates used in HTTPS, server certificates can either be individually trusted (by matching a known server certificate signature), or trusted through a certificate authority (because the client trusts certificates signed by that CA). FTPS clients that use both models by default are common, as are options to "allow any cert" or otherwise ignore the certificate any particular server presents.

Client authentication allows FTPS clients to sign in to the server using an X.509 certificate the server trusts and has been previously associated with the related account. When client certificates are used along with passwords to authenticate users, it is called "two factor authentication," and it provides a very high level of security.

The PKI (Public-Key Infrastructure) Concepts section describes which parts of which certificate are shared with each side of the negotiation, but it is important to remember that each side only ever shares its certificate's public key, and never shares its private key.



Example of an FTPS client certificate authentication configuration.

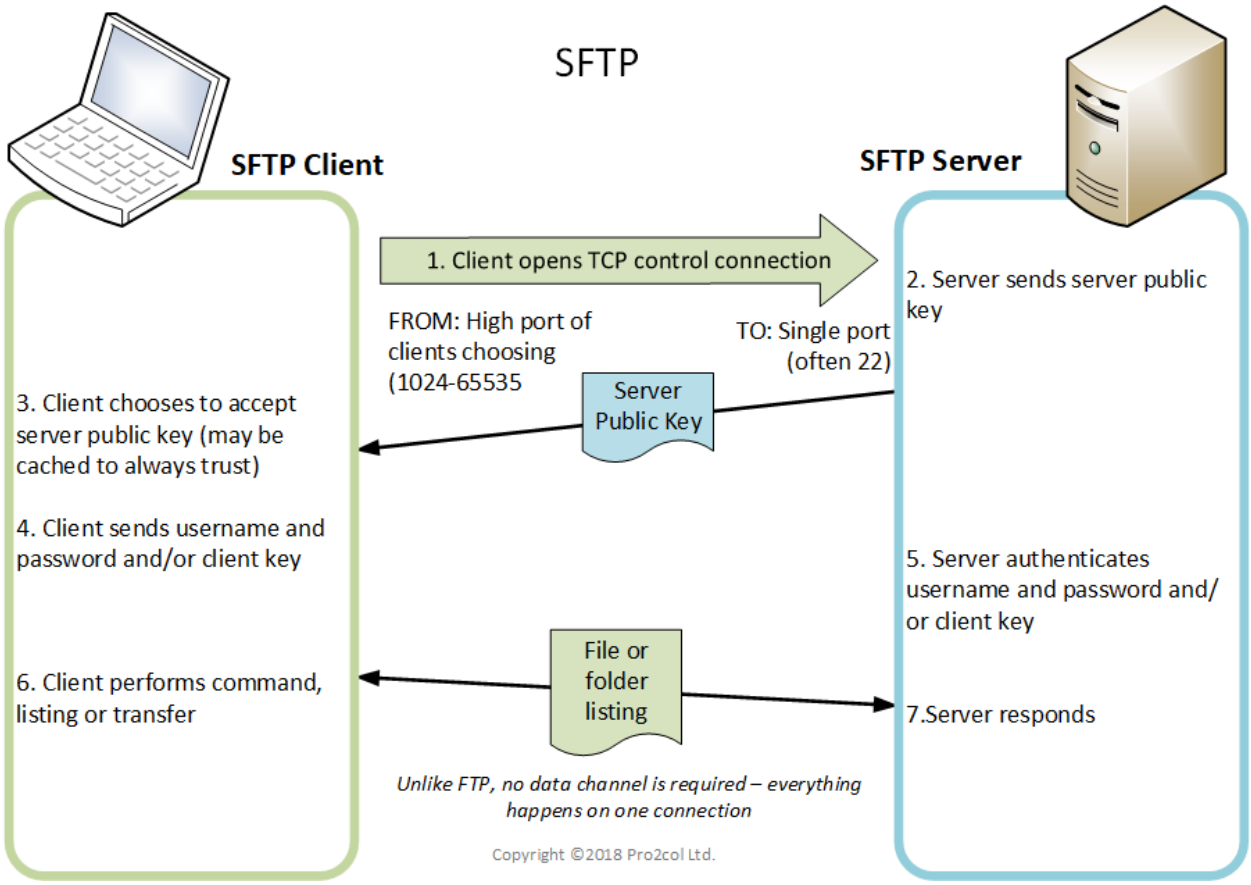
In the example above, the selected certificate comes in two parts: a certificate (or "public key") and private key. For any client certificate authentication configuration to work:

- The server must have this user's certificate (public key) on file, or the user's certificate must have been signed by a certificate authority that the server trusts.
- This user's certificate must be associated with the user's account, or enough information can be gleaned from the user's certificate (e.g., a username in the "CN" field) that a reliable certificate-to-account mapping can be performed on the fly.

SFTP (SSH)

SFTP is a file transfer protocol built on top of and tightly integrated with SSH. It secures data in transit by encrypting the entire session, including credentials, commands and data. SFTP is implemented natively on most *nix-based operating systems, including Mac OS X, and many FTP clients, especially "third party" FTP clients not shipped with an operating system.

Unlike FTP/S, SFTP uses just one TCP connection for data and commands (often on TCP port 22). This makes SFTP a very attractive, "single port" protocol when firewalls are involved. For convenience sake, most commands used in SFTP closely resemble their FTP equivalent, however they are two quite different protocols.



Like FTP/S, SFTP is also supported over both IPv4 and IPv6.

SFTP vs. "FTP Tunneled Over SSH"

SFTP is completely different from "FTP tunneled over SSH" and the two protocols are completely incompatible with each other. When FTP is "tunneled" over SSH, SSH is basically being used as a VPN to securely tunnel full FTP sessions, including FTP's multiple control and data connections. As a fully integrated SSH protocol SFTP avoids this complexity and overhead, and should be preferred over FTP tunneled over SSH in almost any managed file transfer application.

Strong Authentication (via Keys)

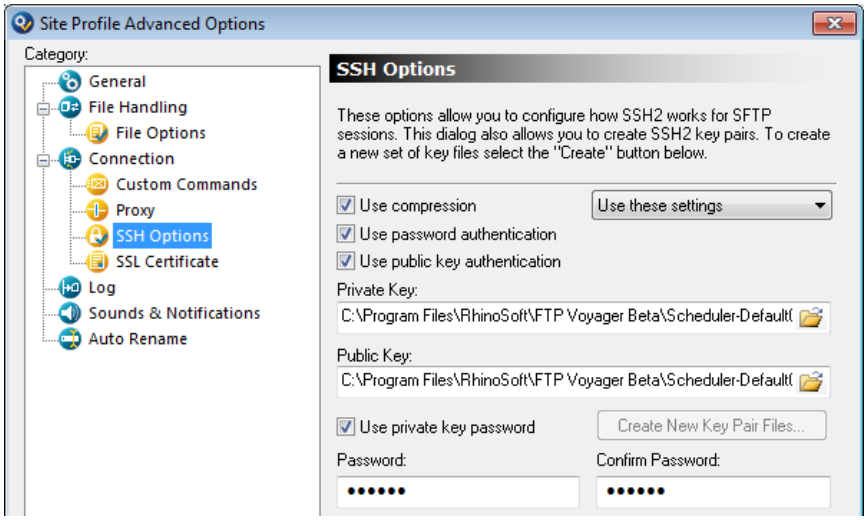
Like SSL/TLS, SSH depends on authentication and encryption based on an initial public/private key exchange and validation. Unlike SSL/TLS, SSH usually uses cryptographic keys rather than the full infrastructure of X.509 certificates or a trusted chain of certificate authorities.

Server authentication allows SFTP clients to confirm that an SFTP server is what it claims to be. Since SFTP typically uses keys rather than certificates, server keys are usually individually trusted by matching a known server key signature. Usually, this is accomplished through a one-time prompt on the client that allows end users to confirm or deny the "thumbprint" hash of the server's key. Some SFTP clients can also ignore server keys.

Client authentication allows SFTP clients to sign in to the server using a key which has been previously associated with the related account. When client keys are used along with passwords to authenticate users, it is called "two factor authentication," and it provides a very high level of security.

The PKI (Public-Key Infrastructure) Concepts section describes which parts of which key are shared with each side of the negotiation, but it is important to remember that each side only ever shares its certificate's public key, and never shares its private key.

SFTP is very popular on *nix machines because a full SSH is usually included with each version of *nix. Furthermore, *nix SFTP servers are often configured to allow SFTP clients to skip password authentication if client key authentication succeeds because *nix SFTP scripts often do not allow passwords, but will allow references to client key files.



Example of an SFTP client certificate authentication configuration.
Note the option to "use password authentication" (or not).

SCP (SSH)

Like *nix's RCP ("remote copy") command, SCP ("secure copy") is a "single command file transfer" utility most commonly found on *nix systems. The SCP/RCP relationship parallels the SFTP/FTP relationship: both applications provide the look and feel of their non-secure counterpoints with SSH underpinnings.

Copy the directory "foo" from the local host to a remote host's directory "bar"

```
$ scp -r foo your_username@remotehost.edu:/some/remote/directory/bar
```

Copy the file "foobar.txt" from remote host "rh1.edu" to remote host "rh2.edu"

```
$ scp your_username@rh1.edu:/some/remote/directory/foobar.txt \
your_username@rh2.edu:/some/remote/directory/
```

SCP examples from http://www.hypexr.org/linux_scp_help.php

Since SFTP does everything that SCP can do and more, SCP is often only useful in situations where a single command-line command must be used and a single-command SFTP script is not desired. Many file transfer implementations omit SCP for this reason. However, for scripted workflows or applications that cannot execute scripts, the single line simplicity of SCP does still provide a working solution (but must be considered against its disadvanteages.)

IV. Advanced File Transfer Protocols

HTTP

HTTP ("HyperText Transfer Protocol") is an insecure protocol used to transfer data between web browsers and web servers. With the rise of the World Wide Web, HTTP became the world's most popular file transfer protocol, even though almost all HTTP file transfers involve downloads of web pages, images, and other files rather than uploads.

HTTP is often bound to TCP port 80, and this is the port web browsers use by default when an "http://…" request is made on any web browser. (Other HTTP ports can be accessed from a web browser by appending a different port after the hostname and a colon.)

HTTP URLs

HTTP depends on URLs ("Universal Resource Locators") that consist of several well-defined components:

- Protocol (e.g., "http://")
- Hostname (e.g., "www.acme.com")
- *Optional* Port (e.g., ":8080")
- *Optional* Path (e.g., "/folder/file")
- *Optional* Query String (e.g., "?arg1=one&arg2=two")

Valid examples of HTTP URLs include:

- http://www.acme.com:8080/folder/file?arg1=one&arg2=two
- http://www.acme.com
- http://www.acme.com?arg1=one&arg2=two

...and many more combinations

Originally, the "path" component of the URL was used to download a specific file from a specific location, and many URLs are still used to do this. For example, "http://www.acme.com/stylesheets/main.css" might be used to download the existing "c:\inetpub\wwwroot\stylesheets\main.css" file from the "www.acme.com" web server, whose root content folder is "c:\inetpub\wwwroot."

Modern web applications often use path variables in a different way, and can generate files on the fly. For example, "http://www.acme.com/reports/budget" might be used to generate an XML file that contains the current budget report, even though there are no "reports" or "budget" folders on the web server.

"Non-URL" HTTP Parameters

In addition to the URL, HTTP includes the ability to support "verbs" (e.g., "GET", "POST" and "OPTIONS"), "headers", "cookies" and other parameters that can directly affect the behavior of an application or a particular request. Form-based "POST" submissions are often used to upload files from web browsers. Advanced uploading functions suitable for large files, integrity checks or multiple file uploads often use client-side JavaScript or applets developed for Java, ActiveX or Flash runtimes.

HTTP Security

The complexity and power of HTTP (and HTTPS) makes it susceptible to its own class of security vulnerabilities, including SQL injection, cross-site scripting and path traversal. Any file transfer application using HTTP should be inspected or come with assurances that it has been inspected for common security vulnerabilities before being deployed in production.

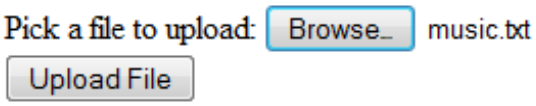
HTTP File Uploads

The HTTP protocol supports many types of data transfer and file transfers may be performed using several HTTP methods. Most HTTP file uploads are initiated using a URL or a form using the "POST" command. However, the most common way HTTP file uploads are performed is through a web page "multipart/form-data" form and a special "file" input field.

For example, the following web page snippet (in HyperText Markup Language or "HTML"):

```
<form action="/upload_file" method="post" enctype="multipart/form-data">
  Pick a file to upload:
  <input type="file" name="file" id="file"><br>
  <input type="submit" name="submit" value="Upload File">
</form>
```

…yields a simple upload form when rendered in a web browser.



However, there are thousands of different ways that a web application could receive and process a file from a form like this. Some web applications allow users to upload files into a directory tree (like an FTP client), but many others guide uploads into specific folders, a selection of queues, or directly into a processing routine.

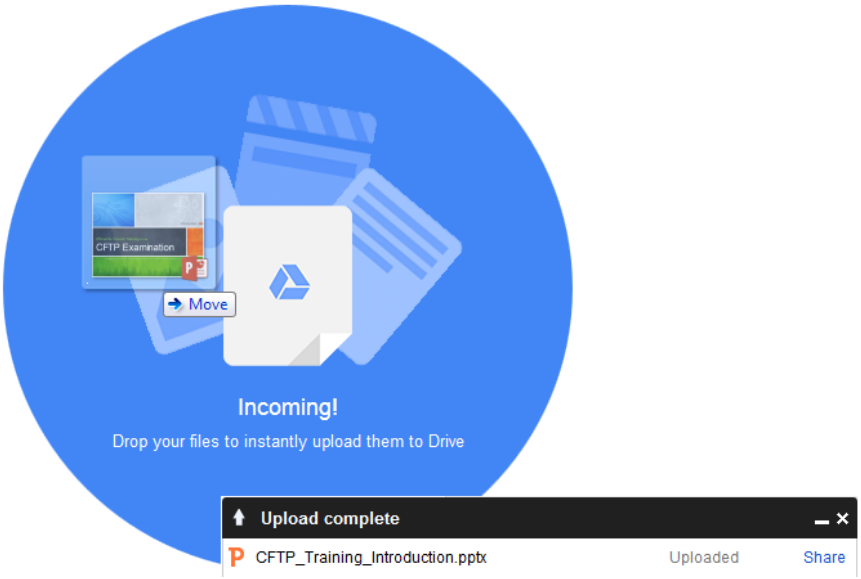
The use of specialized HTTP transfer formats (e.g., "chunked encoding") and the existence of multiple HTTP-based transfer protocols *(including WebDAV and AS2 – both covered below)* also makes it difficult to determine whether an HTTP service allows file uploads without requiring a managed file transfer application's web interface itself.

HTTP Advanced File Uploads

Advanced file uploads are made possible through special web-based file transfer clients provided through the web interfaces of managed file transfer software. Features in these special clients frequently include:

- drag-and-drop
- resuming broken file transfers
- multiple file transfers
- folder transfers
- integrity checks
- support for file uploads larger than 2GB or 4GB (which are common browser upload limits)

First-generation web-based file transfer clients often depended on client-side installations of Flash, Java or Active-X. Current clients typically make use of JavaScript and HTML5 features that are built in to modern browsers. A typical advanced file transfer client is the one used in Google Docs to intercept and upload files “dragged and dropped” on the browser.



Despite the similarity of features between managed file transfer software and cloud solutions, every providers’ implementation is likely to be different from and incompatible with any other provider’s solution. However, as long as end users are protected from having to explicitly install software or plug-ins, the use of unique clients in each web interface rarely creates problems because the special clients appear to be a seamless part of the overall user experience.

HTTPS (SSL/TLS)

HTTPS is HTTP secured with SSL ("secure socket layer") or TLS ("transport layer security"). It behaves much like HTTP and supports the same URL syntax.

HTTPS is generally bound to TCP port 443, and this is the port web browsers use by default when an "https://…" request is made. Other HTTPS ports can be accessed by appending a different port after the hostname and a colon.

HTTPS uses PKI to:

- allow connecting clients to validate the identity of a web server
- allow client and server to securely exchange a symmetric key to secure data transferred between client and server
- optionally, allow web servers to authenticate the connecting user using their optional client certificate

The establishment of a new HTTPS session is a relatively expensive operation because it involves processor-intensive asymmetric encryption to perform PKI operations. However, once an HTTPS session is established, and client and server have agreed on a symmetric key, HTTPS switches to symmetric encryption so the actual data transfer occurs rapidly and efficiently.

Communicating Certificate and Key Strength

The combination of asymmetric encryption and symmetric encryption in HTTPS leads to sometimes confusing security statements such as "2048-bit certificates used to secure a 256-bit TLS session."

Large numbers such as "1024-bit," "2048-bit" and "4096-bit" describe the size of the public and private keys used by the certificates when negotiating SSL and TLS sessions. Smaller numbers such as "128-bit" and "256-bit" describe the key length of the negotiated symmetric keys.

The asymmetric keys used in PKI and HTTPS negotiation must be much longer than symmetric keys of the same strength because any number can be a symmetric key, but only a few special numbers can be used as asymmetric keys. A common rule of thumb in SSL/TLS applications is that the asymmetric certificate key length should be about 10 times as large as the symmetric session key (e.g., "a 1024-bit cert leading to a 128-bit session, or a 2048-bit cert leading to a 256-bit session".)

Strong Authentication (via Certificates)

Web browsers and other HTTPS clients always have the opportunity to validate the identity of an HTTPS server by inspecting its server certificate. Each server certificate is an X.509 certificate and the server certificate is also often signed by a trusted Certificate Authority (CA). Web browsers usually have a built-in list of trusted CAs and will automatically warn end users if an HTTPS site does not offer a certificate signed by one of its trusted CAs.

Web browsers and other HTTPS clients also allow web servers to authenticate by providing their own client certificate. These client certificates are also X.509 certificates and are also often signed by a trusted Certificate Authority (CA). HTTPS web servers often perform the basic negotiation necessary to receive information about client certificates, but often pass the details on to specific web applications which make the final determination to either accept or deny a particular client request. The list of authorized client certificate CAs is also more tightly controlled on HTTPS servers than it is on web browsers. For example, a company's internal CA, a single commercial CA or a trusted government CA, might be designated to sign all client certificates accepted by a particular web application.

SSL Versions and TLS

There have been several versions of SSL used to secure HTTPS over the years. SSL version 1 has essentially been retired, but SSL versions 2 and 3 are still quite common. Today, all versions of SSL are considered as insecure and should be avoided.

