Section 1: System Design

1. Create User
    a. User prompted to enter a personal ID
        i. Must assert ID is not empty ('null') or already exists in our global Keystore(alert user if already exists).
            1. Possible Safety Vulnerability: Buffer Overflow Attack
                a. Mitigate by preventing user ID from exceeding a certain length, and prevent User from including/inputting non-alphanumeric characters.
    b. User prompted to enter a password
        i. Must assert password is not easily guessed (i.e. included in our easily guessed password list), nor that it is weak (i.e. too short or empty).
            1. Buffer Overflow Attack Possibility- Mitigate with a max size for password input, limit non-alphanumeric characters.
        ii. Require User to enter password a second time.
            1. Make sure that User remembers and has entered the password correctly.
2. Generate a Public and Private Key for this User(for clarity, we will name User Alice)
    a. Generate using PKEKeyGen() in Alice's Frame.
        i. Returns for Alice:
            1. Public Key(for All Users): PubKey_A
            2. Private Key(for Alice Only): PrivKey_A
            3. Error Margin Constant: error_key
3. Sign Alice's Public Key with Key Signature(Sign_A)
        i. Used in future verification of the validity of Alice's messages.
        ii. Generate using DSKeyGen().
            1. Returns for Alice:
                a. Signature creation key(for Alice only): Sign_A
                b. Signature verification key(for All Users): VerfSign_A
                c. Error Margin Constant: error_sgn
4. Send Alice's signed Public Key(PubKey_A) to the global Keystore along with Alice's Signature Verification Key (VerfSign_A)
    a. Keystore must trust Alice's validity on first use
        i. Check Public Key Signature's validity with Alice's VerfSign_A.
    b. If valid, add Alice's Public Key and Signature verification key to our global Keystore keychain, and issue a signature certificate.
    c. Monitor future Messages sent by User.
        i. If any change in user signature is detected(using DSVerifyKey()), we must revoke Alice's signature certificate, declare dead to all users, and dump Alice's Public key from the Keystore keychain.
            1. Generate new public and private keys for Alice as replacement, and store them in our global Keystore keychain.

1.Saving Files On Server

5. Sending 1st FILE from Alice to Datastore.
    a. Encrypt FILE with Alice's **Public Key created for this FILE only** (**each FILE for a user has its own Public/Private Key**).

Issue: Large File

i. If the FILE is large, break the FILE into parts to encrypt with separate keys, on accessing FILE in the future, concatenate all associated public keys **in the correct order** to get the whole FILE.
   1. Must take note of which keys are used and in what order, encrypt the array in a separate Public key in Datastore- open this Key first upon accessing the FILE in future.

Issue: Small File

ii. If the FILE is smaller than the available bit-space, pad the FILE completely with an identifiable string(ex. 000001).

b. Add signature to FILE using DSKeyGen().

c. Send the FILE to Datastore to save into our associated Public Key(s) for this FILE for Alice.

6. Save 1st FILE(sent from Alice) in Datastore[Main System Asymmetric Public Encryption].
   a. Verify signature on FILE, and assert that our current certificate for the Public Key associated with the FILE is valid (in our Keystore).
   b. Save Alice's encrypted FILE in Datastore with associated Public Key(s) from Alice.
7. Generate Symmetric Encryption Model "tunnel" between User and Datastore after 1st FILE communication for future **efficiency** in file sharing.
   a. Trust-on-first-use has now been established(in Asymmetric Public Model)
   b. Subsystems with Symmetric Encryption Model decrease runtime drastically for FILE sharing and saving.
      i. Asymmetric Public Key Encryption is very slow due to key generation, so we will only use it once to create an initial connection between source (User) and destination(Datastore).
   c. We will use AES-CTR to increase our efficiency, as our Main-system Asymmetric Public Encryption is very slow.
      i. Use chain of Block Cyphers, composed of:
         1. 'bit' or portion of data of our FILE
            a. Generate a Nonce(unique random Initialization Vector) for each separate bit of data in FILE.
            b. Assign counter- which records the position of FILE our bit is responsible for.
            c. Create a unique Key for this entire FILE only (or portion of FILE if FILE is large).
            d. Pad empty space(Security Precaution).
      ii. Encrypt with AES-CTR: XOR Block Cypher Encryption with Key, Nonce, counter and all individual 'bits' of the FILE. Returns FILE as [Nonce, Encrypted FILE individual Bits [C1,…,Cn] (n=CTR).
      iii. Decrypt with AES-CTR:
         1. Parse(bit-by-bit) Encrypted FILE into [Nonce, Encrypted FILE individual bits].
         2. Find Pi with XOR with Ci and output of Encrypt(Key) on Nonce and CTR.
         3. Concatenate all individual bits to produce the entire original FILE.

