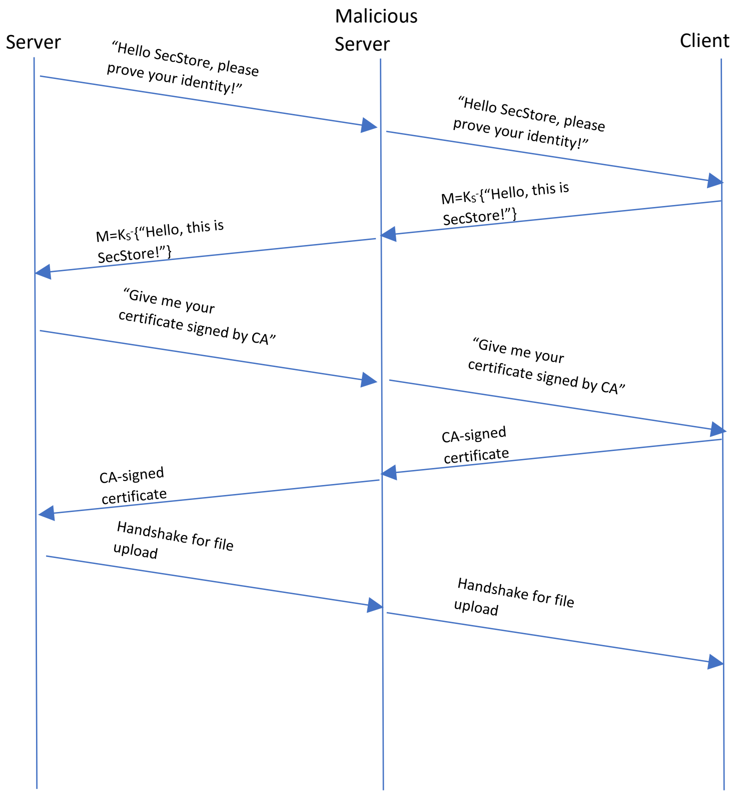
Poh Shi Hui 1002921

Chua Yong Teck 1003378

Reason for problematic protocol:

In the example AP provided with the project instructions to be discussed, this protocol fails when a malicious server interrupts the exchange and executes a replay attack.

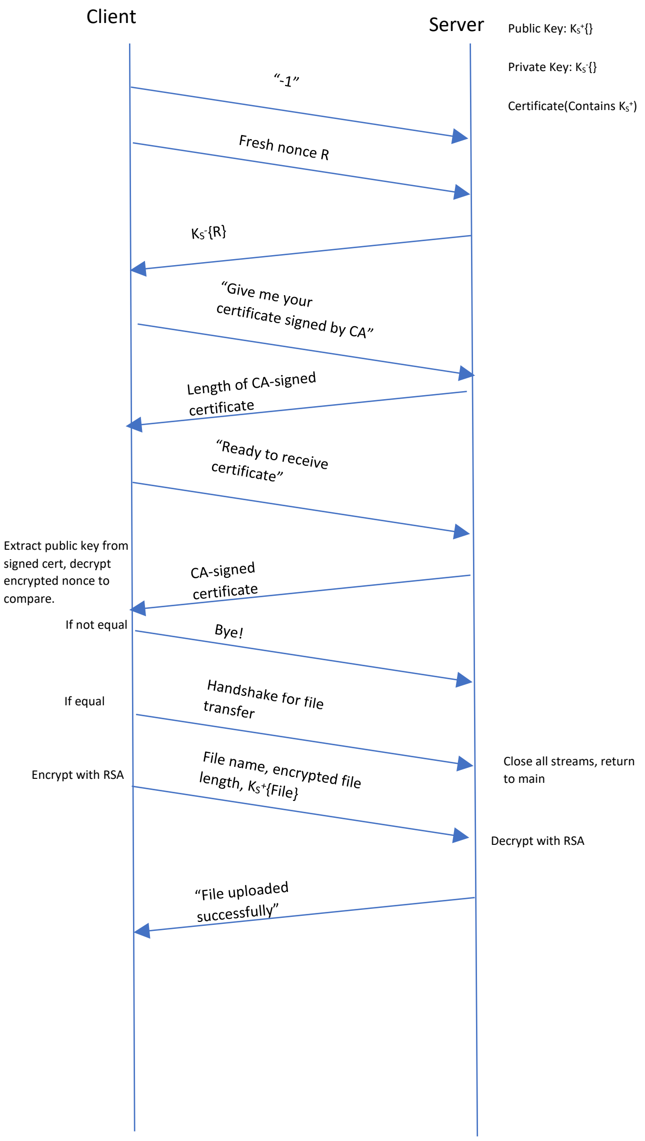
The malicious server can then trick the naive client into believing that they have successfully uploaded their data onto the server when the server has in fact not received anything. Although the malicious server does not own the actual server's private key to decrypt the information, the client must go through repeated redundant uploads that might never reach the actual server.