The modern replacement for SSL is TLS ("transport layer security"), and there are several versions of TLS including 1.0, 1.1 and 1.2. Most modern "SSL" implementations actually attempt to negotiate TLS sessions in place of SSL, but HTTPS is still often considered to be "secured by SSL", whether or not TLS is actually negotiated.

In 2014, it was determined that the "Poodle" vulnerability affected SSL v3.0, TLS v1.0 and TLS v1.1, leading many companies to permit only TLS v1.2 to be used on Internet-facing sites.

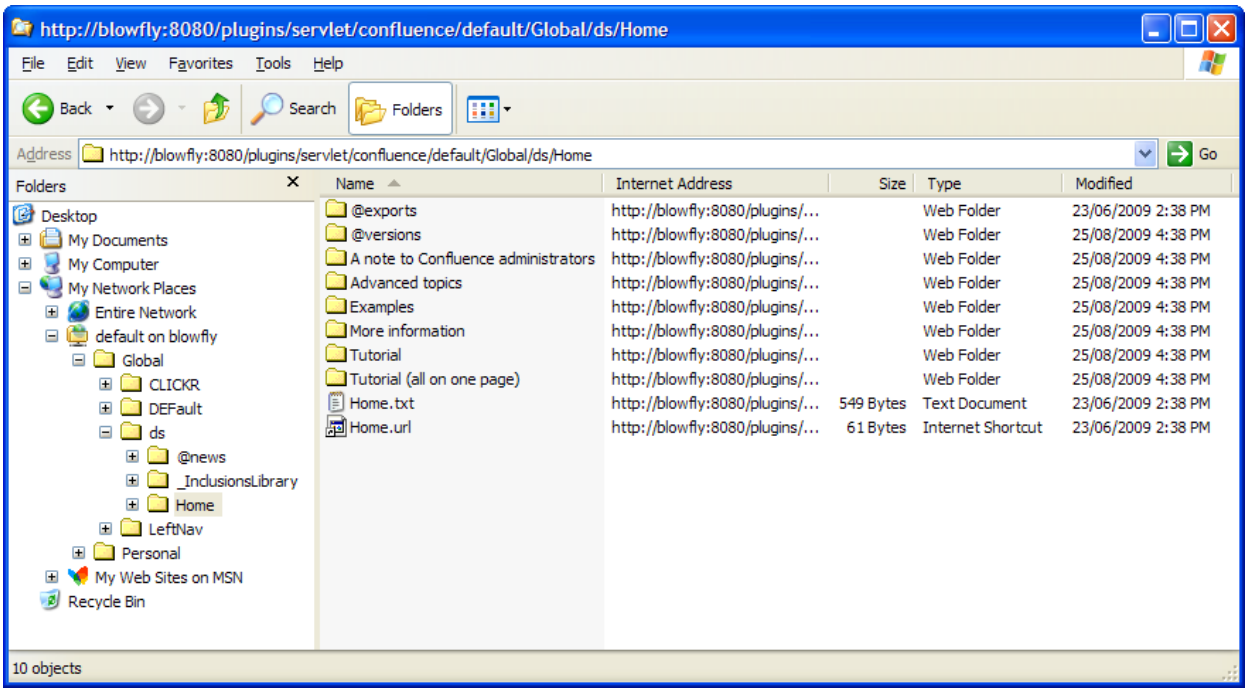
Additionally, when deciding which SSL/TLS versions to disable, you must also disable weak ciphers.

WebDAV

WebDAV (*Web Distributed Authoring and Versioning*) is a vendor-independent framework that extends the HTTP protocol to allow a variety of clients to upload and download files with a file transfer server. A native WebDAV server is available on Windows servers and is readily available for Linux and Mac servers.

In addition to common HTTP commands such as “GET” and “POST”, WebDAV makes use of “extended” HTTP commands such as “COPY” and “MOVE”. These commands make it easy for web proxies to track or prevent WebDAV commands. WebDAV also allows locking of files and management of properties / metadata, making it a good choice as the basis of a collaboration platform.

WebDAV’s advantages over other file transfer protocols include the use of a single port (either HTTP or HTTPS) to perform all commands, and the wide range of native clients on desktops that can easily “map a drive” to a WebDAV server to make it look like a local resource.



WebDAV server mapped into Windows Explorer. *(From Atlassian documentation.)*

However, WebDAV’s popularity has been hindered by initial fears of insecurity (early versions of Microsoft IIS contained several severe WebDAV vulnerabilities), slower performance when compared to other protocols and recently, reluctance of file transfer vendors to include WebDAV as a supported protocol (while offering proprietary APIs or web services instead).

Email Protocols

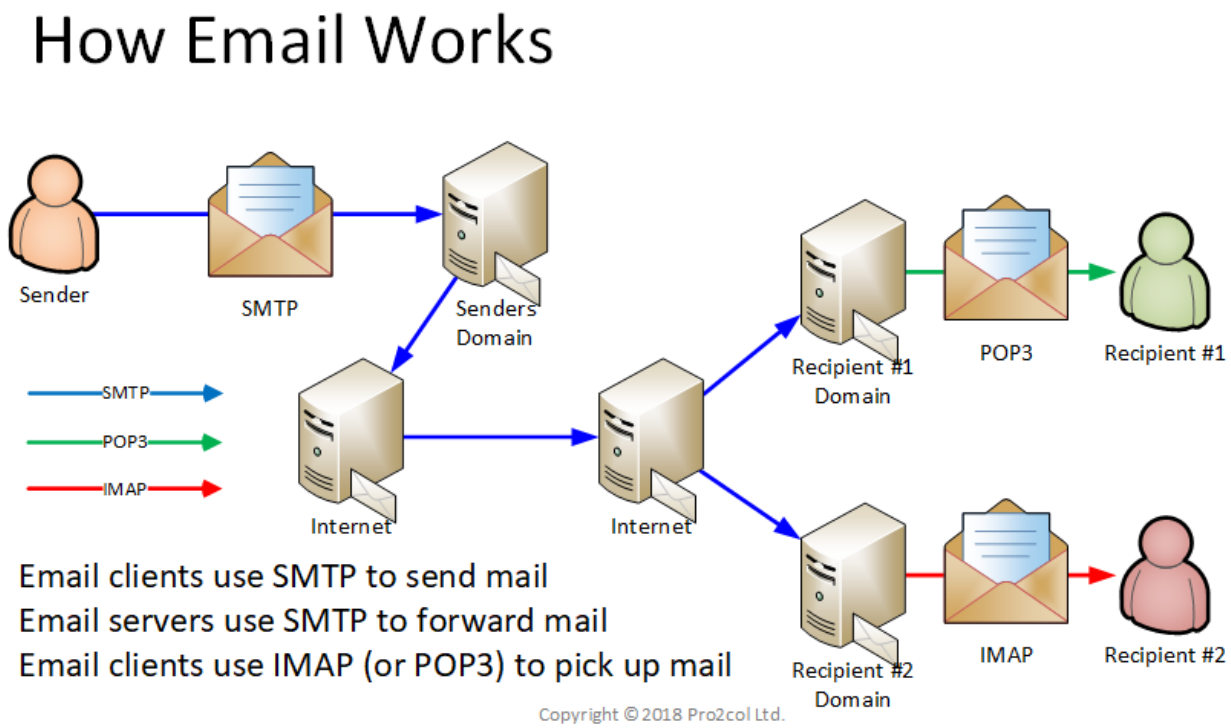
It is important for file transfer professionals to understand how common mail protocols are used in file transfer operations for several reasons:

- notifications are often sent via email
- files are sometimes delivered as attachments to mail messages
- file transfer processes are often introduced to replace files manually sent as attachments
- all of the AS* protocols (including AS2) are built on email concepts

Email messages, including all formatting and attachments, are “encoded” (but not necessarily encrypted) using MIME. They are usually sent to the local email server using SMTP. From there they are usually:

- Picked up by a local recipient if the message was sent to a local user
- Sent directly to the recipient’s mail server using SMTP if the sender’s mail server
- Forwarded (or “relayed”) to the recipient’s mail server using SMTP through one or more intermediate mail servers on the Internet (if the sender’s mail server cannot directly connect or send messages to the recipient’s email server)

Email messages are usually picked up using POP3 or IMAP. IMAP is generally preferred when synchronization of email folders between client and server is desired, or a client wants to act on message headers rather than an entire message. More recently, IMAP has been used to facilitate “push email,” in which email clients are notified of new messages in near real time. POP3 is sometimes preferred when a simpler protocol is desired.



Some tightly coupled email client/server combinations, such as Microsoft Office Outlook and Microsoft Exchange, use additional protocols to send mail and receive mail. However, almost every email client and email server will support three common interoperable protocols: SMTP to send messages and POP3 and/or IMAP to receive messages.

SMTP

SMTP ("Simple Mail Transport Protocol") is an extremely common email protocol used to push email messages and their attachments to an email server. SMTP is also used by individual mail servers to forward mail messages to other servers.

SMTP is often secured with SSL/TLS. As with FTPS's "explicit" and "implicit" modes, the two most common implementations differ in terms of whether they attempt to establish the secure connection immediately upon connection to a special port (often TCP 465 for "SMTP with SSL/TLS") or after the connection is made to the usual port (often TCP 25 for optional STARTTLS or TCP 587 for required STARTTLS).

Some file transfer software however cannot send secure emails; in this case a solution is to create a local SMTP relay on the server hosting the software. In general, the relay can be protected from misuse by restricting access to it to just localhost, and the relay can then be configured to forward the mail using SSL/TLS.

POP3 ("POP")

POP ("Post Office Protocol") is a common but older email protocol used to download email messages and their attachments from an email server. The current version of POP is version 3, and it has been the current version for so long that the POP protocol is now more commonly referred as "POP3."

POP3 can delete original email messages from email servers but cannot completely synchronize folder content.

POP3 can be secured with SSL/TLS and usually uses a special port (TCP 995) to secure itself with SSL/TLS immediately upon connection. Despite its utility, POP3 is often considered obsolete and many email clients (including the default email client in Windows 8) no longer support it. Many servers will also have POP3 access disabled in favor of IMAP.

IMAP

IMAP ("Internet Message Access Protocol") is a very common email protocol used to download email messages and their attachments from an email server. The current version of IMAP is version 4, but the protocol name and version number are not often combined as they are with "POP3" (e.g., IMAP version 4 is usually just called "IMAP," not "IMAP4").

IMAP can delete original email messages from email servers and can completely synchronize folder content. However, it should be used with care in environments where end users can also access the same account's mailbox interactively, as end users may move messages before they can be processed by your file transfer applications. With this in mind, you should set up dedicated email accounts for your file transfer applications, and send copies of emails to those accounts rather than tie automated processes to accounts shared with active end users.

In most email applications, IMAP has replaced POP3.

IMAP can be secured with SSL/TLS and usually uses a special port (TCP 993) to secure itself with SSL/TLS immediately upon connection.

Mail Protocol Port Summary

The following TCP ports are most commonly used to support common mail protocols.

Protocol	Unencrypted	SSL/TLS using "STARTLS"	SSL/TLS upon connection
SMTP	25[1]	587	465
POP3	110	*n/a*	995
IMAP	143	*n/a*	993

For even more information about email protocols, we recommend an excellent article that covers the three main protocols and their use of encryption written by FastMail Pty Ltd for the documentation of their FastMail product. https://www.fastmail.fm/help/technology_ssl_vs_tls_starttls.html

[1] STARTTLS is often supported on port 25, but it is usually required on port 587

NDM ("Connect:Direct")

NDM stands for "Network Data Mover" and is a proprietary file transfer protocol used in Sterling Commerce's Connect:Direct ("C:D") Connect:Enterprise ("C:E") and other products. Sterling Commerce was acquired by IBM and some of the product names have changed, but the link between "NDM" and "Connect:Direct" is as strong as ever.

All of the other protocols in this guide are open protocols that are supported by hundreds or thousands of software vendors and open source projects, but NDM is different because it is the de facto backbone of bulk data transfers between many key financial and commercial entities, including the banking industry and components of the United States government.

NDM provides reliable file transfer, including "checkpoint restart", and can be secured with modern versions of Connect:Direct and its companion software. At a minimum, Connect:Direct requires both a sending and receiving server, both with the Connect:Direct software installed.

NDM has been reverse-engineered, but independent implementations do not exist because of intellectual property challenges involved in developing and maintaining these versions in commercial environments.

Genesis of MFT

Before there was an established "managed file transfer" community, there were FTP servers, FTP clients, encryption packages and automation technologies. In the early 2000's a chorus of unhappy NDM users began to demand features like "checkpoint restart", "auditing" and "automation integrated with my mainframe processes" from providers of open protocols.

The result was a flowering of new software solutions that provided these advanced features using extensions to FTP, enhancements to SFTP, new HTTP interfaces, new web services and AS2. A few years later Gartner's L. Frank Kenney rebranded the industry as "managed file transfer" (or "MFT") and published several Gartner Magic Quadrants covering the space.

Today, a frequent request on RFPs and other inquiries from MFT prospects is "handling NDM or Connect:Direct." In many cases a straight replacement would be organizationally infeasible, so many experienced MFT vendors now preach a message of coexistence with NDM, rather than outright replacement.

V. Applicability Statement (AS*) Protocols

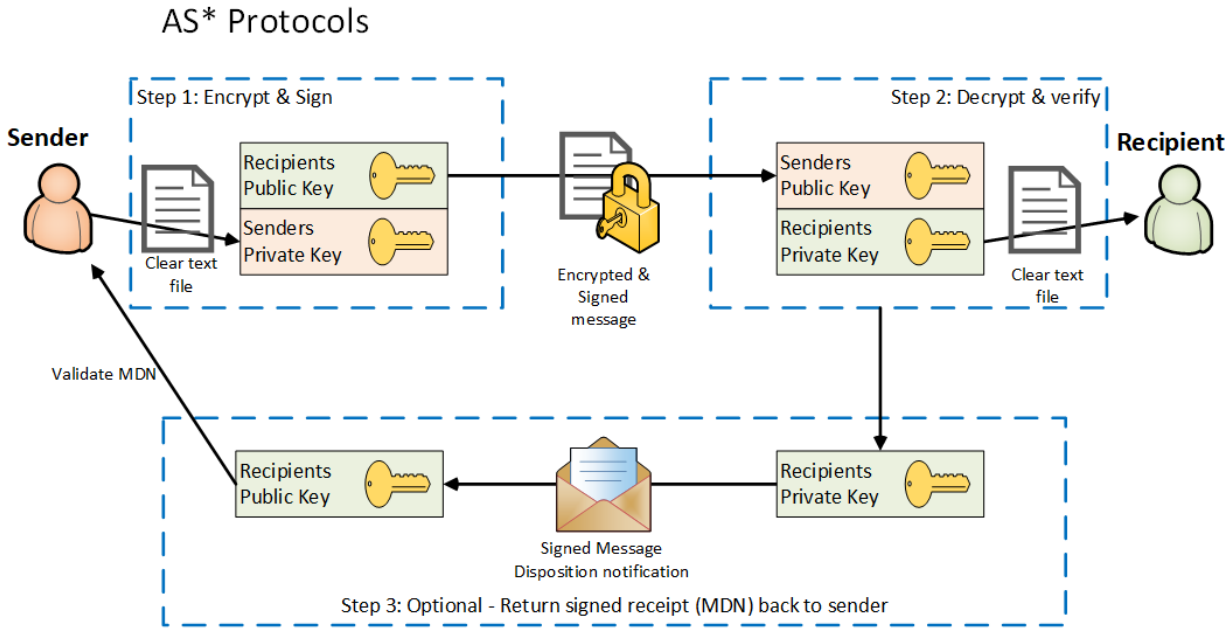
The “Applicability Statement” protocols (“AS*”) are a family of advanced file transfer protocols that include built-in integrity checks, sender authentication, non-repudiation, file encryption, transport encryption and, in later versions, checkpoint restart features. They are all built on PKI concepts, particularly the use of SMIME to encrypt and sign data. Their use is very common when Electronic Data Interchange (“EDI”; the use of structured documents and responses to exchange information) is used.

There are three general steps in any AS* transfer:

1. A cleartext file is encrypted using the recipient’s public key and signed with the sender’s private key. This yields an AS* message, which different AS* protocols transmit using different protocols.

2. The message is received and verified using the sender’s public key. It is then decrypted using the recipient’s private key, yielding the original cleartext file.

3. If the sender has requested that the recipient verify the receipt of the cleartext file, a digital receipt called an “MDN” (“Message Disposition Notification”) is created using a hash (“integrity check”) of the received file, signed with the recipient’s private key, and sent back to the sender using the protocol related to the specific type of AS* in use. The sender verifies the signature on the MDN using the recipient’s public key and, if all checks out, now has cryptographic proof that the recipient is now in possession of a copy of the original data. (This is also known as “non-repudiation” because the recipient cannot deny receipt of the data once the MDN is returned and verified.)

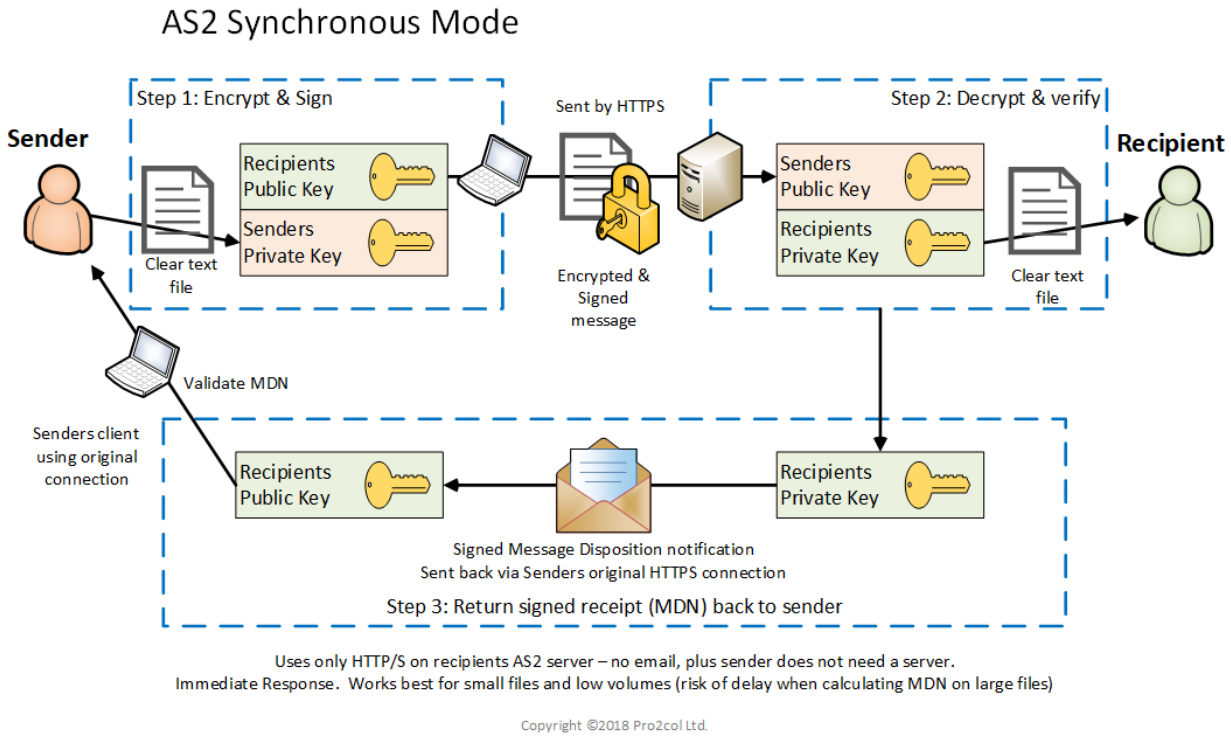


AS2 ("Applicability Statement 2")

The AS2 protocol always uses HTTP or HTTPS to transmit data, and data recipients must always have an AS2 server. However, there are three different ways that MDNs can be returned using AS2: via a HTTP/S post back to the sender's AS2 server, via the sender's original HTTP/S connection, or via email. These three MDN methods are described in detail below.

