**Encrypted Transfer Automation Documentation**

**Purpose**

The purpose of this document is to describe the process in which an automated secure file transfer may be established between the application and a client’s FTP server.

**support@soverance.net**

Contents

[**Pre-Requisites** 2](#_Toc527634755)

[**Client Onboarding** 3](#_Toc527634756)

[STEP 1 : Obtain access to the client’s FTP server 3](#_Toc527634757)

[STEP 1a : Generating SSH keys 3](#_Toc527634758)

[STEP 1b : Generating a .ppk file 4](#_Toc527634759)

[STEP 2 : Obtain and install the client’s PGP public key 5](#_Toc527634760)

[STEP 3 : Provide the client with a Soverance Studios PGP public key 5](#_Toc527634761)

[STEP 4 : Verify PGP Encryption Capabilities 6](#_Toc527634762)

[**Automation** 7](#_Toc527634763)

[STEP 5 : The SoverancePgp PowerShell Module 7](#_Toc527634764)

[STEP 5a : Add-ClientEncryption Function 7](#_Toc527634765)

[STEP 5b : Remove-ClientEncryption Function 7](#_Toc527634766)

[STEP 5c : PGP passphrase file. 8](#_Toc527634767)

[STEP 6 : The Client Sync File 8](#_Toc527634768)

[STEP 7 : Task Scheduling 9](#_Toc527634769)

[**End-User Documentation** 10](#_Toc527634770)

[STEP 8 : Uploading to a Client’s FTP server 10](#_Toc527634771)

[STEP 9 : Downloading from a Client’s FTP server 10](#_Toc527634772)

# **Pre-Requisites**

To manage or modify this automation process requires the following prerequisites:

* Access to the Soverance/Powershell GitHub repository
* An administrative account in the Soverance domain
* The client’s public PGP encryption key
* Both the private and public PGP encryption key for “support@soverance.net”

This automation process makes use of the following dependencies:

* PowerShell
  + (*version 5 or later*)
* WinSCP .NET Assembly
  + *For connecting to SFTP servers via SSH with PowerShell*
* GnuPG executable
  + *For PGP encryption. You can get gpg.exe from a number of sources, but I prefer Gpg4win @* [*https://www.gpg4win.org/*](https://www.gpg4win.org/)*, as it includes additional software packages useful on Windows machines*

# Client Onboarding

## STEP 1 : Obtain access to the client’s FTP server

Before you can proceed with any automation, you must first obtain a user account with which to access the client’s FTP server. The specifics of this process are usually contained within some sort of File Transfer Agreement we make with the client, and often involves a phone conversation or email discussion with their technical team. It will be within this process that you obtain the following information:

* **Client FTP Server IP Address or Hostname**
* **Client FTP Server Port**
* **Client FTP Server Username**
* **Client FTP Server Password**

Additionally, you should take this time to agree with the client upon a suitable folder structure for which to pick up and drop off files. Generally, these would be something like “Inbound” and “Outbound” folders located on the client’s FTP server. If you intend to automate this connection, you should create these same folders within the client’s folder.

* **Inbound**
  + Soverance to Client communications, where files uploaded by Soverance are to be retrieved by the client. *e.x., /Files/Client/Automation/Inbound/*
* **Outbound**
  + Client to Soverance communications, where the client would place files for Soverance Studios to download. *e.x., /Files/Client/Automation/Outbound/*

Finally, you should ask the client whether or not a “whitelisting” process must occur on their firewall for Soverance Studios to successfully access their FTP server. Generally, if Soverance Studios must be whitelisted to access the client’s resources, the client’s technical team will require one or more of Soverance’s external IP addresses. You may provide them with the following external IP addresses in the event they must be whitelisted in the client’s firewall:

* **Atlanta Office –** xxx.xxx.xxx.xxx

### STEP 1a : Generating SSH keys

In some cases, as is the case with United Airlines, the client’s FTP server will require the use of an SSH key for authentication instead of a simple password. In most cases, Soverance Studios would generate this key pair ourselves, and provide the public key to the client. The client would then install the public key on their server and link it with our user account. The fact that a private key is installed on a Soverance Studios machine that matches the public key installed on the client’s FTP server is what allows us to authenticate.