2.Sharing File with Other Users

8. Share FILE with another User(in our case, Bob). [Main System Asymmetric Public Encryption]

a. User Alice:
   i. Signs FILE with personal verification signature Sign_A.
   ii. Encrypts signed FILE using Bob's Public Key PubKey_B.
   iii. Sends encrypted FILE (of type string) over our unsecure network.
b. User Bob:
   i. Checks if Alice's Certificate for the personal key is valid, in Keystore.
      1. Dump FILE if certificate has been revoked or has expired.
   ii. Decrypts signed FILE using personal Private Key PrivKey_B.
   iii. Verifies signature of FILE using VerfSign_A(in Keystore)- to ensure FILE is truely from Alice and has not been tampered with.
c. After 1st Public Encryption Communication[Subsystem Symmetric AES-CTR Block Cypher]:
   i. Trust-on-first-use has now been established.
   ii. Generate Symmetric Encryption Model "tunnel" between User(Alice) and User(Bob).
      1. Same steps as sharing FILE with Database.

3.Revoking User Access

9. Keystore must detect changes in any User's signature during FILE transfer or storage into or call from Datastore.
   a. If change in user signature detected, immediately:
      i. Revoke Certificate(announce to All Users as well)
      ii. Remove corrupted Public Key and Signature.
      iii. Generate new Public/Private Key and Signature/Signature Verification Key for this FILE(corrupted key).
         1. Do not need to generate
         2. Save Private Key and Signature in User only, send Public Key and Signature Verification to Keystore.
      iv. Certify new Public Key and Signature Verification Key, and add to Keystore keychain with original data associated with FILE.

4.Efficient File Append

10. Adding (Appending) Content to FILE.
   a. For Large Files, our process works to first pull the array of [Nonce, Encrypted FILE individual bits (Ciphertext)] which would be concatenated to build up our complete FILE.
      i. Instead of accessing all keys of FILE, we can simply append after the final Cypher bit in array.
         1. If the final key gets filled, create new AES-CTR Block Cyphers for remaining FILE.
         2. Append new cypher segments into array, correct order is preserved due to the CTR variable.
            a. No padding required.
      ii. Process is efficient as, in the AES-CTR system, both Encryption and Decryption functions can be parallelized.

Section 2: Security Analysis

In a situation where valid User Alice is creating an account, or is sharing FILES(s) with valid User Bob, or is storing FILE(s) in Datastore; attacks can be conducted by Malicious User(*attacker*). Possible attacks may involve:

1. *Attacker* may **pretend** to be Alice while sharing with Bob:

   a. In order for Alice and Bob (two valid Users) to communicate securely, there must be some way for them to securely learn each other's Public Key.
   b. Prevention Method: Issue **signatures** with each key for FILE, and create global **Certificates** for these signatures indicating if they are valid, or if the signature is being forged by an *attacker*(must rely on principle of trust-on-first-use).
      i. Provide authentication for Signature by our User by issuing a 'Valid' Certificate for valid keys.
         1. If changes in Alice's key are detected, the certificate should be revoked immediately, preventing Bob from authorizing the false signature and opening the file.
2. *Attacker* may overflow input while creating a User ID or Password, which may allow *attacker* to leak values in our Keystore- responsible for storing Login/Password information(**Buffer Overflow Attack**).
   a. Prevention Method: Limit the string Length and character type User is allowed to give to User's ID.
      i. Prevent use of non-alphanumeric (unlikely to be used by non-malicious users).
      ii. Limiting input length would mean an *attacker* cannot create a Buffer Overflow 'jump'.
   b. Prevention Method: Require and Limit the Length User is allowed to input for Password.
      i. Short passwords can easily be guessed by *attackers*, so we must at least have a minimum length.
      ii. Extensively long passwords are likely attempts by the *attacker* to trigger a buffer overflow, rather than a valid user trying to create an account.
      iii. Limiting input length would mean an *attacker* cannot create a Buffer Overflow 'jump'.
3. In our Symmetric Key Subsystem, an *attacker* may try to **leak** our Initialization Vector, or Nonce, leading to catastrophic failure.
   a. Prevention Method: Unique Nonce(IV) for each bit-portion of FILE stored in the AES-CTR Cypher Scheme.
      i. In the standard 128-bit key size, it would take roughly 20 years to break security, so we can assume that we are protected against Brute Force Attacks.
         1. Add small delays to even further increase security from Brute Force Attacks.
   b. As a result, AES-CTR can be assumed to be IND-CPA Secure, and is efficient as Enc() and Dec() are both parallelizable.