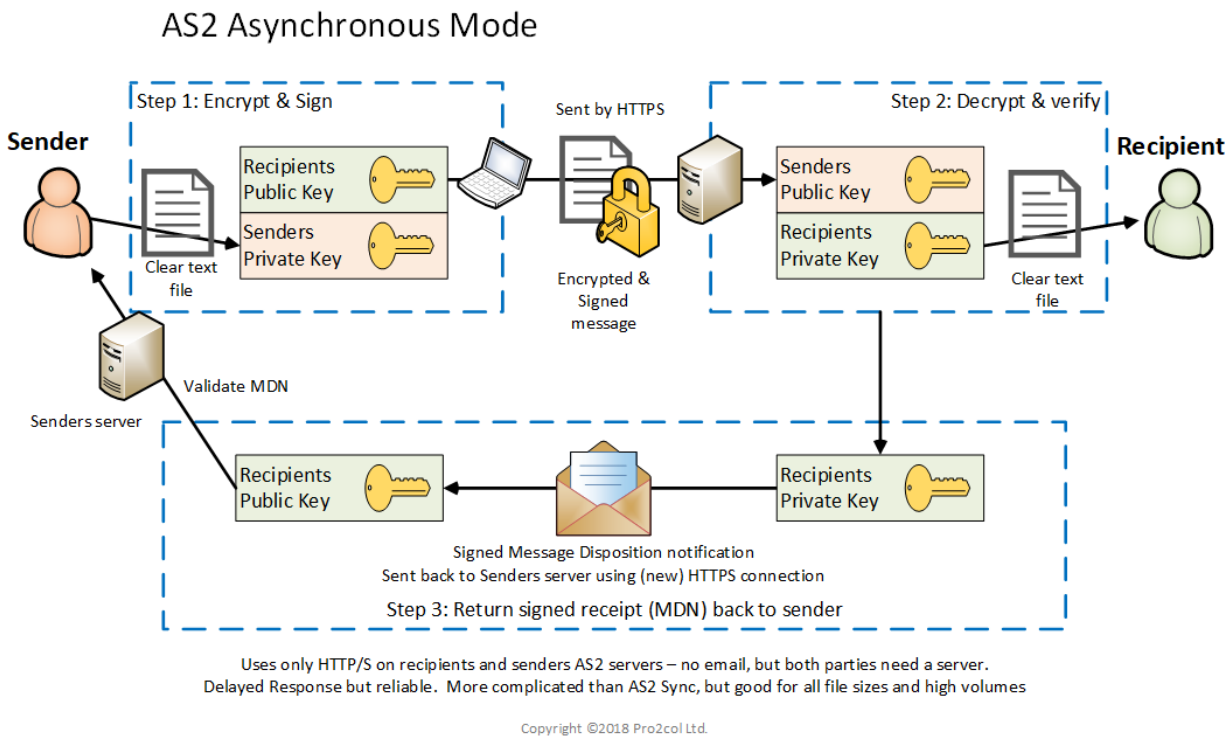
AS2 "Sync" (Synchronous) Mode

AS2 "Sync" (synchronous) mode is the most popular of all AS2 modes because it provides a near real-time MDN response using a single, firewall friendly HTTP/S connection. In AS2 "Sync" mode, a message is posted via HTTP/S, and the MDN is calculated and sent back over the same session after a small delay. Unlike AS2 "Async" mode *(next page)*, this mode does not require the sender have an AS2 server. However, the time required to process the received message and compose the MDN limits the size of files that AS2 "Sync" mode can comfortably support. (This is often not a limitation in an efficient EDI environment.)



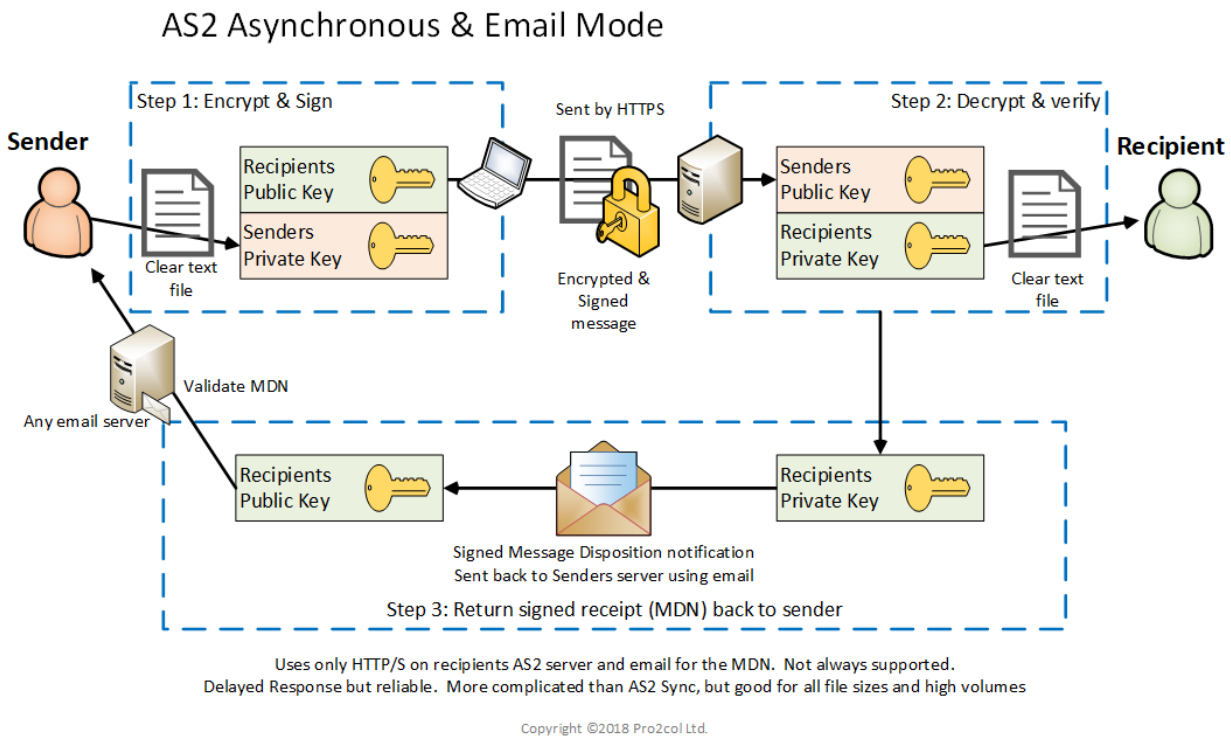
AS2 “Async” (Asynchronous) Mode

AS2 “Async” (Asynchronous) mode uses a second HTTP/S connection to post MDNs back to the sender’s AS2 server some time after the message is processed. AS2 “Async” only works if the recipient and the sender have both set up AS2 servers, but it is a reliable method to use when sending large files because the recipient can take as long as it needs to process the incoming data and send a response.



AS2 "Async + Email" Mode

Like AS1 *(see next page)*, AS2 "Async + Email" mode uses email to post MDNs back to the sender some time after the message is processed. Unlike the regular AS2 "Async" mode, this mode does not require the sender have an AS2 server. Like AS2 "Async" mode, this mode is also is a reliable method to use when sending large files. However, it is not supported by all AS2 clients.

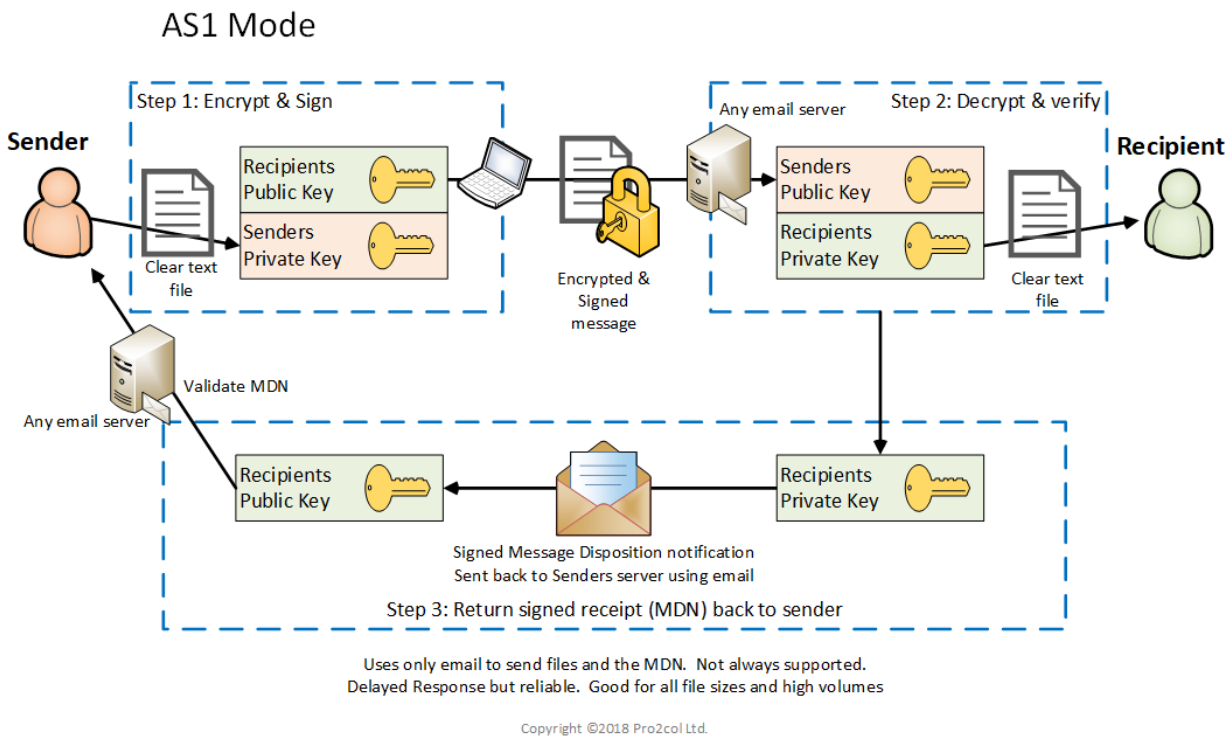


Other AS* Protocols

Other (non-AS2) Applicable Standard ("AS*") protocols provide similar benefits to AS2, but depend on different protocols (SMTP for AS1 and FTP/S for AS3), and do not offer near real-time responses like AS2's popular "synchronous" mode.

AS1

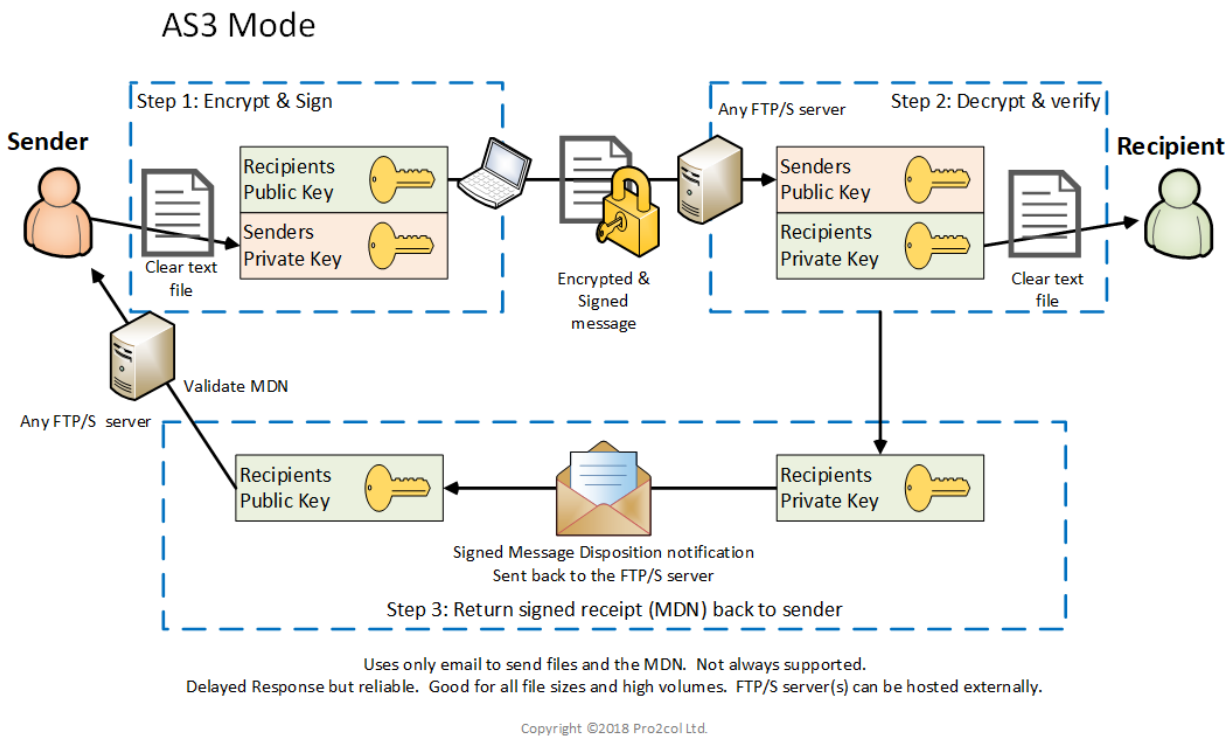
AS1 ("Applicability Statement #1") is similar to AS2 but uses SMTP (the popular email protocol) to send files and return MDNs. All AS1 operations are "asynchronous" - there is no "synchronous" mode in AS1 like there is in AS2.



Most normal email servers can be used to facilitate AS1 transfers: all of the encryption, decryption, signing and verification steps depend on the AS1 implementations of AS1 clients.

AS3

AS3 ("Applicability Statement #3") is similar to AS2 but uses FTP or FTPS (SSL/TLS) to send files and return MDNs. All AS3 operations are "asynchronous" - there is no "synchronous" mode in AS3 like there is in AS2.



Most FTP or FTPS servers can be used to facilitate AS3 transfers: all of the encryption, decryption, signing and verification steps depend on the AS3 implementations of AS3 clients.

AS4

The newest of the AS* protocols uses web services (ebMS 3.0 standards) and functions in a quite similar fashion to AS2. Unlike AS2 however, the actual message does not necessarily have to be an attachment – instead it can be the body of the AS2 message and may even be several messages sent together.

AS4 is more closely aligned with middleware than with MFT, but is included here for completeness. The main advantages of AS4 over AS2 are as follows:

- Pull as well as push messages
- Send multiple payloads in a single message
- Supports newer encryption algorithms
- Web Service support

AS4 take up has understandably been slow quite possibly due to the lack of vendor support in existing MFT products. A good example of an early take up of AS4 is IATA (International Air Transport Association)

AS* Selection Guide

Use the following guide to help select which AS* protocol is best for your use case. (When in doubt, remember that AS2 – both the "Sync" and "Async" varieties - is by far the most popular AS* protocol.)

	AS1	AS2 Async	AS2 Sync	AS2 Async + Email	AS3	AS4
Message Transport Protocol	SMTP	HTTP/S	HTTP/S	HTTP/S	FTP/S	HTTP/S
MDN Transport Protocol	SMTP	HTTP/S	HTTP/S	SMTP	FTP/S	HTTP/S
MDN Sent To	Any Mail Server	Sender's AS2 Server	Sender's Original HTTP/S Connection	Any Mail Server	Any FTP/S Server	Sender's Original HTTP/S Connection
Widely Supported	No	Yes	Yes	No	No	No
Supports Very Large Files	Yes	Yes	No	Yes	Yes	No
Near Real-Time MDN Response	No	No	Yes	No	No	Yes
Requires Sender to Host Own AS* Server	No	Yes	No	No	No	No
Allows grouping of messages into single packages	No	No	No	No	No	Yes
Can pull as well as push	No	No	No	No	No	Yes

Drummond Certification

An independent certification body called “Drummond Group” is the de facto champion and keeper of the AS* standards. If an implementation has been “Drummond Certified” than it has been at least partially tested for interoperability by the Drummond Group. *(Details about all implementations can be found on the Drummond Group’s site.)* Drummond certification was particularly important in the early days of AS2 because large retailers like Walmart specified the exclusive use of “Drummond Certified” software when connecting to their networks.

VI. Accelerated File Transfer

Accelerated File Transfer Basics

Accelerated file transfer techniques push large volumes of data across networks faster than traditional, one-TCP-connection-per-file methods used by FTP and HTTP can push data. The acceleration techniques used by accelerated file transfer solutions typically include:

- UDP packets instead of TCP connections (to avoid “TCP congestion”)
- Multiple transfer streams (“multi-threading”)
- Compression
- Terse (“less chatty”) protocols

Since many file transfer clients and servers support compression and multiple transfer streams when using more traditional file transfer protocols (such as HTTP and FTP), “use of UDP” is generally what sets accelerated file transfer apart from other types of file transfer. Most file transfer protocols are also considered terse, at least as compared to “chatty” protocols like NetBIOS/CIFS.

Multi-Threading

Some MFT products approach the problem by opening multiple TCP channels and splitting each file between the various channels. This relies on a client at the source to split the files, plus an intelligent server at the opposite end of the connection to reassemble the files again. Testing this approach shows very high speed transfers for larger files, however it incurs more overhead and therefore produces lower speeds when sending many smaller files.

TCP/IP Latency

From the perspective of accelerated file transfer solutions, TCP connections waste “extra” packets setting up each TCP connection (called the “TCP handshake”) and acknowledging every packet that comes across the wire. Waiting for these packets can be noticeable when data block sizes are low or there is a lot of distance between client and server. This is called TCP’s “latency” problem.

TCP connections are also linear: each block of data transferred over a connection follows the one sent before it.

