Rich Chen, Tai Thongthai

Design document outline:

- Overview of the application and its functional requirements
    - Authentication and security
        - The client sends the password to the server
        - Server blocks other users from viewing/accessing personal material
    - Files
        - The client can upload files to the server
        - The client can pull files from the server
        - The client can update files from the server
        - Removing files
    - Folders
        - The client can create a folder on the server
        - Can remove the folder from the server
        - List contents of the folder
        - Change the name of the folder
    - Navigation
        - Ask for the name of the current folder (directory)
        - Change directory

- Attacker models and trust assumptions
    - Trust assumptions
        - Impenetrable server
            - For this project we assume that the server is near impossible to hack, so we will not worry about encrypting the files stored on the server.
            - Furthermore, the server holds a public and private key used to authenticate user password when the user logs into the client. We also hold that the private key, public key, and user password stored on the server is encrypted and secured.
    - Possible attackers
        - Outsiders
            - Goals:
                - Get password of users
                - Get plaintext file of users
                - Edit plaintext file/folder of users
                - Upload files/folder not belong to them
                - Delete user files/folder
            - Capabilities:
                - Observe and modify the messages sent between client and server
                - Send messages to the server
        - Legitimate User (Other than self)

        - Goals
            - Log on to another user's account
            - Gain access to another user's file
            - Modify other people's files
            - Upload files to other user's server
        - Capabilities:
            - Observe and modify the messages sent between client and server
            - Send messages to the server

- Security requirements derived from the attacker models
    - Confidentiality:
        - the file contents have to be hidden from members in the server other than the personal user.
    - Integrity:
        - the file cannot be edited from anyone, including the server and other users on the server.
    - Authentication:
        - the user must be authenticated before he/she can reach his file/folder.
    - Replay detection
        - Replayed messages to be detected and discarded by the receiver.

- Protocol specifications
    - Overview:
        - At the start of each session, the client and the server will establish a shared short term session key through an implementation of **diffie hellman key exchange**.
        - All messages between the client and server will go through an authenticated encryption using our previously generated session key as the key. More specifically we will be using **CCM-CTR with CBC-MAC**. In this way, the message is guaranteed to have security, authenticity, and integrity. Lastly, to protect against replay protection we will use **implicit sequence numbering**. We will use this **sequence number as a nonce** during our CCM-CTR with CBC-MAC encryption.

    - Short term session key establishment

      Since the sessions are relatively short, and we would want to limit the use of long term keys (limit use of out of band channels), we will be using a new key for the protocols in every single session. Having a trusted third party is difficult to implement, so we will be using key-agreement instead of key transport. Thus we will be using an implementation of Diffie Hellman with signatures, which

will guarantee key-freshness, key-secrecy, key-authenticity, and forward secrecy.

Once the client has been examined and approved, the CA will send the public verification key to the client.

Client		Server
	----------[Client | g^x mod p] ------------------>	
		select random y compute g^y mod p
	<- [Server | g^y mod p | Sig_server(C|g^x|S|g^y)]-	
	--[Client | g^x mod p | Sig_client(C|g^x|S|g^y)]-->	
Compute k = g^(xy) mod p		Compute k = g^(xy) mod p

- Notations
  - **ENCk(m, N)** in this case means authenticated encryption using CCM-CTR with CBC-MAC on message m using key K and nonce N.
  - **S** means sequence number. **Sc** is the sequence number with the client as the sender, and **Ss** is the sequence number with the server as the sender.
  - **f** stands for file, signifying that the message pertains to actions regarding files.
  - **F** stands for folder, signifying that the message pertains to actions regarding folders.
  - **success** and **error** denotes the server's success or failure in executing the client's request. We shortened it in the schematic, but a specific success and error will be sent by the server in the implementation.

