Project 2
Cryptography: encrypting files before storage in the cloud

Viraj Shah
N12149888

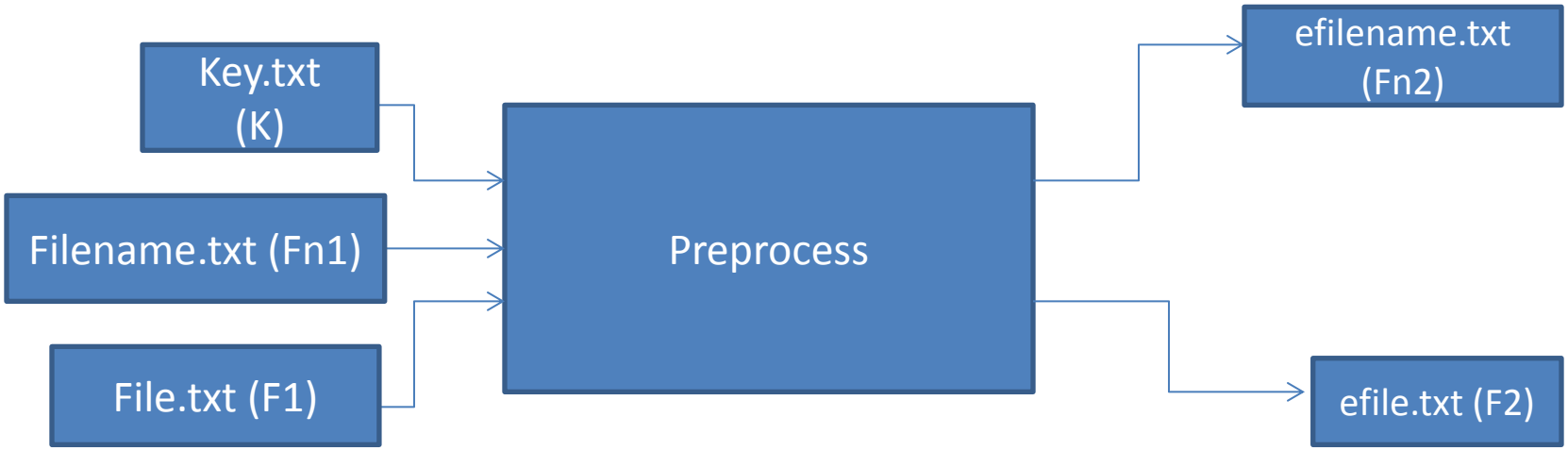
Vishal Shah
N13491580

Note: Work equally divided among both.

Primitive Schemes

- **AES encryption** scheme to ensure confidentiality of file contents
- Corresponding **AES decryption** scheme to recover encrypted data
- **HMAC** scheme to ensure integrity of data
- **Random hexadecimal generator** to obfuscate the filename
- **RSA** scheme for secure transmission of symmetric keys

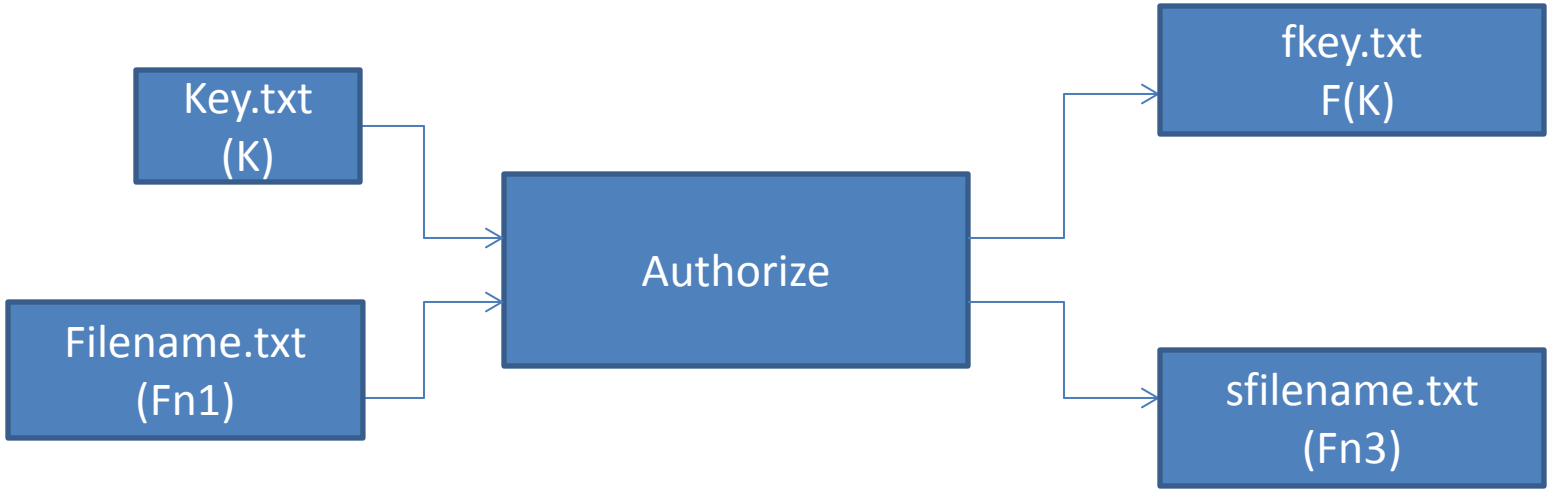
Preprocess Stage



- Preprocess takes as input: *key.txt*, *file.txt* (F1) with *filename.txt* (Fn1).
- It outputs *efile.txt* (F2) with *efilename.txt* (Fn2).
- The Key (***K***) and initialization vector (***IV***) are dynamically generated by the crypto++ library.
- *file.txt* (F1) is encrypted with AES encryption in CFB mode to generate *efile.txt* (F2).
- *eFilename.txt* (Fn2) is generated as follows:

