# Automatically encrypting files before uploading them to a remote server

When you deliver files to a third-party SFTP server you cannot always trust that the storage on the other end is encrypted. PGP encryption solves that problem at the application layer: only the holder of the matching private key can ever read the file, regardless of how the server stores it. This article shows how to combine `FsWatcher` with `PGPEncryptFile` to automatically encrypt every file that lands in a local outbound folder and upload the encrypted copy to an SFTP server. The script runs persistently until halted.

## How It Works

`FsWatcher` monitors a local directory for new files. When a file appears, the callback calls `PGPEncryptFile` to produce an encrypted copy, uploads that copy via a named SFTP VFS, and then deletes the local temp file. The original unencrypted file is left untouched; you manage its lifecycle separately.

The recipient PGP public key path is retrieved from the AFT! Secrets store at startup, so neither the key material nor its location ever appears in the script source.

```
// PGP encrypt before upload
// Watches a local folder for newly created files, encrypts each one with a
// recipient PGP public key, uploads the encrypted copy to a remote SFTP server
// via a named VFS, then removes the local encrypted temp file. Runs persistently
// until halted.

var localWatchDir    = "/local/outbound";         // local folder to monitor for new files
var remoteDestDir    = "/remote/encrypted-inbox"; // remote destination for encrypted files
var sftpVfsName      = "your-sftp-vfs-name";     // named SFTP VFS in the VFS Library
var recipientKeyFile = GetSecret("pgp-pubkey-path"); // path to recipient PGP public key file

var watcher = new FsWatcher();
watcher.NotifyCreate = true;
watcher.NotifyWrite  = false;
```

```
watcher.NotifyRemove = false;
watcher.NotifyRename = false;
watcher.NotifyChmod  = false;
// Wait a moment before firing so newly written files are fully closed.
watcher.DelayBySeconds = 2;

var sftp = new VirtualFSByName(sftpVfsName);

try {
   // Watch only the top-level directory; subdirectories are not processed.
   watcher.WatchDir(localWatchDir, false);

   watcher.Start(function(evt) {
      if (evt.Event !== EVT_CREATE) {
         return;
      }

      var sourceFile    = evt.Object;
      var encryptedFile = GetTempFileName() + ".pgp";

      Log.Info("encrypting: " + sourceFile);
      if (!PGPEncryptFile(sourceFile, encryptedFile, recipientKeyFile)) {
         Log.Error("PGP encryption failed for: " + sourceFile);
         return;
      }

      // Append .pgp to the original filename so the recipient can identify
      // the algorithm and strip the suffix after decryption.
      var remoteName = ExtractName(sourceFile) + ExtractExt(sourceFile) + ".pgp";
      var remotePath = remoteDestDir + "/" + remoteName;

      Log.Info("uploading encrypted file to: " + remotePath);
      var r = sftp.ImportFile(encryptedFile, remotePath);
      if (!r.Ok()) {
         Log.Error("upload failed: " + r.ErrorMsg());
      } else {
         Log.Info("uploaded: " + remotePath);
      }

      // Remove the local encrypted temp file regardless of upload outcome.
```

```
        // The original unencrypted file is left in place; manage its lifecycle
        // separately based on your retention policy.
        DelFile(encryptedFile);
    });

    // Block here until an operator sends a halt signal via the AFT! web UI or API.
    WaitForHaltSignal();
    Log.Info("halt signal received, shutting down");
} finally {
    watcher.Stop();
}
```

`watcher.DelayBySeconds = 2`: this small delay prevents the callback from firing while the producer process is still writing the file. Without it, `PGPEncryptFile` might try to read a file that is only half-written. Two seconds is enough for almost all local writes; increase it if your producer writes large files slowly.

`GetTempFileName() + ".pgp"`: `GetTempFileName()` returns a unique path in the system temp directory. Appending `.pgp` is not strictly necessary for the encryption to work, but it makes the temp file recognisable if you ever need to inspect the temp directory manually.

`DelFile(encryptedFile)`: the encrypted temp file is deleted after every event, successful upload or not. Keeping it around would accumulate encrypted copies in the temp directory with no benefit.

## Storing the key path as a secret

Rather than hardcoding the path to the recipient's public key in the script, store it in the AFT! Secrets store. In the Admin UI, go to **Secrets**, create a new secret named `pgp-pubkey-path`, and set its value to the absolute path of the public key file on the machine running AFT!. `GetSecret` retrieves it at runtime without ever touching the script source.

## Encrypting for multiple recipients

`PGPEncryptFile` accepts a single public key file. To encrypt for multiple recipients simultaneously, concatenate all their ASCII-armored public key blocks into one `.asc` keyring file and pass that path. Any holder of a corresponding private key will be able to decrypt the result.

---

Revision #1
Created 12 April 2026 14:47:26 by DevTeam
Updated 12 April 2026 14:52:27 by DevTeam