* + If the client requires the use of an SSH key for authentication to their FTP server, a member of Soverance Studios must generate a new SSH keypair, and provide the public key to the client. **Please note that each SSH keypair we generate must be specific to each client.** For security purposes, no two clients should ever be given the same SSH key.
  + The easiest way to generate a new SSH keypair is to onboard a user for that client to CompleteFTP. Once the external client user can access the FTP server, you can use the CompleteFTP Manager application to generate a new SSH keypair for that user from within the ***User Properties -> Authentication -> Public Keys*** section. Importantly, using this method will store the generated keypair in the CompleteFTP database so that it may be stored for future reference.
    - Here, you will create a new RSA keypair at 2048 bits.
    - During creation, you will be prompted to save the private key as a file in a location of your choosing. Store this private key file in a secure location, as it must be installed on the machine from which you intend to access the client’s FTP server.
    - Once the private key is saved, you will be prompted to enter a password. This password is required to use the private key, and a user who attempts to use the private key to access a server will be prompted with a password dialog. Essentially, this is two factor security, being a piece of information you have (*the keys*) and a piece of information you know (*the password*). It is recommended that you store this password in a safe place, along with your private key.
    - Once created, export the selected key in SECSH format, and save the file in a location of your choosing. It does not require a file extension. **It is this public key file which you will provide to the client.**

### STEP 1b : Generating a .ppk file

* + To actually use these keys with an FTP program such as FileZilla, they may need to be combined into a single “key file”. This “key file” often has a “**.ppk**” extension, to be used with the PuTTY terminal emulator. You can create these .ppk files by using the PuTTY Key Generator application (***puttygen.exe***) located along with your PuTTY install.
    - Navigate to your PuTTY install directory and run “**puttygen.exe**”.
    - Click the “**Load**” button to load an existing private key file
    - Select the private key that you exported from CompleteFTP during the previous step.
    - Enter the private key’s password when prompted.
    - With the key now loaded into puttygen, modify the key comment to be descriptive for this client.
    - Finally, hit the “**Save Private Key**” button to save the compiled .ppk file.
  + You can then use this .ppk file in an FTP application, such as FileZilla.
    - Set “Protocol” to “SFTP – SSH File Transfer Protocol”
    - Set “Logon Type” to “Key file”
    - Specify username (*usually provided by client*)
    - Specify the .ppk key file created in the previous step.
    - Upon connecting to the server, you will be prompted for the password to the private key. You may choose to save this password for the duration of your session.

**Verify that you can successfully logon to the client’s FTP server before proceeding.**

## STEP 2 : Obtain and install the client’s PGP public key

Before you can proceed with any PGP encryption process for this client, you must obtain their public PGP key. This public key is usually provided to us (*or requested*) during a conversation with their technical team. **We will never have any reason to obtain the client’s private PGP key**.

We require the client’s public key so that we may sign files with it during the encryption process. This signing process allows the file to be decrypted using the private key associated with this public key signature. With the client’s public key properly signed as the recipient of an encrypted file, they will be able to use their private key to decrypt the file.

This PGP public key is usually provided to us by the client as a file with an “**.asc**” extension. This .asc file must be imported into the PGP keyring of your PGP program. The PGP program can vary per user (*as some developers may have their own preference of PGP programs*). If the process is intended to be automated via CompleteFTP, the public key must be installed on the Sov-FTP (CompleteFTP) server. On UA-FTP, we have **Gpg4win** already installed, which conveniently includes the **Kleopatra** GUI software to manage the PGP keyring.

* On Sov-FTP, open the “Kleopatra” certificate manager GUI application from the Start Menu.
* Select the “Import…” option.
* Select and import the .asc file provided to us by the client.
* Choose to “Certify” this certificate, proving that you trust the identity of the user who provided you with this certificate.
* **The client’s public PGP key is now installed and ready for signing.**

## STEP 3 : Provide the client with a Soverance Studios PGP public key

The client may need to perform the same encryption signing process for us – allowing their files to be encrypted and signed with our public key, so that when we receive the files we may decrypt them using our own private key.

We generally provide the same public key to every client, as this simplifies the encryption process on our end (*allowing us to easily decrypt files from any client, using a single private key*). All files encrypted by Soverance Studios with the intent to send to a client should use this PGP key. This key was generated using the [support@soverance.net](mailto:support@soverance.net) email address, and is already available for automation on UA-FTP.

If you need to provide this file to a client, you may do so via email. You will find a copy of the .asc file @

**H:\Departments\Technology\IT\PGP\support@soverance.net.asc**

Please note that if you wish to install the Soverance Studios PGP key on any other workstation (*such as your company laptop*), you would need to install to your PGP keyring the “support@soverance.net.gpg” file, located in the same directory. The .gpg file also includes the Soverance Studios PGP private key.

The passphrase for this private key is “**REMOVED FOR SECURITY**” (*no quotes*).

