FAST Protocol

(File Asynchronous Secure Transfer)

Design document changelog

Hossam Mabed

May 4th, 2020

Overview

This document describes the changes made in the final implementation stage of version 1.0 of the File Asynchronous Secure Transfer (FAST) protocols, as compared to the original design document submitted earlier. These changes vary from substantial alterations to protocol design for improved security, to minor changes, to clarifications on ambiguities in the previous document, to extra additional features not required by the original document.

Core Protocol Changes

This section describes the biggest changes made to the original design document, which make any implementations that used the original document no longer compatible. These are the most important changes to the original design document

Handshake Protocol

1. User and server IDs are now just one byte (a single uppercase alphabetic ASCII character), to account for the simulated network restrictions. All the message lengths are adjusted accordingly, and checks are made to verify that IDs are valid.
2. The Handshake protocol now includes data in the header, specifically the nonce, since it is no longer predictable (see below). The header should be the associated data used for symmetric encryption, with the header sent in plaintext before the encrypted payload.

Tunnel Protocol

1. The header now includes an extra field: one byte representing the user ID
    a. In the client, it's the current user ID
    b. In the server, it's the destination client's ID
    c. The ID is one byte inserted after the message length and before the nonce
        i. Message format now is: 2 bytes version + 4 bytes message length + 1 byte user ID + 16 bytes nonce
        ii. Total header length is now 23 bytes
    d. On the server side, the server simply drops any message that does not have the ID of the current session's user, and goes back to waiting for new messages. This is the first check the server makes after receiving the message.
    e. On the client side, if the received ID is not its own ID, it displays an error message to the user, does not continue parsing the message, and goes back to accepting new commands.

Both

1. The nonces sent using symmetric encryption, for both Handshake and Tunnel, are modified in the following way
    a. Nonces are always 16 bytes long
    b. The first 10 bytes are the message sequence number, treated the same as before
    c. The remaining 6 bytes are a random stream of bytes generated by a cryptographic pseudorandom number generator by the sender
2. During message verification, the receiver truncates the first 10 bytes of the nonce to check as the message sequence number, but the full nonce including the random bits is used for encryption and decryption.

Minor Protocol Changes

This section describes minor deviations from the original design document. These changes do contradict what was in the original design document, but they are minor and not as important as the previous changes. If an implementation does not have these changes, it would still be compatible with this version.

1. The server does not save the state of the session to file, but instead maintains session information in its state.
2. If a user attempts to login using a user ID not saved by the server, the server interprets this message as a sign-up message, and establishes a new user with that ID and password, if session creation is successful. All other Handshake protocol rules apply normally.
3. The client no longer stops waiting for server response after 5 minutes. Instead, the client waits for the server response indefinitely.
4. During Handshake, if the client receives a bad response from the server, it tries to send the initiation message again for 3 times. If after 3 trials, no valid response is received, the client stops trying to establish a session, and goes back to prompting the user for a new ID and password.
5. The server now sends one extra output for the command “END,” the string “success” encoded in utf-8
6. If the user inputs the command “END,” the client automatically exits the session and logs out the user, regardless of server response.

Clarifications and Extra Specification

This section describes clarifications to the design document. These do not contradict anything mentioned in the original design document, but instead provide additional clarification and specificity, if certain areas were ambiguous in the original design document.

Handshake

1. The client gets the server's public key from a file stored in its same directory, or specified by the user in a commandline argument
    a. The public key must be encoded in OpenSSH format
2. The server gets its private key from a file stored in its same directory, or specified by the user in a commandline argument
    a. The private key must be encoded in PEM format
3. For asymmetric public key encryption
    a. The exponent used is e=65537
    b. 2048 bit RSA is used

Tunnel

1. The client prompts the user to enter an ID and password, which are used to establish a session using Handshake Protocol
    a. If the user inputs a user ID longer than one character, the first character is used
2. The server stores the user data as a file in its root directory, specified in its script or by a commandline argument. The format of the user data file should be a valid Python expression for a Dictionary. The server also maintains the file in that format.
3. The client accepts user input for commands and arguments separated by whitespaces of any length
    a. A string enclosed in quotes is counted as a single argument, even if it includes whitespaces
4. On the server side, if an error is encountered during the authentication stage of processing (if the authentication/decryption fails, or if the message doesn't follow the format of the protocol), the message is dropped and no response is sent back to the client. However, once decryption and authentication are successful, for any errors that arise after that (e.g. an error in the execution of a command, an invalid command, or an invalid number of arguments), the server send an acknowledgement message with the error message, in the same format as described in the original document.

5. The client automatically checks that the command provided is a valid command before sending it to the server. If it is not, the client displays an error message to the user. However, the client does not verify the number of arguments. The number of arguments is verified by the server.
6. The command “UPL” will create the file if it doesn’t exist, and overwrite it if it does exist, provided the user is allowed to access/create that file.
7. Session keys are always promptly wiped from memory upon session termination in both client and server

Both

1. The current protocol version is 1.0 for both Handshake and Tunnel
2. All messages over the networks are sent as byte arrays
3. All strings are encoded using utf-8
4. All numbers are encoded with big-endian
5. Nonces are always 16 bytes long

File Management

The original design document omitted any description of how to manage user files and instead left it outside the scope of the protocol. This section describes the rules established in this protocol version for file handling and file management. This section is only relevant to the Tunnel protocol.

1. The server stores all user files and folders in a directory in its own root directory, or specified by the administrator using commandline arguments
    a. If the specified directory doesn’t exist, the server creates it
    b. If the server cannot access the directory or cannot create it, it prints an error message and exists
2. The server creates a new directory for every user ID, with the name equal to that ID, which acts as the root directory for that user and stores all their files and folders
    a. Once a user logs in, if the directory doesn’t exist, the server creates it
3. Users are not allowed to navigate away from their root directory or make any changes, deletions, or creations outside their root directory. The server handles any attempts to do so and sends an “Access Denied” error message to the client.
4. All commands to the server accept both absolute and relative paths
    a. Absolute paths are treated relative to the user root directory

b. Relative paths are relative to the current working directory in that session
c. Working directories are restarted to the user root directory on every session

Extra Features and Fun Add-Ons

This section describes additional features and add-ons, that are not required by the protocol design document, and do not break compatibility, but are provided in this implementation version.

1. The server can connect to multiple clients at once
    a. Once a new session is established using the Handshake protocol, the server forks a new process to handle the session using the Tunnel protocol, then the parent process goes back to waiting for new Handshake connections
    b. There can only be one session active per user at the same time. Trying to log in with the same user for simultaneous sessions can lead to undefined behavior.
    c. The client, as before, can only handle one user session at a time
2. Both the client and server exit if the session if they reach the maximum number possible of messages sent in that session, which is 2 to the power of the sequence number length, or $2^{10} = 1024$ messages
3. The client automatically capitalizes commands for the user, so commands are not case-sensitive. A user can type “gwd,” “GWD,” or “gWd,” and they would all work.
4. The client also keeps track of the current working directory of the current logged in user, which is updated on every successful command of “CWD,” and displays it to the user at all times at the shell prompt
5. The client supports arrow keys when inputting commands (e.g. the user can press the up key to retrieve command history and left and right keys to change characters already input)
6. The client displays the output in a fancy and colorful way (please check it out I spent a lot of time making it pretty lol)
7. The root implementation directory contains installation and running scripts to make the process of running the network, server, and client as easy and smooth as possible :)
    a. The client and server require the following Python modules to run
        i. PyCryptodome
        ii. ast
        iii. termcolor
        iv. colorama