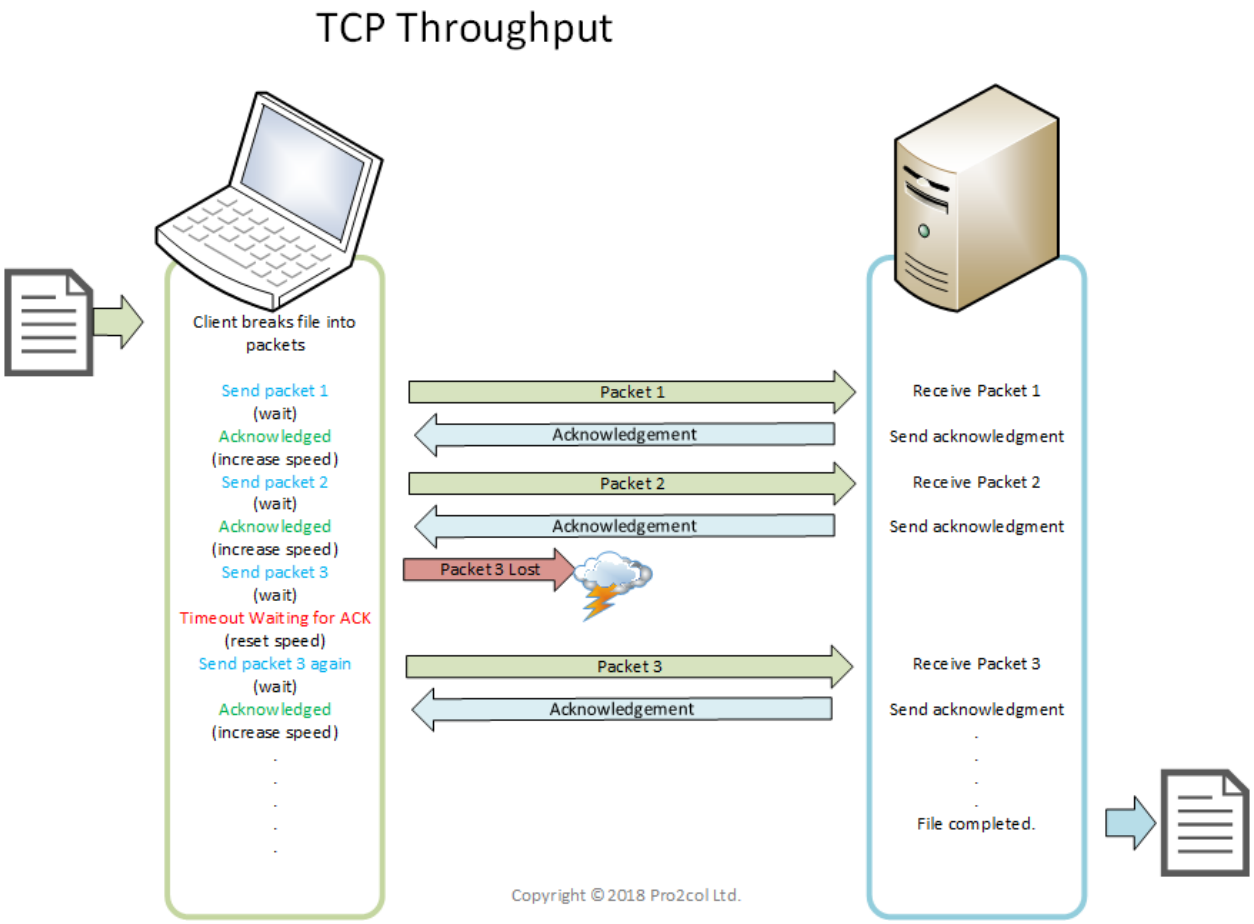
TCP/IP Throughput

Transfers via TCP have some basic limitations that many users aren’t aware of. TCP/IP uses a concept of a “Receive Window” (RWIN) which is a buffer to hold the incoming data packet prior to acknowledging receipt to the sender. This buffer was originally set at a maximum of 64 Kbytes; however extensions made to TCP mean that this limit can now be increased via TCP scaling to improve performance, although this has to be considered against the negative impact of potentially losing bigger packets.

Other principle factors affecting the throughput are the network bandwidth and the round trip time (obtainable from a ping command).

Assuming a round trip time of 68ms and the default RWIN of 64 Kbytes, the maximum throughput possible would be 7.71 Mbit/sec, regardless of whether it is a 100Mbit or 100Gbit connection. (A handy calculator for reaching this figure can be found at https://www.switch.ch/network/tools/tcp_throughput/)

Another consideration is the Bandwidth Display Product, or BDP. This is a value that can be determined from the same calculation and is used to determine the maximum amount of data residing on the line at any given time. Given a line speed of 100 Mbit/second and a Round Trip Time (RTT) of 68 ms, this results in a BDP of 0.85 MB of data in transit at any given time. If a momentary failure occurs and an acknowledgement is not received for a packet, this amount data is lost and must be resent, further slowing the transfer.

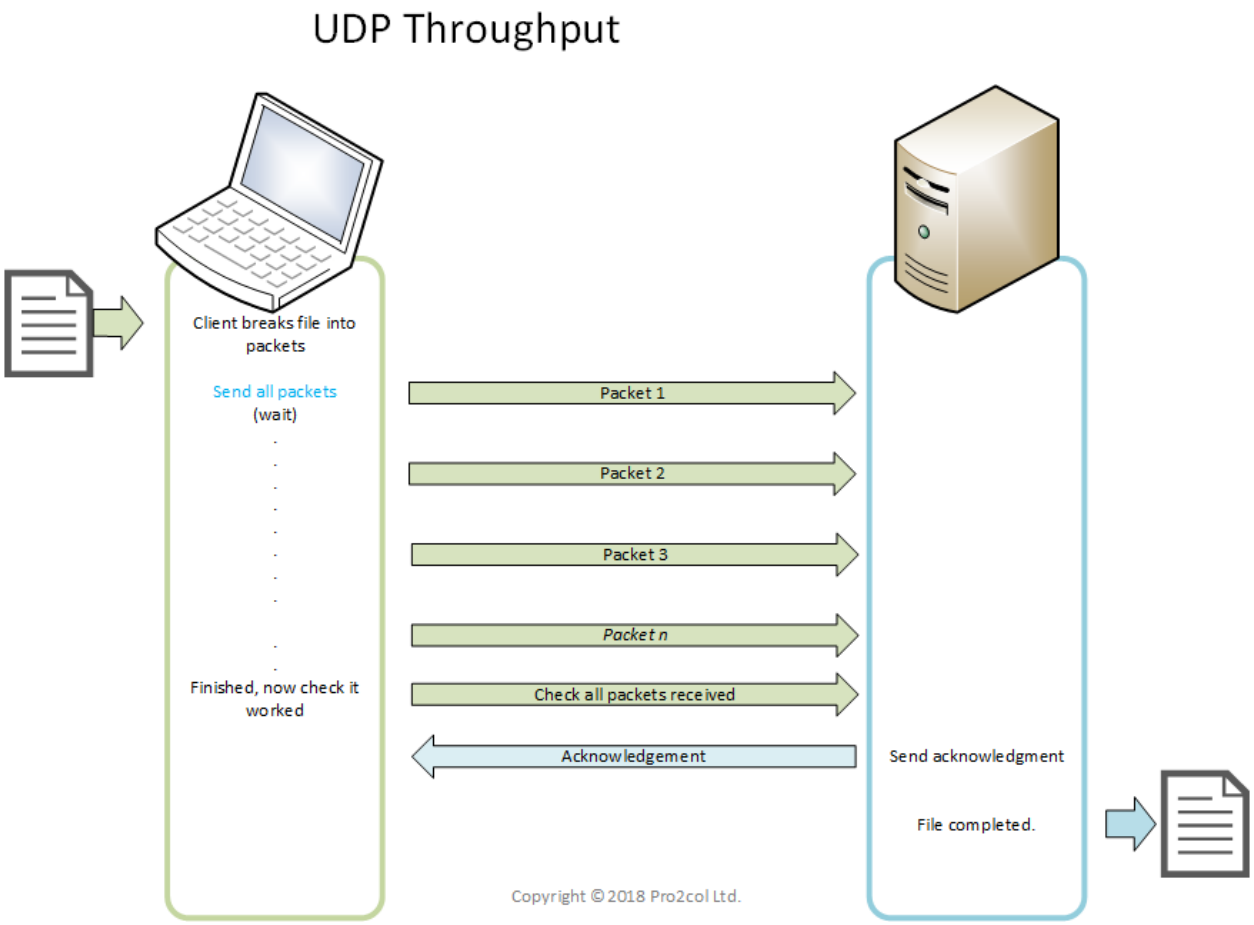


Finally, TCP makes use of a congestion control window – a method of limiting the amount of unacknowledged packets that may currently be in transit. Simply put, this mechanism starts any transfer at a low speed, but continually increases the speed until either the maximum possible speed is reached or packets are lost. At this point, the speed drops and the process starts again. This gives the distinctive saw-tooth pattern in transfer speeds found in TCP (each peak in the pattern represents one Congestion Avoidance Cycle)

UDP/IP Blasting

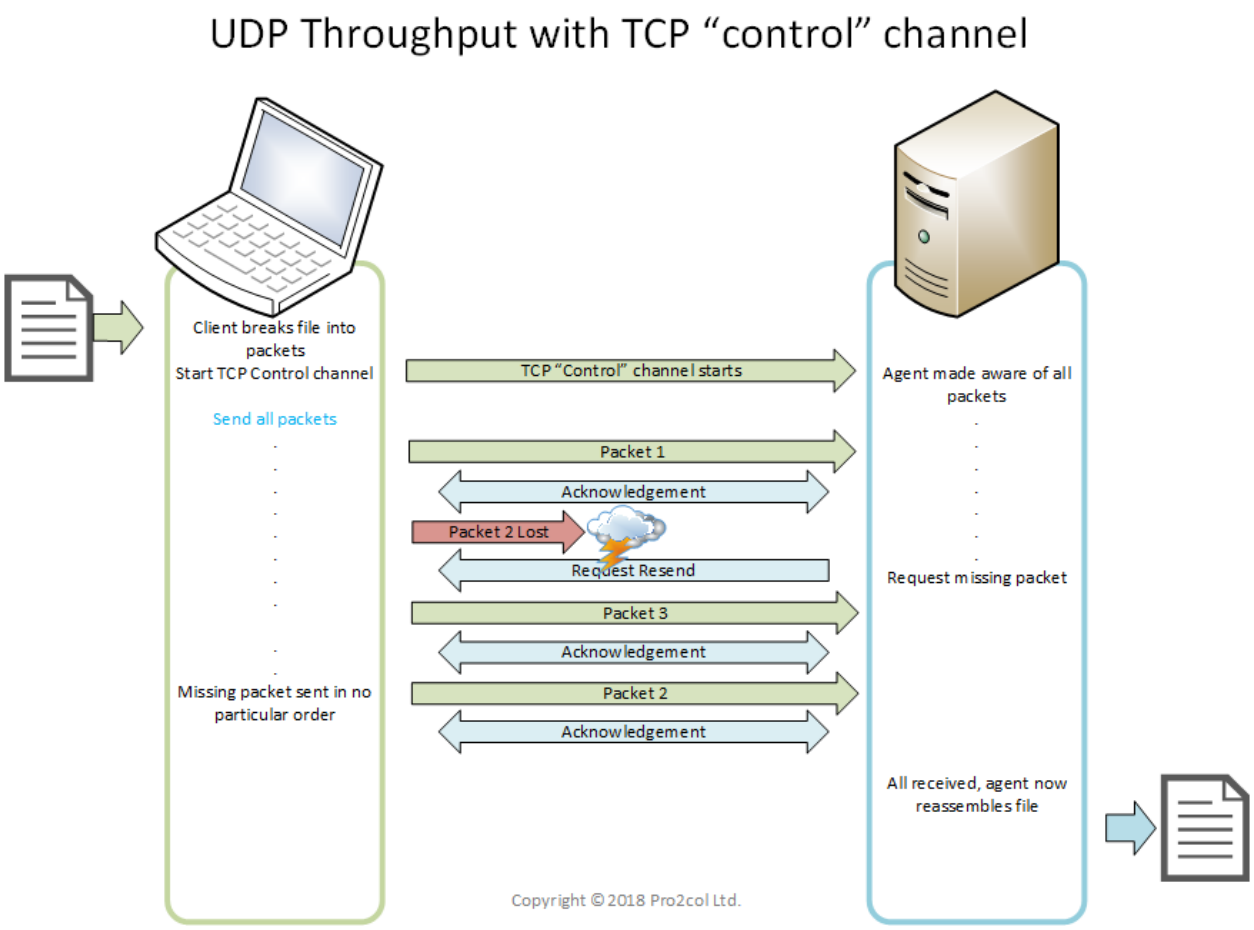
Like TCP, UDP is a popular way to send data across IP networks. Unlike TCP, UDP is an "every packet for itself" protocol that allows programs to communicate without the overhead of setting up a TCP connection, or waiting for other packets in a TCP stream to be delivered, get acknowledged, or ever show up in the original order.

By using UDP instead of TCP, accelerated file transfer solutions avoid the extra time it takes for TCP packets to reach their destination and send an acknowledgement of receipt back to the sender. Accelerated file transfer solutions also often send multiple UDP packets at once and/or out of order, so more data can be pushed across the IP network than could be if a single connection was used.



Controlled UDP

An increasingly common approach taken by several vendors is to transfer the data by UDP, but perform any control activities by TCP. This allows the speed of UDP to be fully utilized whilst still ensuring that data is received successfully. The downside with this combination is that it requires a proprietary controlling mechanism on both ends of the connection – so the sender will need to be using a client (or agent) provided by the recipient.



Aliases

Accelerated file transfer is also often known as "Large file transfer", “extreme file transfer”, “UDP file transfer”, “Fast File Transfer”, “WAN acceleration” and several other monikers.

Accelerated File Transfer Issues

Issues with Firewalls

Firewalls are generally optimized for TCP traffic, and it is rare to find a firewall that allows untrusted UDP packets through. Part of the reason for firewalls' distrust of UDP is that the source IP addresses stamped into UDP packages are easy to forge. All this distrust makes it difficult to securely support accelerated file transfer across the Internet without opening special ports between trusted parties.

Issues with VPNs

A traditional solution to networking issues like untrusted UDP is to use VPNs to tunnel untrusted traffic between two trusted endpoints. However, tunneling accelerated file transfer traffic over TCP connections often reduces or eliminates the benefits of using accelerated file transfer, so much care and testing should be performed before committing to this model.

Issues with QOS

A common side effect of using UDP to engage the full capacity of an IP network is that the network gets so flooded with UDP traffic that it cannot effectively process TCP traffic. This can have the unintended side effect of shutting down highly visible services like email and web services if they are on the same IP network.

To avoid this problem, many accelerated file transfer solutions come with their own throttle controls. In addition to these controls, many organizations use network segmentation (e.g., through VLANs) or "quality of service" (QOS) controls on their networking equipment to prevent UDP traffic from interfering with business-critical TCP traffic.

Issues with Standards

Most accelerated file transfer implementations are proprietary and only work with one or two software packages or (frequently) only work with a single vendors' products.

There are a handful of open accelerated file transfer implementations such as QUIC, UDT and Tsunami, but they are not widely used in file transfer operations. Some peer-to-peer network protocols (including the "torrent" protocols) offer tantalizing visions of similar protocols, but peer-to-peer protocols generally lack the identification and authentication traditional file transfer protocols and accelerated file transfer implementations include (and regulations frequently demand).

VIII. File Synchronization and Sharing

"File sharing" (a.k.a. "Enterprise File Synchronization and Sharing" or "ad hoc file transfer") gives end users an easy way to share files with other end users, without requiring administrators to set up permissions or relationships between participating end users. (End users can pick which other end users are part of the exchange.) File sharing is often distinguished from more traditional file transfer models that depend on administrators to grant permissions to specific folders.

File sharing normally uses HTTP/S and "secure file sharing" almost always uses HTTPS. End users participating in file sharing typically use an "email metaphor" or a "folder metaphor" to select and share files with other people. Some file sharing solutions use both metaphors, but using both in the same interface can confuse end users and make the entire solution less accessible.

