CSL707
ASSIGNMENT-4

SUBMITTED BY:

Gaurav Mittal (2012CSB1013)

Kaushal Yagnik (2012CSB1039)

Deepak Chawla (2012CSB1010)

Design or Workflow

What encryption to use?

- Asymmetric encryption techniques like RSA(Rivest-Shamir-Adleman) encryption are quite secure, but it is very complex to compute due to their asymmetry. On the other hand, symmetric encryption techniques like AES (Advanced Encryption Standard) are much more fast to compute due to their symmetry but are less secure than their asymmetric counterparts.
- So, we decided to use a combination of both AES and RSA along with some randomization to maximize the security.
- First we randomly generated a symmetric key for the AES encryption and encrypted the entire file/data with that key. Then, we used RSA public key (again either randomly generated or supplied by the user himself) to encrypt the generated random AES key used for the encryption.
- Now, for any adversary to decrypt the file details, me must either of the following:
- 2 or more files which have been encrypted using the same AES key, using which he can make reasonable guesses about the content of the files. But

this scenario is highly unlikely as, AES keys are randomly generated and hence, no two files would be encrypted using the same AES key.

- The AES encryption key itself. But, since AES encryption key is also encrypted using the RSA keys which are asymmetric and known to be nearly unbreakable. Hence, the adversary cannot possibly not know the AES key as well.
- Finally, to retrieve/decrypt the file, first the AES key is decrypted using the RSA private key, and then this key is used to decrypt the data that was used with the AES algorithm.
- A small drawback of combining these two methods is that we need to store the (RSA encrypted) AES key along with the file/data that was encrypted. This is to know what AES key it should the data be decrypted with, after the AES key has the decrypted using the private key for RSA.

GUI

- A very simple and easy to use interface has been developed to use our application
- To login to the AWS account, the user is provided with 2 options:

- Choose the default login details mentioned in the config.properties file.
- Manually enter the keys into the space provided.
- Thereafter, on successful connection, the user needs to fill in the name of the bucket he wants to interact with. If the bucket name does not exist, it will be created for the user.
- Now, the user need to choose the RSA keys that should be used for the encryption/decryption process. At this point, the user can either opt for loading his own keys, or choose to generate and store new ones at any desired location. Also, we have provided customized extensions for the keys that will be used by our Java-App (i.e. ".pubs4" for public key and ".privs4" for the private key). This is done to ensure that there are no malicious attempts or mistakes by the user to use incompatible keys. Hence, the user will be able to view and select only the respective public/private keys.
- After providing the above parameters, the user must choose if he wants to upload or download a file or a complete folder.
- Then the corresponding upload/download operation is performed after encryption/ decryption

on the files to be transferred to/from the bucket on Amazon cloud.

- Here, each upload-download operation is spawned in a new separate runnable thread which provides better performance, especially in case of folders. Apart from alleviated performance, we can also constantly monitor the progress bar for individual uploads/downloads concurrently.
- This encrypted file/folder is stored as temp file in the /tmp directory which is then decrypted and stored at the appropriate location as specified by the user. After the decryption is over, the encrypted file is deleted from the /tmp directory to free space in the hard-disk.
- All the other details pertaining to the status of the operation being performed are constantly displayed in the status bar at the bottom of the App.