## STEP 4 : Verify PGP Encryption Capabilities

In terms of best practice, you should verify that the encryption relationship between Soverance Studios and a client works successfully before attempting to automate it. Generally, this verification occurs by simply trading encrypted files and attempting to decrypt them:

* Manually encrypt a file using the “**support@soverance.net**” PGP key, and specify the client as an authorized recipient.
* Send the encrypted file to the client, and verify that they are able to decrypt it successfully.
* Request the client encrypt a file, using our public PGP key as an authorized recipient.
* Obtain this file from the client and attempt to decrypt it.

If we can both decrypt files that were encrypted by the other party, then our PGP encryption capabilities with this client can be deemed verified.

At this point, you could stop, and an employee of Soverance Studios could process all file transfers manually. They would need to manually encrypt/decrypt files, and manually connect to the client’s FTP server. This workflow may be acceptable in some cases.

As the public/private key structure is often too complicated for the average user, we can automate this process with CompleteFTP to simplify file transfers between our employees and this client.

# Automation

## STEP 5 : The SoverancePgp PowerShell Module

The SoverancePgp PowerShell Module (**SoverancePgp.psm1**) is a custom module designed to automatically encrypt or decrypt files for a specified client. In order to accomplish this, it depends on the gpg.exe command line tools and the PGP keyring infrastructure installed on the machine where this module runs. You must have GnuPG installed on whatever server runs this module (*GnuPG is included as part of the Gpg4win software*).

The SoverancePgp PowerShell module is capable of being extended to produce different encryption results. Refer to the gpg.exe documentation for more: <https://www.mankier.com/1/gpg>

### STEP 5a : Add-ClientEncryption Function

The “Add-ClientEncryption” function is designed to encrypt all files in a specified directory for the specified client recipient. It signs the files using the “support@soverance.net” PGP key. This function accepts four (4) parameters as input:

* **FolderPath**
  + *The directory path which contains the files you wish to encrypt*
  + *Mandatory!*
* **Recipient**
  + *The client email address associated with a public PGP key which is already installed in the keyring. See* ***STEP 2*** *for more details.*
  + *Mandatory!*
* **GpgPath**
  + *The absolute path to the gpg.exe executable.*
  + *If this param is not provided, it defaults to the UA-FTP install location @ “C:\Program Files (x86)\GnuPG\bin\gpg.exe”*
* **PassphraseFile**
  + *The absolute path to the passphrase file for “support@soverance.net” PGP private key.*
  + *If this param is not provided, it defaults to the UA-FTP install location @ “C:\Scripts\Your-Passphrase-File.txt”*
  + *See* ***STEP 5b*** *for more details.*

### STEP 5b : Remove-ClientEncryption Function

The “Remove-ClientEncryption” function is designed to simply remove all encryption from any files in the specified directory. The files must have been encrypted by a client and signed with the “support@soverance.net” PGP public key as a valid recipient. This function accepts two (2) parameters as input:

* **FolderPath**
  + *The directory path which contains the files you wish to encrypt*
  + *Mandatory!*
* **GpgPath**
  + *The absolute path to the gpg.exe executable.*
  + *If this param is not provided, it defaults to the Sov-FTP install location @ “C:\Program Files (x86)\GnuPG\bin\gpg.exe”*

Unlike the Add-ClientEncryption function, Remove-ClientEncryption does not require any PGP keys to be specified, and therefore does not require a passphrase file either. The gpg.exe process will retrieve the appropriate key from the keyring based on the file’s recipient signature. In this case, the recipient should always be “support@soverance.net”, and since the public and private keys for this user already exist on our keyring, the files are decrypted without further question.

### STEP 5c : PGP passphrase file.

The PGP passphrase file is a simple text file that contains the password used for the Soverance Studios PGP private key. You can find an example of this file on the H:\ drive @

H:\Departments\Technology\PGP\Your-Passphrase-File.txt

If you are running the SoverancePgp PowerShell Module on a machine other than Sov-FTP (*CompleteFTP*), then you must specify the -PassphraseFile parameter when calling Add-ClientEncryption. You will need to create or copy or this file into a working location of your choice.

The only reason this file is necessary is to avoid the password prompts from gpg.exe during the encryption process. With the file’s existence, we can specify that the encryption process redirect it’s standard input stream to read from this file (*thus passing the password along with the other arguments to gpg.exe*). For whatever reason, gpg.exe will not accept a string parameter as a valid password argument despite the documentation saying otherwise, and must instead use STDIN stream for headless automation.

## STEP 6 : The Client Sync File

The “client sync” file can be considered to be the master automation program for a client. This file will be specific to each client, as every file transfer automation process may be slightly different depending on our (*or the client’s*) needs and capabilities.