File Sending ("Email Metaphor")

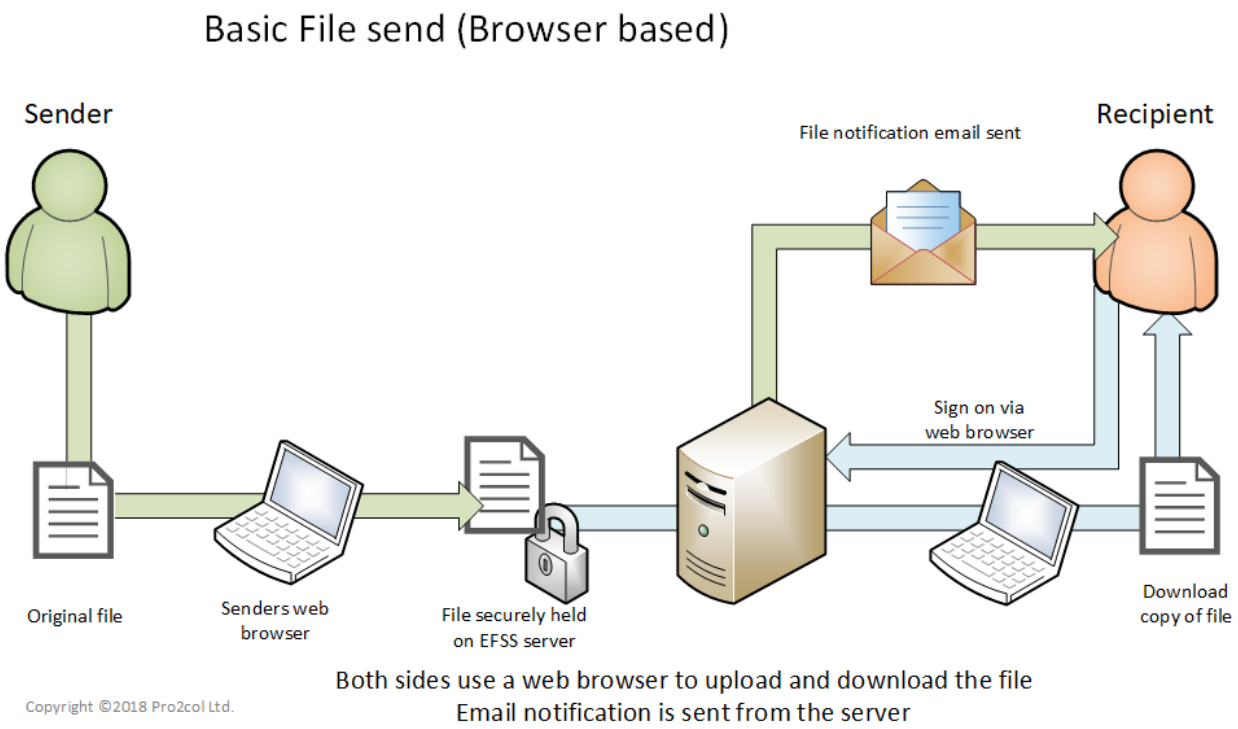
When file sharing uses the email metaphor, end users share files using a workflow and terminology similar to the email:

- End user enters the names or email addresses of file "recipients"
- End user picks one or file files to "send" or "attach"
- End user may type an optional "subject" and "message"
- A list of previously shared files is visible in a "sent items" collection
- Links are sent to recipients
- Recipients authenticate to server (if necessary)
- Recipients download files.
- Previously shared files can be found be looking at a list of "sent files" or "sent items"

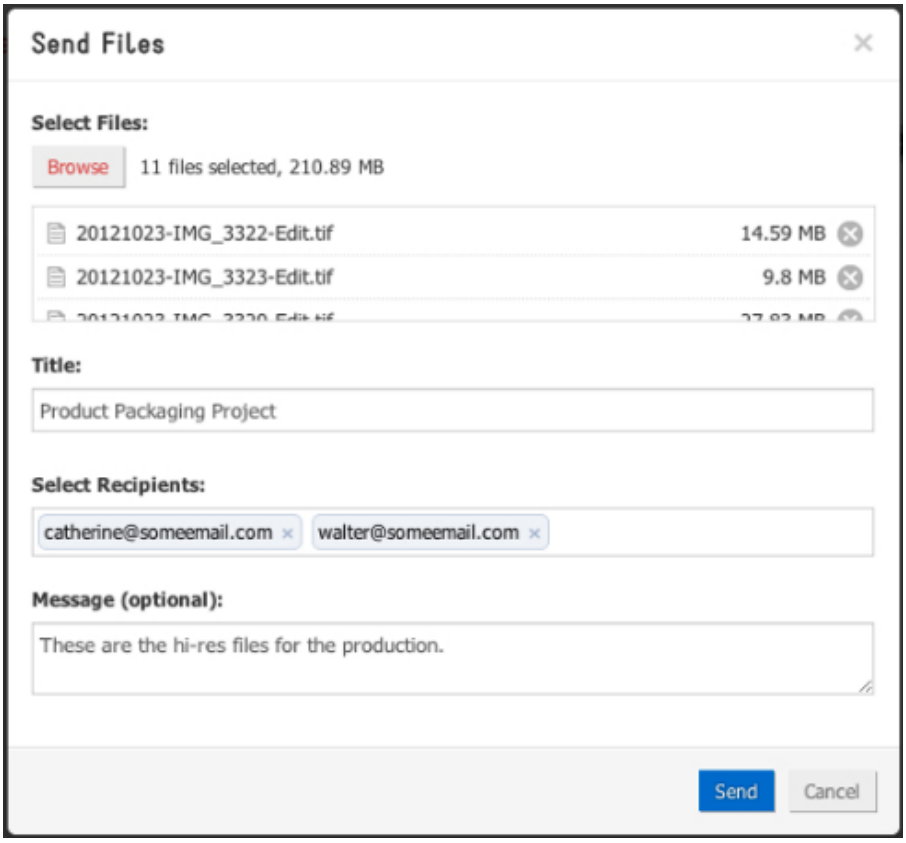
End users may start a file send from a web page or email client. Email client "plug-in" software is often used to facilitate this file sending by diverting files from their normal flow (to an email server) and saving them onto a file transfer server instead. Once files are "sent", they are stored on the file transfer server, notifications are sent, and they are delivered when recipients sign on to pick up their files.

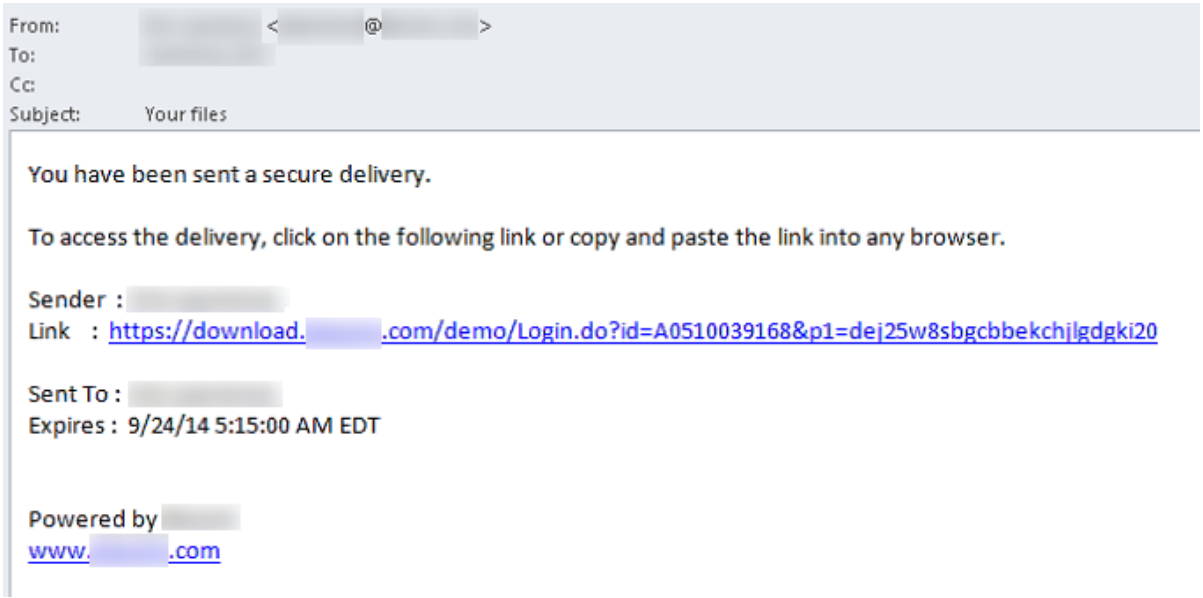
Basic File Send (Browser-Based)

The most basic of file send interfaces requires both the sender and recipient to use a web browser to upload and download sent files. A server in the middle stores files while they are awaiting download and sends email notifications to the recipients to tell them that files have arrived.



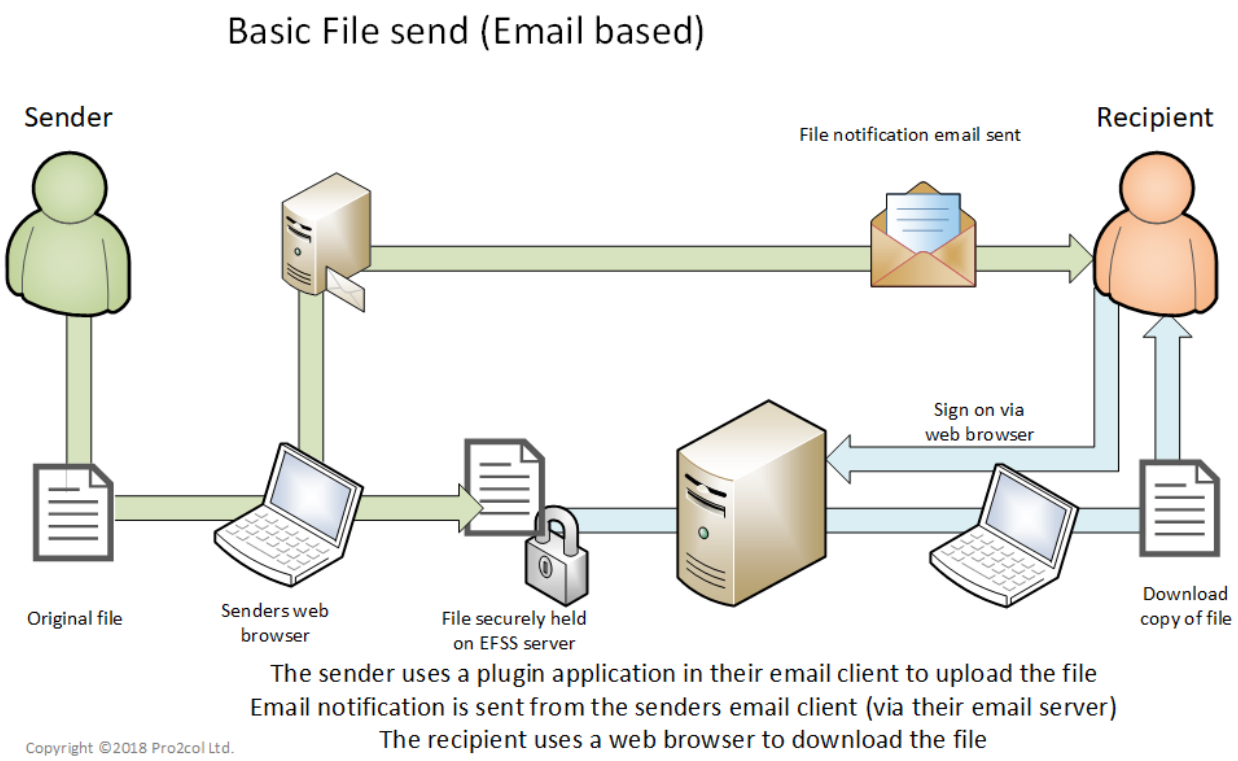
Examples of a typical basic file send form (from BitTorrent) and a typical email notification (from Biscom) are provided below.



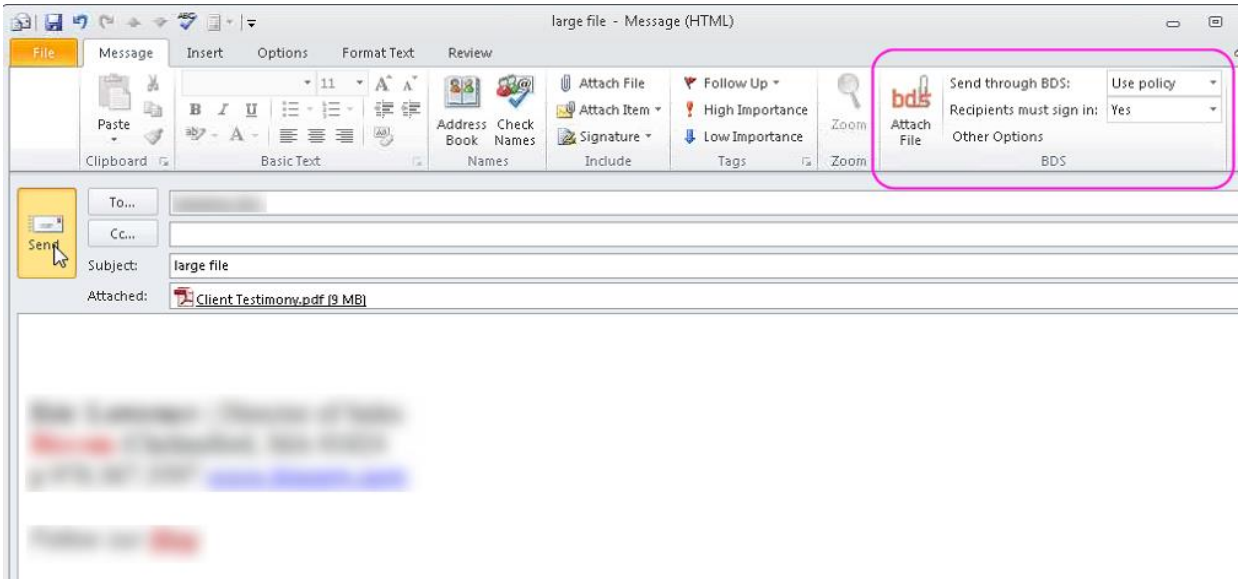


Basic File Send (Email-Based)

A common variation uses a "plug-in" application in the sender's email client to upload files. A server in the middle stores files while they are awaiting download. However in this variation, the email notification sent to the recipients to tell them that files have arrived is often sent from the sender's email client, not the file transfer server.



An example of a typical plug-in interface (from Biscom) that uploads files to a central server and then uses normal email to send notifications is provided below.

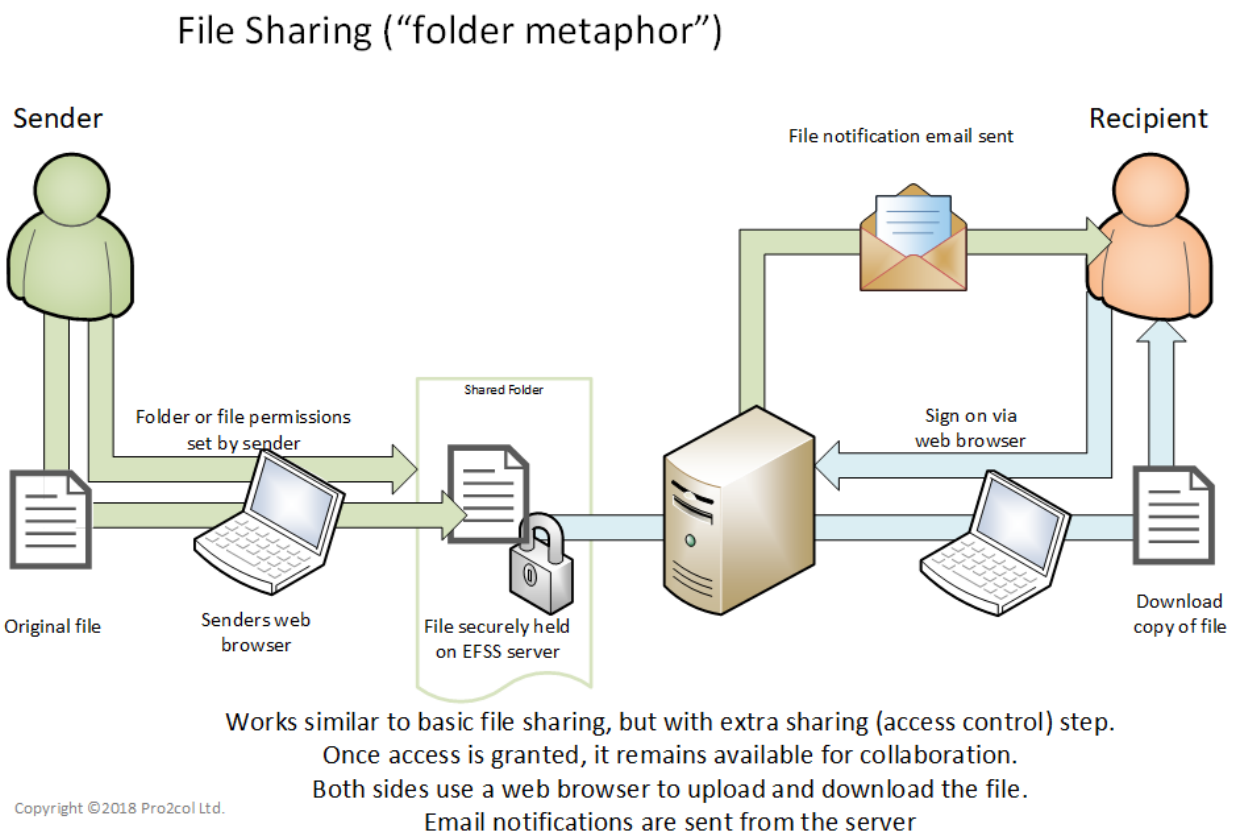


File Sharing ("Folder Metaphor")

When file sharing uses the folder metaphor, end users share files using a workflow and terminology similar to the following:

- End user navigates into a file folder, selects one or more files, and clicks a "share" action (button, menu item, etc.)
- End user enters the names or email addresses of people who should view the files.
- End user may optionally select the level of access the selected people should have on the files
- Links are sent to recipients
- Recipients authenticate to server (if necessary)
- Recipients download files.

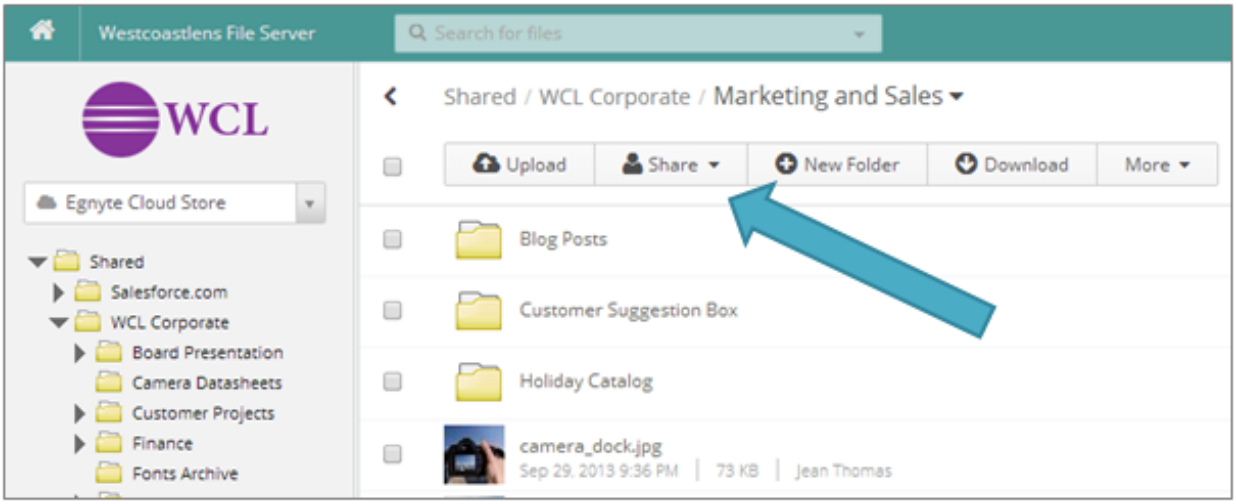
Previously shared files can be found by looking at folder or file "permissions" or "access rights".



End users may initiate this workflow from a web page or native client. Native client software can take the form of mobile applications, desktop file manager plug-ins and standalone desktop applications.

