

CS6238 Project 2 Report

Joel Wilhite, Patrick Nelson

Our project submission is a Java 8 client server application stack that implements the Secure Shared Storage architecture supporting multiple users using industry standard protocols and cryptography. The secure channel between client and server is established as a mutual TLS connection between client machine and server, with the ephemeral session ID being signed by the current user's RSA private key to authenticate the user in that TLS session. This establishes a secure channel with both the TCB+Application and user separately authenticated. This is also resistant against replay attacks, since the user authentication salt is ephemeral to the specific current TLS session.

The document store is implemented with the file data being AES256 encrypted, and the file AES key being encrypted with the server's RSA private key. This ensures that even if an attacker were to steal the ciphertext files, there exists no plaintext file to recover the AES keys from. File integrity is ensured by recording a HMAC-SHA256 HMAC using the file AES key for each file and storing this with the file metadata. This HMAC is checked on each file access to ensure file integrity at all times. The use of the AES key + SHA256 hashing ensures that the HMAC is cryptographically safe from all foreseeable attacks. The file permissions and ownership is maintained via separate metadata file that contains the encrypted AES key, the file HMAC, owner, and permissions.

Files are identified by their fileID, which is defined as the file name signed with the user's private key. This provides a globally unique ID that allows for multiple users to have files with identical names. This also provides attribution for files, as you can verify the file ID against known user public keys and derive the file signer.

Signaling between the client and server is accomplished with strings written to an SSLSocket object, and data transfer is simple strings with files being serialized to their Base64 representations. This allows for arbitrary file types.

Safe Delete is implemented as an atomic deletion of the keying material for the file as well as the ciphertext file itself. If file deletion fails for whatever reason, the whole transaction is reported as an error, allowing the user to take appropriate action. There does not exist a case where a ciphertext file is deleted but keying material is still present. This is to prevent a possible data recovery attack if the keying material were compromised.

Work distributions was as follows:

Joel + Patrick : architecture, data signaling format, platform selection

Joel: Document store (check-in, checkout, delegate, secure delete), AES implementation, TLS client|server shell, integration testing, debugging

Patrick: RSA signing and signature verification, HMAC generation, file ID generation, client side implementation, report draft, test cases draft, debugging

Project source and history can be found here: <https://github.com/TyphoidMary/cs6238p2>