Server
Malicious Server
Client
“Hello SecStore, please prove your identity!”
“Hello SecStore, please prove your identity!”
M=K_S^-{“Hello, this is SecStore!”}
M=K_S^-{“Hello, this is SecStore!”}
“Give me your certificate signed by CA”
“Give me your certificate signed by CA”
CA-signed certificate
CA-signed certificate
Handshake for file upload
Handshake for file upload

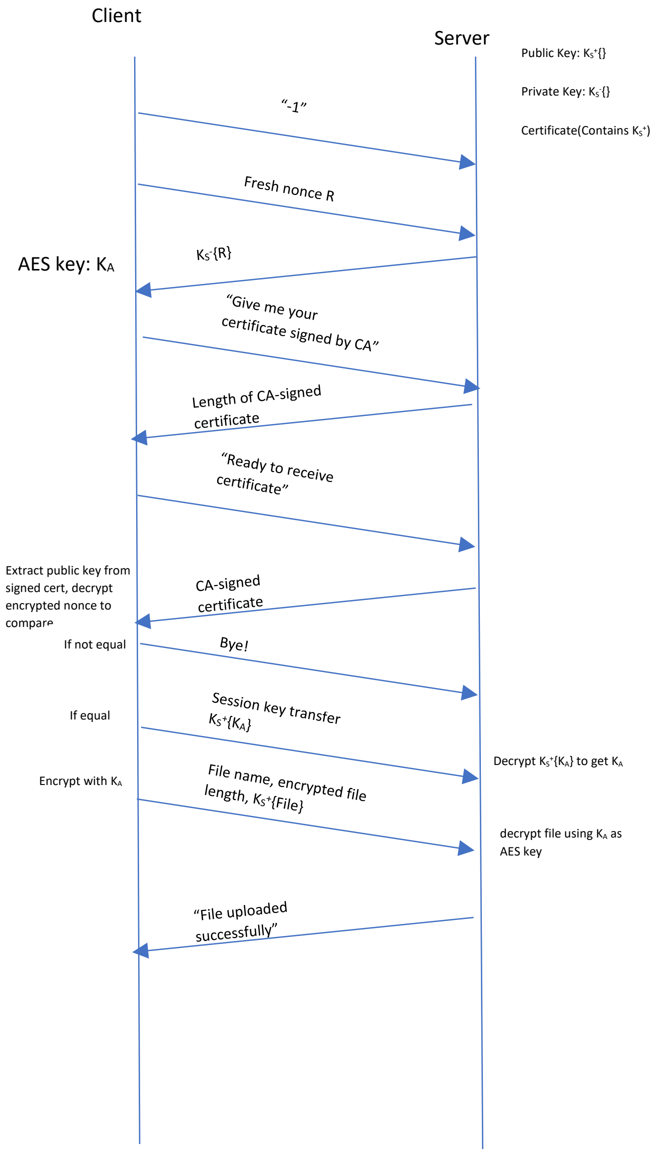
Our CP1 and CP2 implementation:

To resolve this issue, the client sends a freshly generated nonce (random int) to the server. The server encrypts this nonce with its private key and sends it back to the client. Since the nonce is different for every session, the malicious server cannot save the transmitted information to trick the client in future sessions.