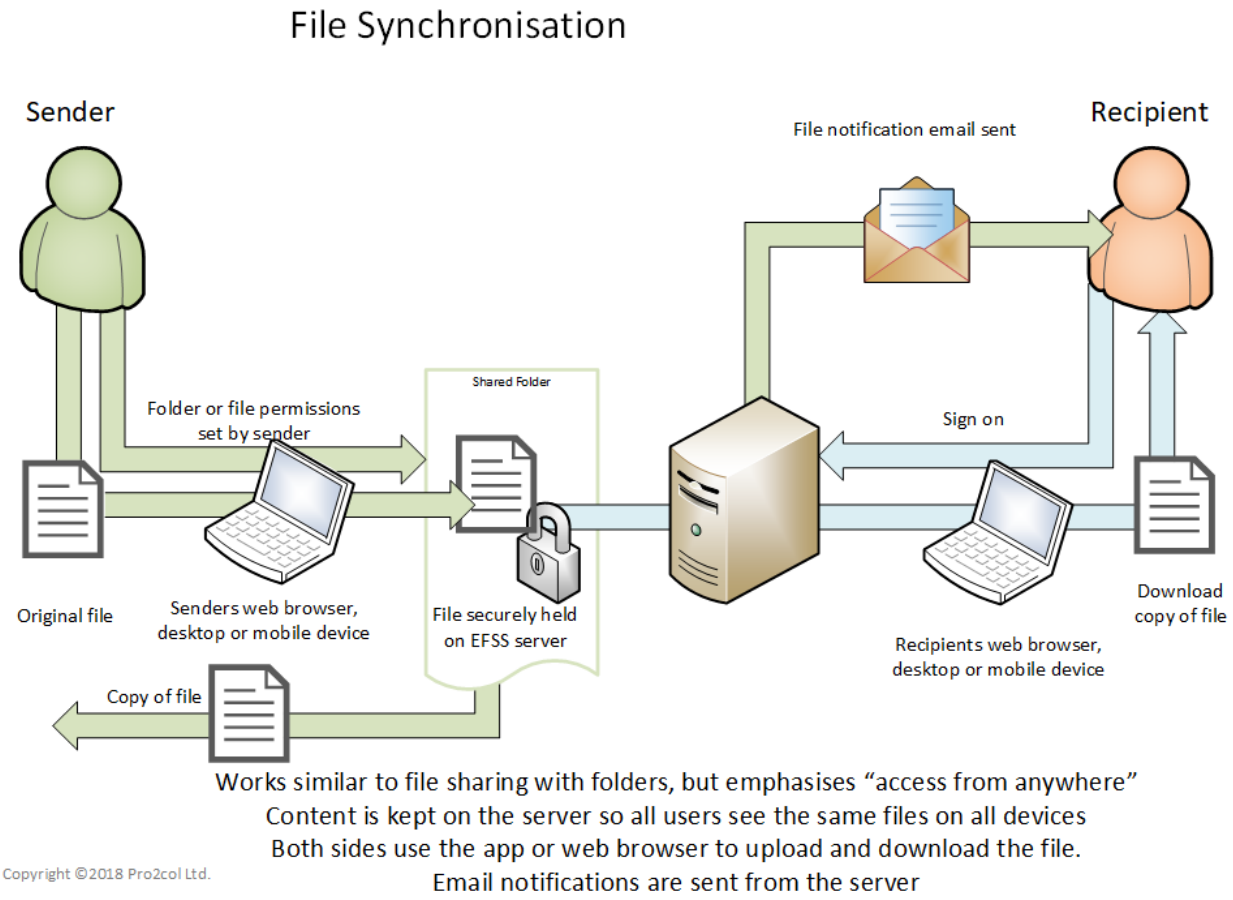
Once access to selected files has been granted, they are stored (or continue to be stored) on the file transfer server. The server also sends email (or other types of) notifications to the recipients to tell them that files have arrived. Files are delivered when recipients respond to their notifications and sign on to pick up their files.

An example of a folder sharing interface (from Egnyte) is provided below.

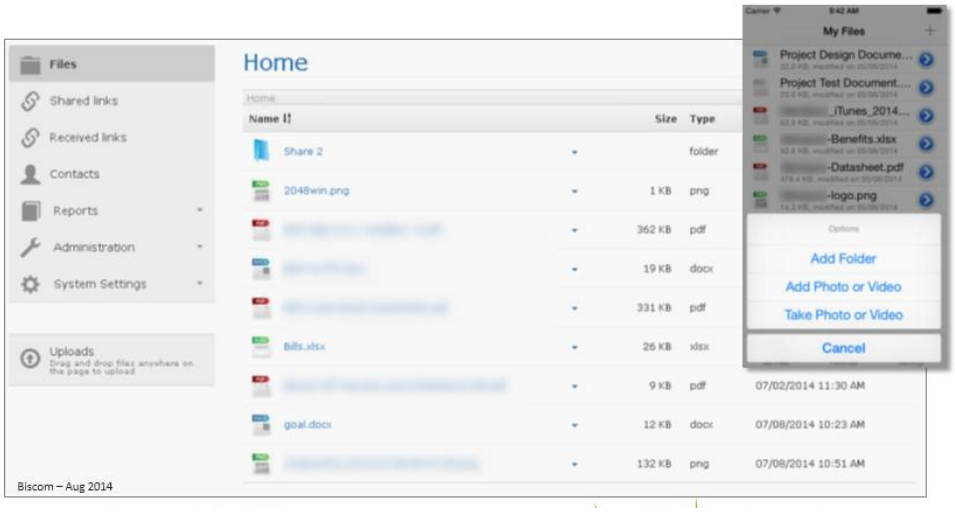


File Synchronisation

Many modern file sharing applications now use a second-generation model called “file synchronization.” In file synchronization, web applications, native desktop clients, email plug-in clients and (especially) native mobile applications all work together to provide end users seamless access to “their” files from anywhere. They also provide seamless access to at least one of the two major file sharing models (email-like or folder-based) through all or most of their native file transfer clients and web interface.



Examples of file synchronization interfaces for mobile devices and web browsers (from Biscom) are provided below.



Policies

Challenge Mechanisms

Once an end user has shared files, those files are stored on the file transfer server and access to the end user's selected recipients is granted. Most of the time, an email notification, text message or other notification is sent to all recipients, and these notifications usually contain a link each recipient can use to access the files, download them, and sometimes send files back in reply.

Many file sharing implementations (and all secure file sharing implementations) will challenge recipients before allowing them to access files. These challenges typically take one of the following forms:

- **Password only**: The end user picks a password and each recipient must use it to access the shared files.
- **Self-Registration**: Each end user must go through a web-based registration process to establish more information about himself/herself before he/she is allowed to download the shared files. (Subsequent access to more files omits the registration step.)
- **Pre-Registration**: Each end user must already be an authorized user on the system to access shared files. (Note that administrators still do not need to set up user-to-user relationships, even if they need to individually provision end users in this model.)

Non-secure file sharing implementations allow anyone with the URL that refers to shared files to access those files. File sharing implementations that send passwords in the same message as the file sharing access links are nearly as insecure.

The secure implementations listed above increase in security from top to bottom. However, more secure implementations also make it more difficult for end users to access shared files without additional training.

Retention Policy

Most file sharing implementations include file retention policy settings, many of which can be set by end users at the time they elect to share files. Common expiration settings include:

- Expiration after a certain number of hours, days, weeks or months
- Expiration at a specific “drop dead” date and time
- Expiration after a certain number of reads
- Expiration once all identified recipients have a copy

Once expiration has been reached, no recipient may access the shared files any longer. However, file sharing implementations vary widely on what happens next. Some implementations, particularly those using email metaphors, will delete the shared files. Others, particularly those using folder metaphors, will keep the files and continue to allow the original sender to access them. Multiple implementations also archive expired files (for eDiscovery, etc.), send notifications to affected parties and perform other activities.

Mobile Devices

There is an increasing use of mobile devices to access file transfer servers, and file transfer applications have changed to support this. Some File Transfer server vendors have created discrete client applications that can be installed on the devices to simplify the file transfer process. There are several considerations to be taken into account when dealing with mobile devices, for example:

- Device applications have to be able to operate on multiple (preferably all) device operating systems
- Mobile Devices generally do not have a preconfigured IP address; therefore permanent blacklisting of IP addresses on a server may impact legitimate users (and have no effect on hackers)
- How to handle data on a device if it becomes lost or stolen. In some applications, the device contains an encrypted “vault” where data is stored; in others, the data is automatically wiped after a certain period of time has elapsed

Although most devices will use some kind of web service to connect to a file transfer server, there are also a number of traditional FTP and SFTP clients available for mobile device applications. These can of course be used in the same way as those clients would be used on a computer, however it is also important to recognize the dangers associated with storing credentials on a mobile device.

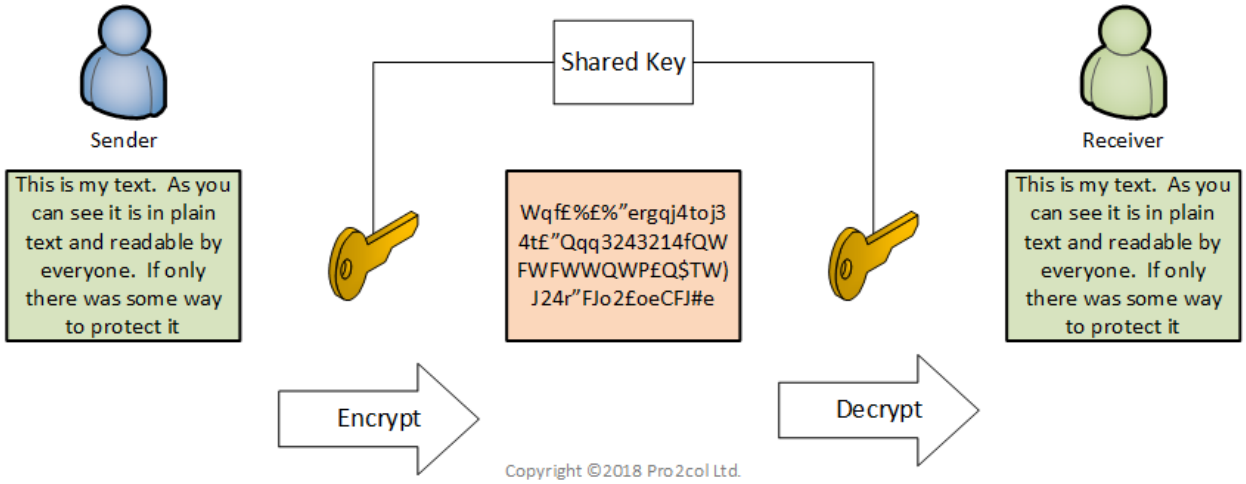
VIII. "At Rest" Encryption

Several secure file transfer protocols encrypt files in transit, but encrypting files "at rest" (i.e., while stored on disk or memory) is also an important part of many file transfer workflows.

Note that combining "at rest" encryption with non-secure file transfer protocols (such as PGP-encrypted files sent over FTP, or using AS2 with HTTP) has the effect of also securing data in transit because file data is encrypted end-to-end. Nonetheless, use of cleartext protocols in this way can still expose sensitive information like credentials.

Symmetric Encryption

Symmetric at-rest encryption uses a single key to encrypt and decrypt a file.



The key may be selected and stored on a central file transfer server, or selected by one end user and shared with the other end user.

Symmetric At-Rest Encryption Advantages:

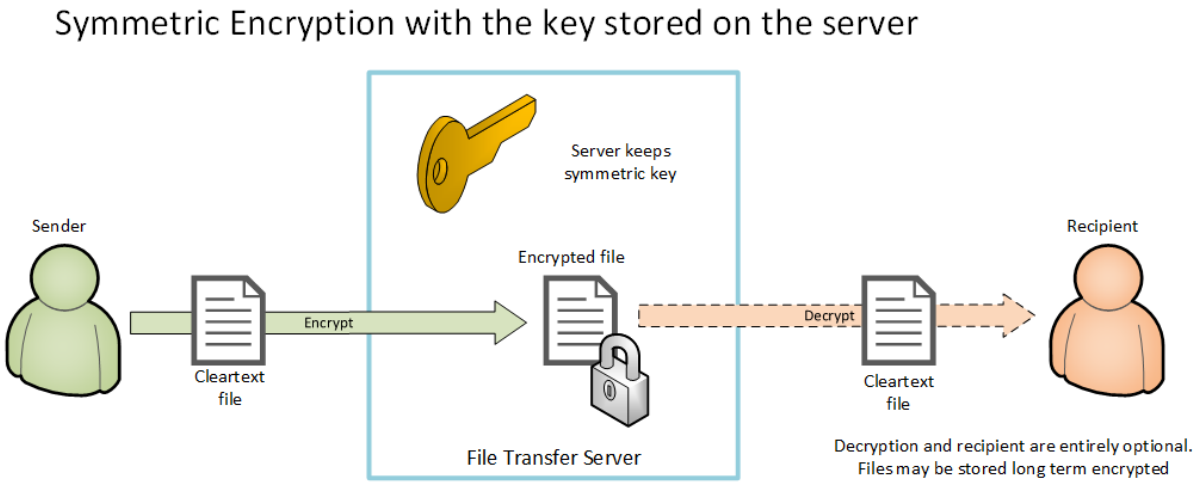
- If implemented in a file transfer server, all encryption and decryption can be invisible to end users.
- Symmetric encryption offers higher performance than asymmetric encryption.

Symmetric At-Rest Encryption Disadvantages:

- Signing files requires a separate procedure.
- If a file transfer server or the channel users use to communicate keys (e.g., email) is compromised, keys and therefore files can be compromised too.
- End users who are allowed to select their own key often reuse and share the same key for all transfers (e.g., keys are easily discoverable), or create their key using a weak password.
- Symmetric encryption is generally regarded as less secure than asymmetric encryption.

Key Retained on Server

If the key is retained on a central file transfer server, end users can be completely oblivious to the fact that their files are being encrypted and decrypted on the fly. There is a chance that the server keys could be stolen and all files would be at risk, but this type of symmetric at-rest design is used today in many file transfer software packages, "online backup" services and other applications.

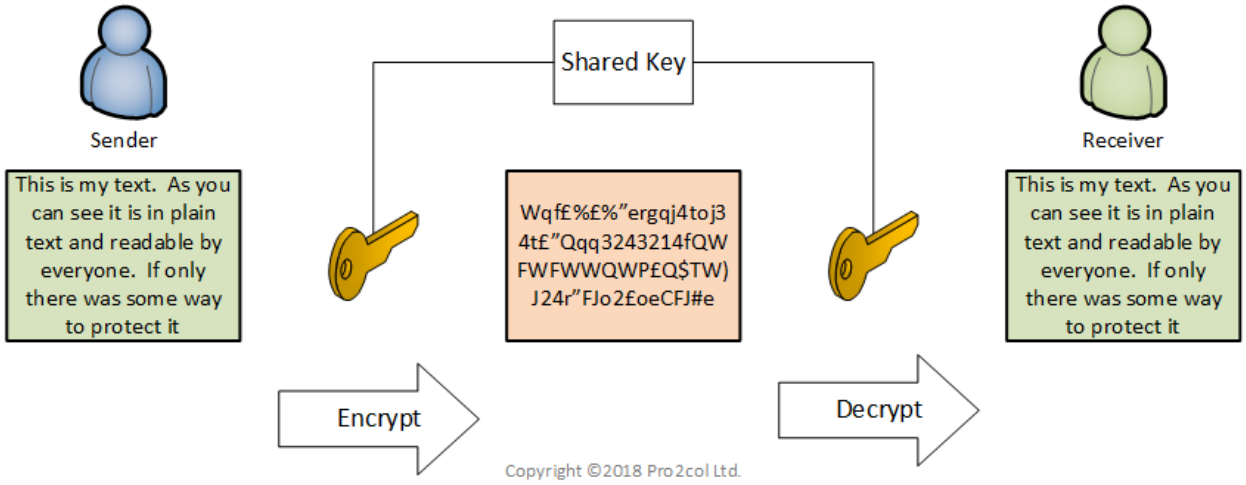


Convenient: neither the sender nor the recipient need to know they are encrypting or decrypting files
Somewhat insecure: If the keys are stolen from the server, all files can be exposed

Key Sent to Sender

Senders and receivers can also choose to use symmetric file encryption directly, but at the cost of additional hassle since both parties need to use compatible encryption software, and questionable security since a key needs to be sent from one person to another.

The most common application of this design remains the use of "encrypted zip files, protected by a password," where the password is communicated over a phone call, IM session or fax. However, this design has largely been replaced with asymmetric file encryption designs (like PGP or SMIME or even more advanced Zip encryption) that eliminate the need to communicate keys over different channels.



"Automatic Encryption"

When a file transfer server or cloud service describes its security and mentions that "files are automatically encrypted with AES or DES3," it is usually using symmetric encryption, and the keys are often stored on the server, service or a nearby authentication source such as AD. *(See Key Retained on Server above.)*

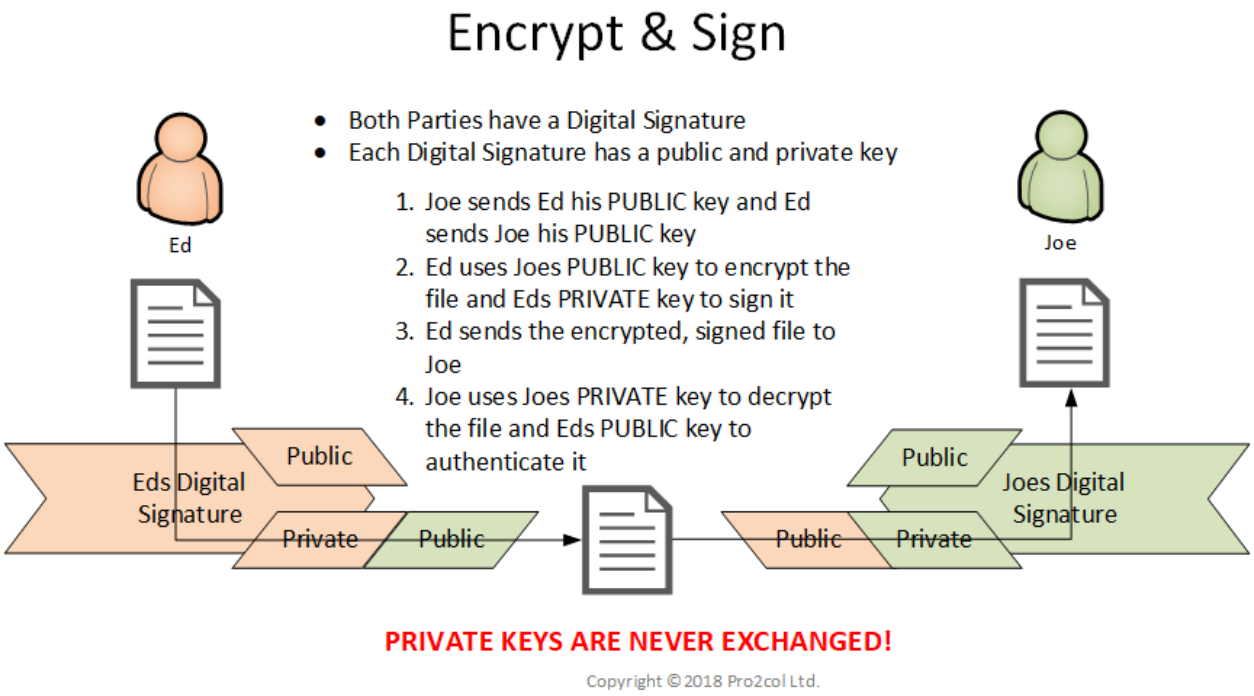
"Zip Encryption"

Many compression tools, including many Windows Zip utilities, have options to encrypt compressed output with a password or key. These options generally use symmetric encryption (unless they prompt for "public" and "private" keys) and provide a durable at-rest encryption solution that can be used to provide end-to-end encryption. However, the price of this convenience is the relative lack of security around the key shared between parties. (See Key Sent to Sender above.)