If you are onboarding a new client, you must create a new client sync file dedicated to that specific client (*or copy an existing one*). In general, the client sync file will need to accomplish the following tasks:

* **Connect to the client’s FTP server**, using the credentials obtained in STEP 1.
* **Encrypt and upload** any data from Soverance destined for the client.
* **Download and decrypt** any data from the client destined for Soverance.
* **Log all transfers** between Soverance and the client’s server for auditing purposes.

An example client sync file has been created, called **ClientSync.ps1**. You can find the source code for this client sync file on GitHub.

When considering the nature of this client’s required security protocols, their client sync file is arguably one of the most complex we’ll ever need, so it makes for a great example. A client with more lenient security requirements would likely have a simpler sync file. For example, it’s significantly easier (*in PowerShell*) to connect to an FTPS server on port 21 with a password than it is to connect to an SFTP server on port 22 with SSH keys. In this example, a client that allows FTPS and password authentication would require significantly less code within their sync file.

The ClientSync.ps1 file is well commented with great detail, in order to help more easily modify or create new client sync files. As a general overview, the client sync file performs the following tasks in sequence:

1. Import the SoverancePgp Module
2. Create the client sync log file, if it does not already exist
3. Load the WinSCP .NET assembly
4. Create a new WinSCP session that connects to United’s SFTP server
5. Create temporary “Inbound” and “Outbound” working directories, in order to avoid the gpg.exe limitation of poorly handing spaces in file paths.
   * “C:\temp\client\Inbound\”
   * “C:\temp\client\Outbound\”
6. Check for any files stored in the “Inbound” folder on CompleteFTP
   * If there is more than one file in the CompleteFTP “Inbound” directory…
     + Move everything from the CompleteFTP “Inbound” directory into the temporary “Inbound” directory
     + Encrypt everything in the temporary “Inbound” directory
     + Move only encrypted files from the temp “Inbound” directory back into the CompleteFTP “Inbound” directory – leaving the unencrypted files in the temp dir.
7. Synchronize the CompleteFTP “Inbound” directory with the “Inbound” directory on the client’s FTP server.
   * Log any uploads that occur
   * After uploading, delete the files from the CompleteFTP “Inbound” directory.
8. Synchronize the temporary “Outbound” directory with the “Outbound” directory on the client’s FTP server.
   * Log any downloads that occur
   * Remove all encryption from files in the temporary “Outbound” directory
   * Move all decrypted files into the CompleteFTP “Outbound” directory.
9. Close the WinSCP session
10. Delete the temporary directories and all files inside.

## STEP 7 : Task Scheduling

Once a client sync file has been created, it can be run on a schedule through the Windows Task Scheduler on Sov-FTP (*or any equivalent server cron task*). Simply configure a new task to trigger at an appropriate schedule, and to run the client sync file as it’s action.

You must use an administrative account to process this task, and you should run the task whether that user is logged on or not.

The Client Sync task schedule is to run once per day, starting at 8:00 AM. After triggered, the task will repeat (*in a new process*) every 1 hour for a duration of 12 hours. This schedule essentially allows 12 “opportunities” to automatically transfer data to/from this client during the workday, and is a good starting point for most client automation schedules. You may choose whatever schedule works best for you and the associates who will be handling this data.

The task’s Action should always be to Start a Program, pointing at the absolute path to powershell.exe

C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe

You will add the absolute path to the client sync script within the action’s arguments, like so:

-noprofile -noexit -executionpolicy bypass –file "C:\Scripts\PowerShell\Applications\WinSCP\UnitedSync.ps1"

# End-User Documentation

The automation process described above prevents Soverance employees from needing to worry about technical information such as FTP credentials, SSH keys, or PGP encryption. Instead, the average Soverance employee simply needs to have access to CompleteFTP, and understand how to use the “Inbound” and “Outbound” directories for their client.

## STEP 8 : Uploading to a Client’s FTP server

1. Login to CompleteFTP
2. Navigate to your client’s “/Automation/Inbound/” folder
3. Place any files you wish to upload inside this folder.

That’s it! You’re done! The upload automation process will take it from here.

The upload process occurs every hour, on the hour, during the workday. Your files will be automatically PGP encrypted before being uploaded. Depending on when you place the file into the “Inbound” directory, you may need to wait for a while before the file is processed.

## STEP 9 : Downloading from a Client’s FTP server

1. Login to CompleteFTP
2. Navigate to your client’s “/Automation/Outbound/” folder
3. Download any files you may need.

That’s it! You’re done! The automation process has already processed all the back-end stuff. Your files should be unencrypted and immediately ready for use.