CP1

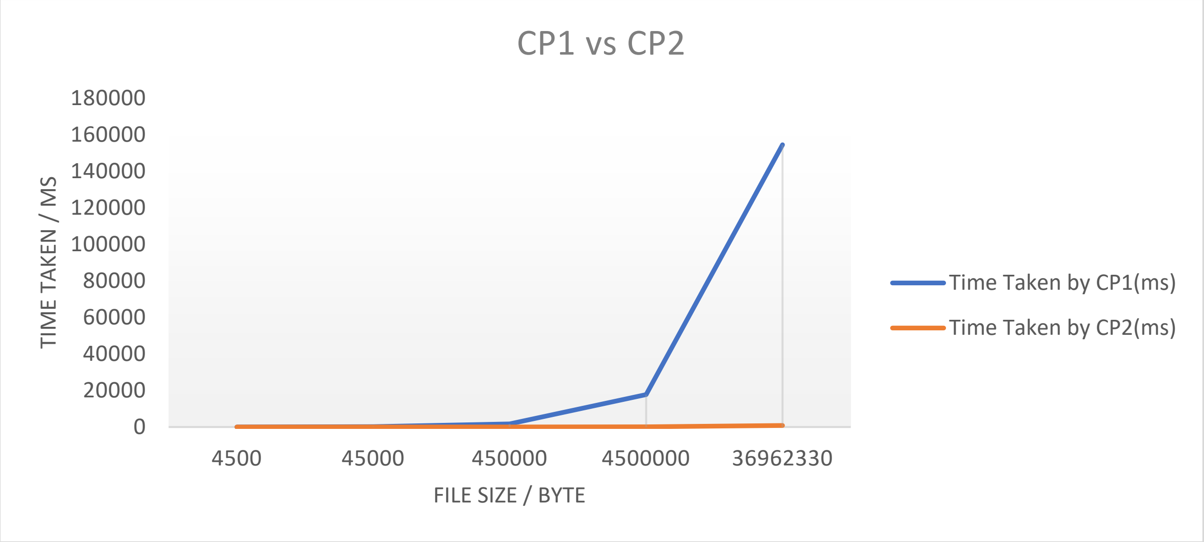

Client
Server
Public Key: $K_S^+\{\}$
Private Key: $K_S^-\{\}$
Certificate(Contains K_S^+)
"-1"
Fresh nonce R
$K_S^-\{R\}$
"Give me your certificate signed by CA"
Length of CA-signed certificate
"Ready to receive certificate"
Extract public key from signed cert, decrypt encrypted nonce to compare.
CA-signed certificate
If not equal
Bye!
If equal
Handshake for file transfer
Close all streams, return to main
Encrypt with RSA
File name, encrypted file length, $K_S^+\{File\}$
Decrypt with RSA
"File uploaded successfully"

CP2


Client
Server
Public Key: $K_S^+\{\}$
Private Key: $K_S^-\{\}$
Certificate(Contains K_S^+)
"-1"
Fresh nonce R
AES key: K_A
$K_S^-\{R\}$
"Give me your certificate signed by CA"
Length of CA-signed certificate
"Ready to receive certificate"
Extract public key from signed cert, decrypt encrypted nonce to compare
CA-signed certificate
If not equal
Bye!
If equal
Session key transfer $K_S^+\{K_A\}$
Decrypt $K_S^+\{K_A\}$ to get K_A
Encrypt with K_A
File name, encrypted file length, $K_S^+\{File\}$
decrypt file using K_A as AES key
"File uploaded successfully"

Throughput:

File Size(bytes)	Average Time Taken by CP1(ms)	Average Time Taken by CP2(ms)
4500	22.7724	1.0288
45000	188.1113	2.9713
450000	1741.6443	14.2488
4500000	17741.8838	113.18364
36962330	154446.8682	842.1843



CP1 makes use of an asymmetric RSA encryption for both certificate and data encryption. CP2 uses a symmetric AES encryption for the data, and RSA encryption for the certificate. The performance for each program can be seen above. CP1 and CP2 have linear relationship between average time taken for file transfer and size of file, but CP1 has a steeper gradient and thus a larger throughput that CP2