Asymmetric Encryption

Asymmetric encryption uses two different keys to encrypt and decrypt each file, two more keys to sign and verify each file, and requires each party to exchange their public keys before any operation can take place. It requires more work than symmetric encryption up front, but is generally regarded as more secure.

The key used by a sender to encrypt a file is the public key of the recipient, and the key used to decrypt a file is the private key of the recipient. A sender's private key is also normally used to sign the encrypted file, and the sender's public key is normally used by the recipient to verify/validate the signature of the sender.



Advantages:

- Even if channel or servers used to transmit files or channel used to share public keys are compromised, private keys and files remain safe.
- Signing files (which allows validation of the sender's identity) is a built-in procedure.
- Generally regarded as more secure than symmetric encryption.

Disadvantages:

- Requires end users to exchange keys and use their encryption technology correctly.
- End users often accidentally send their PRIVATE keys to each other.
- Slower performance than symmetric encryption.

PGP Encryption

PGP ("Pretty Good Privacy") is a very popular implementation of PKI and asymmetric encryption that includes a signed, cryptographic-quality integrity check. All modern PGP implementations conform to the vendor-neutral OpenPGP standard, but PGP is also a commercial product and trademark currently held by Symantec.

An alternative open source product – GPG – is fully interchangeable with other products following the OpenPGP standard.

PGP clients can both encrypt and decrypt files. The term "PGP server" is sometimes applied to a server-class PGP client, but is also applied to servers that store multiple PGP keys (usually public keys) for multiple users.

PGP clients are often implemented as command-line utilities, GUI utilities, email client add-ons, or components of FTP clients. PGP clients are rarely built into operating systems, but multiple implementations of PGP are available for most platforms.

SMIME Encryption

SMIME ("Secure/Multipurpose Internet Mail Extensions") is a popular implementation of PKI and asymmetric encryption commonly associated with "secure email" and the "AS*" protocols (including AS2). It can be used to provide encryption at rest, or, when sent via email or other protocols, encryption in transit. It also includes a signed, cryptographic-quality integrity check.

Standalone SMIME clients are rare, but SMIME is built in to most popular email clients, including Microsoft Outlook. SMIME is also integral to all AS* protocol implementations and is therefore built in to AS1, AS2 and AS3.

"Strong Zip" Encryption

A number of compression utilities allow the use of PKI in addition to single-key encryption and decryption. This feature is often known as "strong zip encryption" just as PKI-based encryption is sometimes called "strong encryption" and PKI-based authentication is often called "strong authentication."

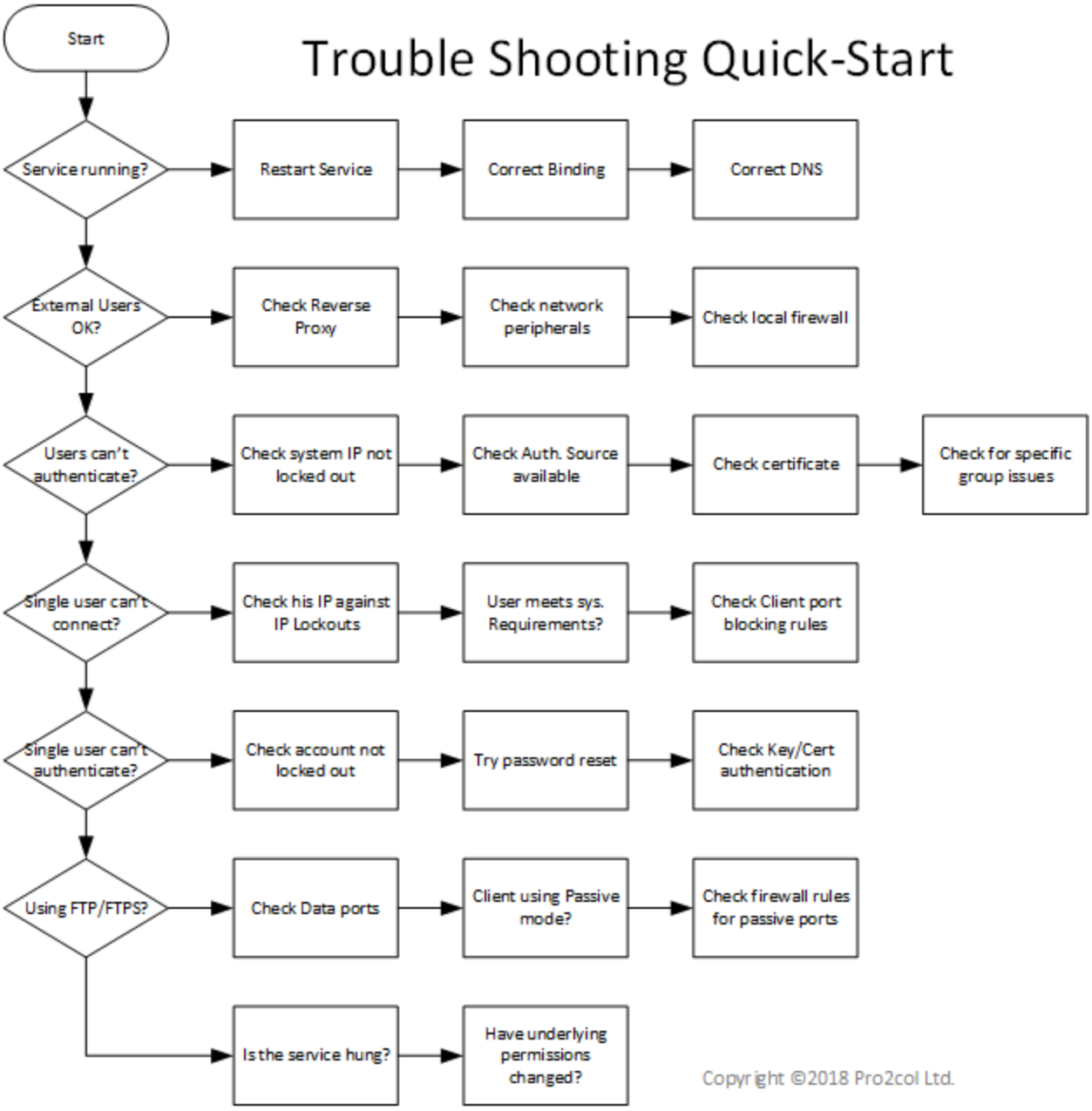
IX. File Transfer Operations

This section covers the operational aspects of managing a file transfer solution. Every organization is different in how file transfer operations are managed, however there are some core aspects that are common to all.

Server Troubleshooting Guide

Connection issues are common in client/server applications, and are especially common in file transfer applications. To quickly and reliably diagnose these issues, the following procedure can be used.

1. Check your file transfer service - is it still running?
    - Restart it if it has failed.
    - Check to see if it is binding properly to its expected ports and IPs.
    - Check the DNS entry of the service points to the correct address
2. Are any of your external users able to connect?
    - If internal users are OK and you are using a reverse proxy, check it is running
    - If not, your firewall, routers, load balancers or other networking equipment may need attention - investigate.
    - Check local 'soft' firewalls have not changed
3. Are all (or most) of your users able to connect but not authenticate?
    - Has your application locked out the IP used by all incoming connections? (Some networks expose external IP addresses to an application, others hide them.)
    - Are you using an external authentication source like Active Directory? Is that service active and is the connection to that resource also available?
    - Is your server certificate or key still valid? (These often periodically expire.)
    - Is a specific set of users impacted? (May be a group issue)
4. Is a single user not able to connect?
    - Is the user's IP locked out?
    - Is the user using the correct type of client, hostname/IP, protocol, and port?
    - Are there any port-blocking rules configured at the user's end?
5. Is a single user able to connect but not authenticate?
    - Is the user's account locked out?
    - Are the user's credentials correct or expired? (E.g., try resetting a password.)
    - Is there a keypair issue?
6. Are some of your end users able to connect and authenticate but they cannot transfer files or list folders?
    - Are you using FTP or FTPS? If so, the data channel required to transfer files and list directories may be blocked.
        - Is the end user's FTP client configured to use "passive" or "firewall friendly" connections?
        - Have you banned active connections?
        - Are you allowing a select number of passive ports in through your firewall?
        - Is the user's equipment blocking FTP ports?
    - Is the file transfer service hung or partially hung? Can you safely restart it?
    - Has someone changed the underlying permissions on the files?



File Transfer Service Levels

Like other IT groups and cloud services, file transfer operation groups can guarantee a level of service that encapsulates uptime, response time and throughput. When other parties ("customers") acknowledge a promised level of service, it is known as a "Service Level Agreement" (SLA).

File transfer service levels may include promises to watch for groups of files during scheduled file transfer windows or respond to files bearing certain characteristics within a specific period of time. File transfer uptime promises are similar to uptime promises on other systems and typically involve an availability percentage within a specific time window, sometimes with exceptions for scheduled or emergency downtime.

Customers enjoy an obvious benefit from SLAs, but providers of file transfer SLAs also often use the provisions of the agreement to give their systems and processes the space and time to generate and respond to files.

To avoid SLA violations, providers should set up alerts and notifications that cover unresponsive file transfer services (which would violate uptime agreements) and conditions that signal that customers or responses are “about to miss” certain transfer windows. For example, an alert may be set if a watched directory still contains no files 30 minutes before its associated transfer window expires for the day.

Sample File Transfer SLAs

Response Examples:

X files from customer Y should arrive by time Z on days A, B and C.

When files under size X are receives between the hours of Y and Z, a well-formatted response or error will be made available to the sending customer within A minutes.

Uptime Examples:

The file transfer server will be up at least X% of the time in any Y day period.

The file transfer server will be up at least X% of the time between the hours of Y to Z, except for scheduled downtime (never to exceed A hours and provided at least B days in advance).

Automated File Transfers

Automated file transfers depend on a number of concepts like transfer windows, scheduled file transfers, trigger files and event-driven file transfers.

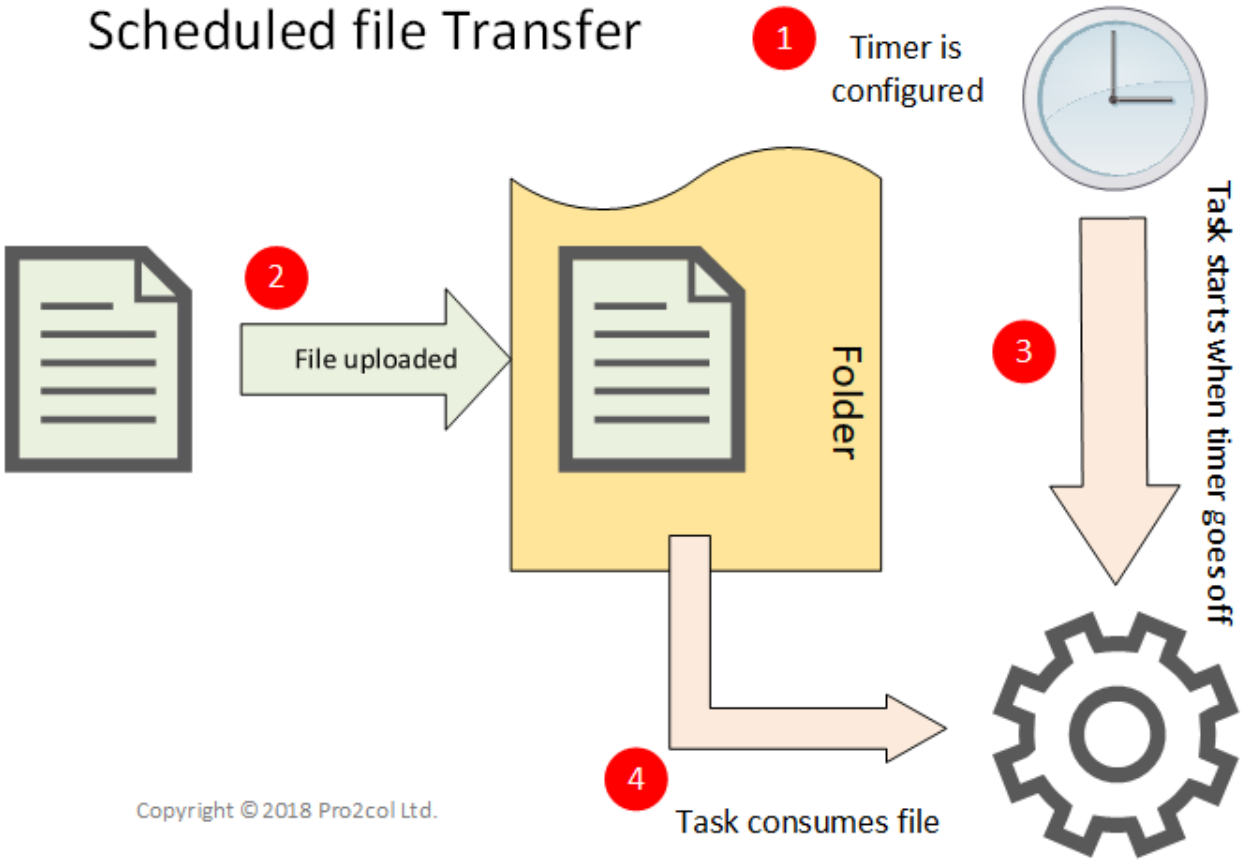
Transfer Windows

File transfer "windows" are periods of time when certain file transfers are allowed or expected. Transfer windows are often specific to certain types of files, and are generally related to other file transfer automation and file transfer SLAs.

For example, a company may expect "*.rpt" files from its customers during a daily transfer window that opens at 2am and closes at 10pm. "*.rpt" files received outside the transfer window may be denied, ignored, queued for later processing or processed as best as possible (not subject to the transfer window's SLA), all at the discretion of the company that set up the transfer window.

Scheduled File Transfers

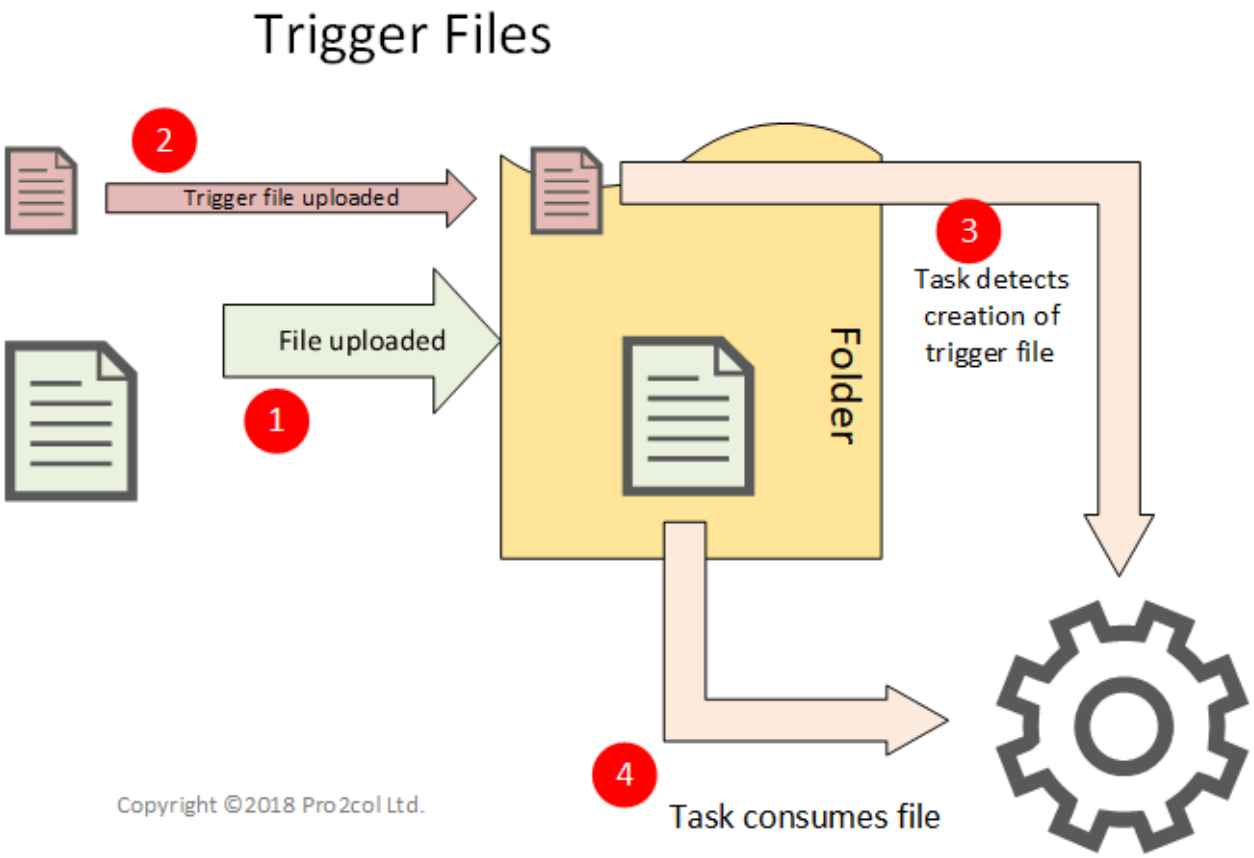
Scheduled file transfers can be used to periodically send nightly backups, batches of reports and avoid the delivery of physical packages. They are of little use in real-time or "near real-time" workflows such as interactive web queries.



The schedule that powers a transfer can be driven by a variety of operating system schedules or scheduling applications. The powered transfer application is usually a file transfer client, but can be a file transfer server in certain cases. For example, a workflow could be configured to automatically delete files after a certain period of time.

Trigger Files

Trigger files are used to kick off or complete other file transfer operations. Trigger files are often empty files with important filenames, or files with meaningless names that contain important parameters. Another file transfer process watches for these files to appear, reads (or "consumes") them, and then takes action on a related target.