- Files
  - Uploading files

Client		Server
	---[client | f | upload | Sc | ENCk(file, Sc)]->	
		Checks sequence

		number, and MAC.
	<- [server | f | success | Ss | ENCk(success, Ss)] or <- [server | f | error | Ss | ENCk(error, Ss)] --	

- Pulling files

Client		Server
	-- [client | f | pull | Sc | ENCk(file, Sc)] --->	
		Checks sequence number, and MAC.
	<- [server | f | success | Ss | ENCk(file, Ss)] - or <- [server | f | error| Ss | ENCk(error, Ss)] ---	

- Updating files

Client		Server
	--[client | f | update | Sc | ENCk(file_a|file_b, Sc)] --->	
		Checks sequence number, and MAC.
	<- [server | f | success | Ss | ENCk(success, Ss)] --- or <- [server | f | error | Ss | ENCk(error, Ss)] -------	

*file_a is the target file to be updated, and file_b is the updated version of file_a to be overwritten on the server.

- Removing files

Client		Server
	-- [client | f | remove | Sc | ENCk(file, Sc)] ----->	
		Checks sequence number, and MAC.

	<- [server | f | success | Ss | ENCk(success, Ss)] -- or <- [server | f | error | Ss | ENCk(error, Ss)] ------	

- Folders
  - Create folder

Client		Server
	-- [client | F | create | Sc | ENCk(folder, Sc)] --->	
		Checks sequence number, and MAC.
	<- [server | F | success | Ss | ENCk(success, Ss)] -- or <- [server | F | error | Ss | ENCk(error, Ss)] ------	

  - Remove folder

Client		Server
	-- [client | F | remove | Sc | ENCk(folder, Sc)] --->	
		Checks sequence number, and MAC.
	<- [server | F | success | Ss | ENCk(success, Ss)] -- or <- [server | F | error | Ss | ENCk(error, Ss)] ------	

  - Update folder

Client		Server
	-- [client | F | update | Sc | ENCk(folder_a | folder_b, Sc)] --->	
		Checks sequence number, and

		MAC.
	<- [server | F | success | Ss | ENCk(success, Ss)] -- or <- [server | F | error | Ss | ENCk(error, Ss)] ------	

*folder a is the target folder, folder b is the updated version of the folder

- Pull folder contents

Client		Server
	-- [client | F | pull | Sc | ENCk(folder, Sc)] --->	
		Checks sequence number, and MAC.
	<- [server | F | success | Ss | ENCk(contents, Ss)] - or <- [server | F | error | Ss | ENCk(error, Ss)] ------	

- Folder Navigation

Client		Server
	-- [client | F | nav | Sc | ENCk(src|dest, Sc)] --->	
		Checks sequence number, and MAC.
	<- [server | F | success | Ss | ENCk(success, Ss)] - or <- [server | F | error | Ss | ENCk(error, Ss)] ------	

*src is the source/current folder and dest is the target location

- Arguments on why the security requirements are satisfied by the proposed protocols
  - confidentiality:
    - This is covered by using CCM-CTR mode with CBC-MAC where the user and server have the key to encrypt and decrypt. He sends the message to the client with this key.

- Integrity:
  - The authenticated encryption CCM-CTR mode with CBC-MAC provides security to make sure that files were not tampered with.
- Authenticity:
  - Since we are using Diffie Hellman, we can assure that the user and the server use a shared unique key. This ensures that the server and the client are really talking to each other during the session.
  - Furthermore, the messages sent between them are passed through authenticated encryption, which ensures authenticity of the messages.
- Replay Detection:
  - We use implicit sequence numbering. More specifically we use this sequence number as the nonce in the CCM-CTR mode with CBC-MAC, which ensures that the attacker cannot just come up with another sequence number.
- Non-repudiation:
  - Since we have implemented a form of Diffie Hellman, then we know that the user and server are actually talking to each other and no one else can interfere because they have the correct keys.
- Perfect forward secrecy
  - Diffie Hellman along with other forms of security provides perfect forward secrecy so that part of the key creation protocol is secure.