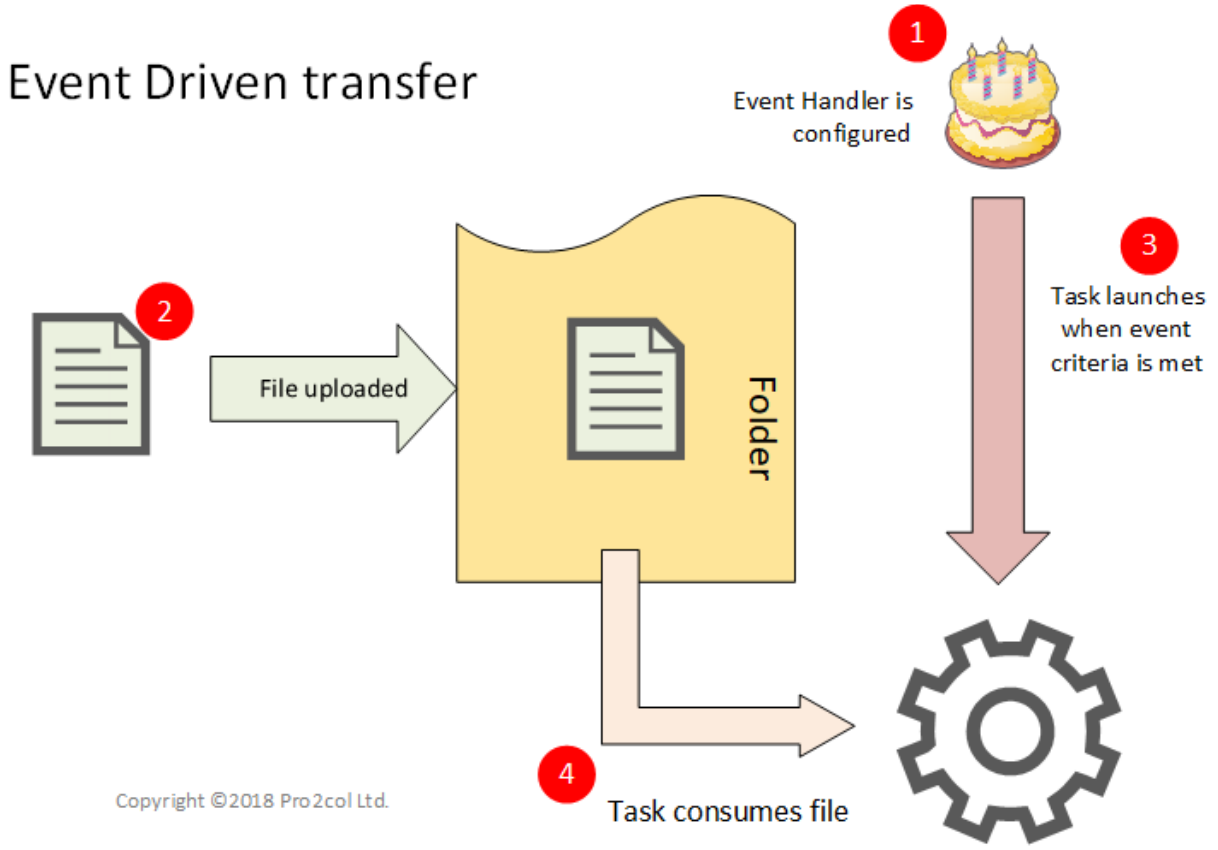
For example, an empty ("zero byte") trigger file named "report20140214.trigger" may be written to a folder to tell a monitoring application to generate and send a report file for February 14, 2014. Or, a file named "234924.xml" containing a date and filename could be written to a folder to tell a monitoring application to generate and send the same file. Commonly, the trigger file will have the same name as the file to be transferred, but with a different extension; a trigger file named 1234.ok would indicate that it is safe to transfer file 1234.txt.

Trigger files are sometimes used to "speed up" scheduled file transfer tasks by essentially converting them to event-driven file transfers.

The trigger file is generally deleted after the transfer, however it may also be transferred to the destination, especially if it contains information about the file (for example header information). This is entirely dependent upon the situation, however to avoid a 'false triggering' it is important not to leave the trigger file in place after the transfer.

Event-Driven File Transfers

Many events besides schedules and trigger files may kick off file transfers. These events include uploads, downloads, deletes, transformations, low CPU usage and thousands of other possibilities. File transfer events may also kick off email notifications, log entries, alerts, custom scripts, command-line applications and thousands of other possibilities. All of these types of automation are generally called "event-driven" tasks, and if they kick off file transfers, they are known as "event-driven file transfers."



Unlike scheduled tasks, event-driven tasks can be used to drive "near real-time" workflows, such as processing and responding to a file that was just uploaded.

Some file transfer software contains a 'file notification' event handler. This is a specific kind of event handler which will detect changes to file systems and is used to trigger a transfer following the creation or modification of a file. This is commonly used when transferring files from a Windows UNC path.

Continuous File Transfer

A crude but occasionally effective method of file transfer involves a script that attempts the same file transfer again and again in an infinite loop. If a pause (or "sleep") in incorporated into this kind of loop, it functions like a scheduled task. However, if the loop mindlessly attempts the same transfer again and again it can unnecessarily tax the resources of the client and server machines involved.

The signature of a continuous file transfer can be detected on a file transfer server by looking for a rapid series of successful sign-ons and sign-offs (or disconnects) involving either the same transfer or directory queries without action. In some circumstances there is a risk of triggering anti-hammering protection at the firewall level when the frequency of connections reach a certain threshold.

File Transfer Resume ("Checkpoint Restart")

Several file transfer protocols, including FTP/S, SFTP, HTTP/S and the newest iteration of AS2 support file transfer "resume" or "checkpoint restart" commands. These commands can be used to:

- Restart the transfer of large files (interrupted through lost connection, etc.)
- Speed up transfer of files by simultaneously requesting different parts of the same file at the same time (multi-threading)

Refer to the Multi-threading section for more information of resume for speed.

Both client and server must support resume commands for them to work, but the file transfer client is always in charge of deciding when resume commands should be used.

Common scenarios that may trigger the use of resume commands include:

- Client notices that destination file is smaller than the original file
- Client notices that its connection to the remote server was lost during a file transfer

Some scenarios that should not trigger the use of resume commands (and may trigger the restart of particular file transfer instead) include:

- Client performs a integrity check on a partial file, compares it with the same section of the original file, and finds that they do not match
- Client notices that the destination file is equal in size or larger than the original file

File Transfer Workflows

File transfer workflows allow people to automate complex tasks using one or more file transfer operations.

Common file transfer workflows include:

MOVE

- Download/Upload a file
- Delete the original

DELIVER

- Trigger on file download
- Delete the original

STAGE AND FIRE

- Upload a fire
- Start a remote task (either directly or indirectly – i.e. rename file)

READ AND RESPOND

- On file upload
- Start task to process it
- Send response to sender

UNPACK AN ARCHIVE

- Download an archive (e.g. zip file)
- Extract the files from the archive
- Delete the archive file

ENCRYPT/DECRYPT

- Encrypt or decypt a file
- Delete the original file
- Send resulting file to someone else

File transfer workflows are often implemented in scripts, programs, scheduled tasks, GUI automation and many other technologies. More advanced workflows make allowances for dynamic changes to the flow of a task, for example setting the destination host on the fly, or renaming files based upon the value of certain variables.

Monitoring

An important consideration of any file transfer system is the way that it is monitored for issues. Today, most valuable monitoring tools are sold or deployed separately from file transfer systems, and a degree of configuration is required to ensure that the correct information is emitted from the file transfer system to achieve the desired monitoring.

Server Functions

There are key parts of the server that need to be monitored. The first of these is checking the availability of key file transfer services (or “daemons” on *nix). This function can be performed by many monitoring tools that scan for listening TCP services and/or specific protocols. However this level of monitoring will not necessarily indicate if the file transfer server is working correctly. In mature deployments, the following scenarios are often monitored:

- Actual use of the service, i.e., uploading or downloading files. (Simulated activity is often called “synthetic transactions”)
- Checking gateway ports (if a file transfer gateway has been deployed)
- User threshold exceeded (both total number of user accounts and active accounts)
- Passive port range exhausted (if FTP/S is in use)

- Filesystem full / unavailable
- Hammering, or impact to system from DDOS

Of course, there are many other circumstances that should be monitored, including file transfer errors, and often including "missing" servers to which the software can usually connect. In many cases the file transfer software will take this into account automatically and generate alerts of some kind.

Notifications

Most file transfer software will generate some type of email or text alerts to inform administrators of unusual events. Many administrators configure these notifications to be sent directly to a shared mailbox or alerting system. Systems will also often send log events to a database, syslog server, Windows Event Log or other message queues, where monitoring systems and other automation may consume them.

Another layer of notifications centers on informing users of new files for them to download. Many file transfer servers use a "notification permissions" concept to achieve this functionality. In this case, administrators select folders on which users should receive an email alert when new files arrive, just like they would select folders from which users could download. Some servers require specific emails on these notification settings, but many others simply use the email address associated with the user account. These notifications will often include a hyperlink to the file in the email to simplify things for the end-user, but it is important to ensure that these links do not include any form of credentials.

Automation

Workflow automation also needs to be monitored, regardless of whether it is an off-the-shelf product or a home grown cron scheduled suite of batch scripts. First, whatever tool is used to schedule transfers needs to be monitored. This is often achieved by scheduling a repetitive script to perform some activity, then checking that it actually occurred.

For example, file transfer automation may be configured to periodically update a text file, then validate that the updates are happening. If the file stops being updated, the lack of activity probably indicates that the scheduling has stopped for some reason (e.g., *nix cron or Windows task scheduler scripts may both stop working if system resources are depleted). Similarly, a scheduler built into an application may be impacted by system resources, locked accounts or other contributing factors.

A strategic approach to failing transfers is also often helpful. For example, if automation retries until success, then it should include a maximum counter of some kind to avoid infinite loops. For critical transfers, email or text alerts are probably warranted. For others it could make more sense to simply check the last run status every few hours.

Reporting

It is important to report on the activity of a file transfer server or workflow engine to operations and the business units the system supports. Evidence of activities is also crucial to auditing, and is often mandatory when industry standards or regulations are considered. Additionally, plotting growth in the number of transactions or users will permit intelligent capacity planning. Reporting and logging (including detailed traces) are also vital during troubleshooting.

The method of reporting employed will be driven by the design of the environment. Many file transfer solutions use a database to record activity and have built-in tools or add-on modules for reporting purposes. If these are not available, but a database is in use, custom or bespoke database queries may

be needed to create reports. Either solution can often use of the existing file transfer solution to store or distribute the reports as needed.

If no database is in use, it may be necessary to build reports by interrogating log files. This is often performed by scripts. Even when these scripts exist, it is also important to maintain constant logging levels to ensure key information is retained (and the scripts work); otherwise, less verbose logging could kill a “log scraping” process.

Dashboard

A dashboard is a way of presenting information about several aspects of a file transfer system in one easy to locate place. Dashboards may be updated dynamically in real time (for example, number of currently logged on users), or may be populated with several reports showing recent historical activity. Many dashboards make use of web pages to display this information and are often configurable to individual’s requirements.

To get the best results from a dashboard, an administrator should be able to interact with the file transfer solution based upon the information being presented – for example by dropping connections or stopping transfer jobs.

High Availability and Disaster Recovery Considerations

File Transfer is no different to any other computer environment in that a plan needs to be in place to safeguard the environment should a serious disruption occurs. Disaster Recovery (DR) tends to imply a backup-restore process, whilst High Availability (HA) implies that normal service can continue using an alternative site or server without end-users being generally aware of an issue.

Disaster Recovery

First, it is important to note that in a file transfer environment, data is frequently volatile. This means that it appears in the system only as a midway point between two or more other systems. Because of this, many organizations may find that backups of transient data bring no benefits as they are frequently outdated within minutes of a backup occurring.

However, it is important to always back up the configuration of your file transfer solution. Key components of the configuration to consider are:

- Folder structure & permissions
- User profiles & credentials (both passwords and SSH keys/SSL client certificates)
- Server keys, certificates & port allocations

Often it makes sense to restore configuration to a server where the software is already installed and tested – generally a non-production system. When moving to a DR environment there are a couple of timings that need to be set in advance and agreed by the business:

- Recovery Point Objective (RPO)
  This is the point that the system will restore back to, generally when the backup was taken. Remember that if there are other applications that need to be recovered, teams may need to find a commonly agreeable RPO between all of the systems
- Recovery Time Objective (RTO)
  This is the amount of time it takes to recover the system from backup. In the case of critical systems, this should be a very short time.

When planning how the system would recover in a disaster, the RPO is used to determine what time(s) of day the organization will need to run backups. The RTO is them often used to determine how often backups will have to be created. For example, organizations with a very short RTO will often need to run more frequent non-cumulative backups to reduce the amount of time it will take to restore.

Many organizations already have “tier” RPO/RTO targets for systems with a particular exposure or importance, and these will often be applied to their file transfer systems as well.

High Availability

In addition to making systems recoverable, it is often important to make a system highly available, so that the loss of any particular component does not cause the loss of the entire system. This is achieved by replicating everything that happens on the file transfer server into another server. Every single point of failure needs to be duplicated so that if any part fails, the rest can continue. If the file transfer solution

uses a database, then it should be clustered or at the very least mirrored, and the data filesystem needs to be replicated too.

The distance between servers plays an important part in planning an HA solution. While it may be possible to provide asynchronous database clustering between servers located a few miles apart, it would hardly be possible if they are in different continents. The ability of a file transfer solution to be highly available is of course dependant upon the solution – some applications simply do not allow it.

When setting up an HA solution, it is also important to decide whether the servers will function in an active:active ("AA") or active:passive configuration. In active:active configurations both servers will normally receive requests distributed to them via a load balancing device, whereas in active:passive all requests are directed to the active server. (In active:passive configurations the passive server listens to a 'heartbeat' from the active server and becomes the new active server when the old one becomes unavailable.)

Generally speaking, the application software will determine whether it can handle active:active or active:passive. However, it is possible to turn an active:active application into an active:passive setup by simply configuring the network load balancing device to only use one node. The benefit of this configuration is that the two nodes don't necessarily have to have a fully synchronous connection between them – small delays in database or filesystem replication can occur. (To switch the passive server to active the load balancer is reconfigured manually, or knows to direct all traffic to the passive server if the active server is down for a certain number of minutes. In this scenario the passive server takes on the role of "Hot Standby" and normal service is resumed seconds after the load balancer makes the switch.)

X. Compliance

File transfers are frequently subject to regulations which often require certain levels of security to be maintained. The following are the more common regulations which need to be followed

List of main regulations affecting file transfers

FIPS (Federal Information Processing Standards).

These are public standards for use by U.S. government agencies. Most commonly, FIPS 140-2 is listed as a requirement – this is a requirement for evidence of tampering and a certified cryptographic standard. FIPS 140-2 applies to software – if it cannot be attained by your current software, you must change to a compliant system instead.

GDPR (General Data Protection Regulation).

GDPR is a set of regulations designed to protect the rights of all EU citizens and any data held about them. GDPR makes it illegal to process (including storage and transfer) data about EU citizens unless a restrictive set of criteria is met. For example, all personal data must be encrypted both at rest and in transit, and workflows must be clearly detailed and recorded.

GLBA (Gramm–Leach–Bliley Act).

This act relates to potential mergers (generally in the realm of finance), but contains provisions to safeguard collection, storage and disclosure of customer personal information. It relates to U.S. companies and requires that data being transmitted must be sent and stored securely

HIPAA (The Health Insurance Portability and Accountability Act).

Title 2 of this act ("Administrative Simplification") applies to electronic healthcare transactions. Specifically, the legislation contains detailed security safeguards applicable to administrative, physical, and technical access to data.

ISO 27001 (Information security management systems).

This standard provides for certification of information security management systems which meet the standards criteria. ISO 27001 does not apply to any specific industry. ISO 27001 contains 114 controls which must be implemented

PCI-DSS (Payment Card Industry Data Security Standard).

This is an information security standard applicable to the major well-known credit cards (Visa, Mastercard etc). PCS-DSS contains 12 requirements to be adhered to.

SOX (Sarbanes-Oxley Act)

This act applies to any company listed on the U.S. stock exchange. Although at a high level it is only relevant for finance and accounting, section 404 of the act directly applies to computer systems and validates that checks and controls are in place and working correctly (especially with regard to security systems).

…

Appendix A: Protocol Selection Guide

Use the following guide to help select which file transfer protocol is best for your use case.

	FTP	FTPS *(SSL/TLS)*	SFTP *(SSH)*	SCP *(SSH)*	HTTP	HTTPS *(SSL/TLS)*	AS2
Upload or download	Both	Both	Both	Both	Both[2]	Both[3]	Up
Secure in transit	No	Yes	Yes	Yes	No	Yes	Yes
Uses a single port	No	No	Yes	Yes	Yes	Yes	Yes
Default port (TCP)	21	21 (explicit)[4] 990 (implicit)	22	22	80	443	80 or 443
Supports custom commands	Yes	Yes	No	No	Yes	Yes	No
Commonly supported by servers	Yes	Yes	Yes	No	Yes	Yes	No
Supports strong ("two factor") authentication	No	Yes *(certs)*	Yes *(keys)*	Yes *(keys)*	No	Yes *(certs)*	Yes *(certs)*
Native Windows client	Yes	No	No	No	Yes	Yes	No
Native *nix/Mac client	Yes	No	Yes	Yes	Yes	Yes	No
Native Mobile client	No	No	No	No	Yes	Yes	No
Native Windows server	Yes	Yes	No	No	Yes[2]	Yes[3]	No
Native Linux/Mac server	Yes	No	Yes	Yes	Yes[2]	Yes[3]	No
Easily scripted or automated	Yes	Yes	Yes	Yes	No[5]	No[6]	Yes
Transfer resume (after interruption)	Yes	Yes	Yes	Yes	Yes[7]	Yes[8]	Yes

[2] Many file transfer servers support HTTP uploads, but any HTTP server supports downloads
[3] Many file transfer servers support HTTPS uploads, but any HTTPS server supports downloads
[4] See the "FTPS" section above for more information about "implicit" and "explicit" modes
[5] HTTP downloads are often scriptable, but parsing web pages for file lists and scripting uploads can be difficult
[6] HTTPS downloads are often scriptable, but parsing web pages for file lists and scripting uploads can be difficult
[7] HTTP downloads generally support resume, many servers that support HTTP uploads also do upload resume
[8] HTTPS downloads generally support resume, many servers that support HTTPS uploads also do upload resume

Appendix B: Acknowledgements

The CFTP curriculum would not have been possible without the generous contributions of the members of the CFTP Advisory Board. They include Van Glass of JSCAPE, Pam Reid of Coviant, Ben Spink of CrushFTP and DataExpress, Kevan Bard of Standard Networks and Ipswitch, Doug Papenthien of Rhinosoft and Solarwinds, John Kiernan of EDS and FIS Global, Clifton Gonsalves of bTrade, Michael Ryan of South River Technology, and Andy White of File Transfer Consulting.

Special thanks are also due to Tony Lloyd, CFTP and Ian Percival, CFTP, who piloted the CFTP training with the entire transmissions staff of BNY Mellon.