  Fn1 XOR (random hex generator = ***KFN***), gives us Fn2.
- The Key ***K***, ***IV*** and ***KFN*** are now stored in *key.txt* file.
- We now generate HMAC of file *efile.txt* (F2) with key ***Kmac*** and store it in filename *hmac_efilename.txt*.
- The key ***Kmac*** for HMAC is also stored in the *key.txt* file.

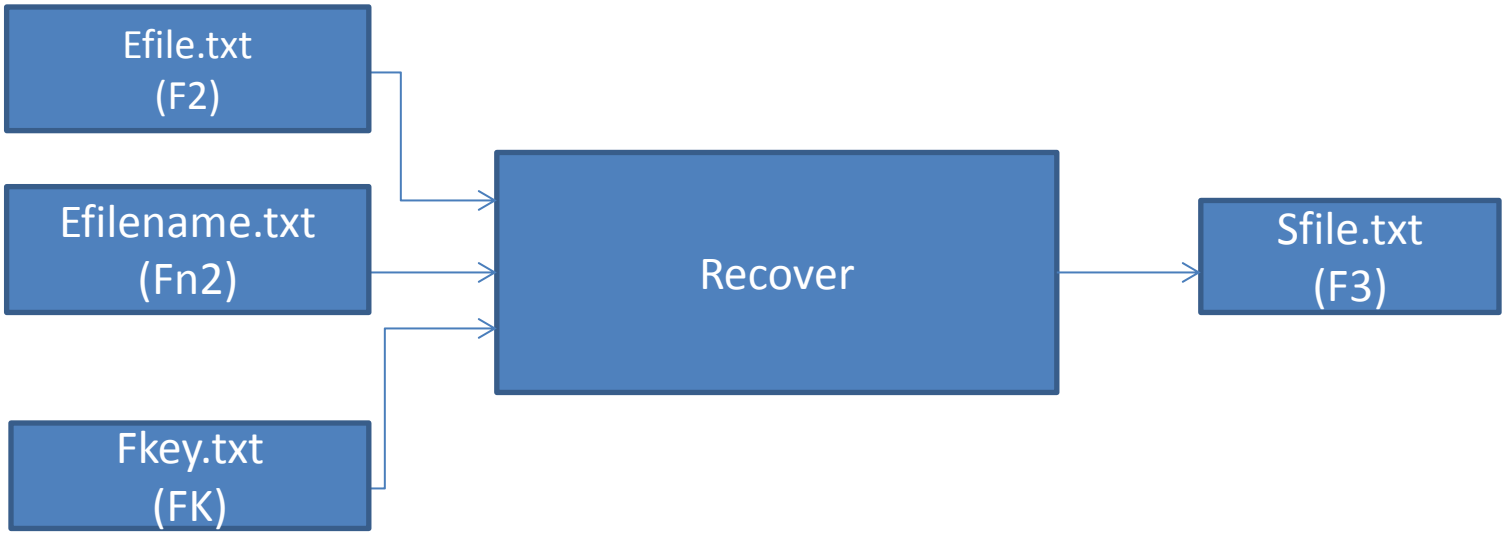
Authorize Stage



- Authorize Stage takes as input *Key.txt* (***K***) and *Filename.txt* (Fn1)
- It gives as output, *fkey.txt* (***FK***) and *sfilename.txt (*Fn3).
- We use random generator ***KFN*** from the *key.txt* file which we stored in the ***PreProcess*** stage.
- *Sfilename.txt* (Fn3) is generated by ***Fn1 XOR KFN***.
- ***FK*** is generated as follows

  Kmac + ***KFN*** +***[(K + IV) XOR Fn1]***

Recover Stage



- We transfer the symmetric key ***FK*** and ***HMAC*** of *efile.txt* to the user using asymmetric encryption that is via secure channel implemented with ***RSA***.
- We use this ***HMAC*** to verify for integrity by using the key ***Kmac*** stored in ***FK***.
- Once we verify that the integrity of the file is not compromised, we move forward with decryption of the file.
- Once the user receives the symmetric key he can now proceed with the decryption of the file.
- Decryption of the file is as follows:-

1. We first remove KFN from ***FK***.
2. ***KFN XOR Fn2 = Fn1***
3. Now, once we get Fn1, we can use
4. ***(K + IV) = F(K) XOR Fn1***
5. Once we derive ***(K + IV)*** we can now get *sfile.txt* (F3), plaintext file from *efile.txt*(F2) by using the decryption algorithm in AES crypto++ library.

How to run the Program

- Use the bash script provided
- bash script.sh
- You can enable Auto-execute (Y)
- To manually execute all commands (N)

1. ./PreProcess key.txt file.txt filename.txt
2. ./Authorize key.txt filename.txt
3. ./Recover efile.txt efilename.txt fkey.txt

- Please see next slide for screenshots

Used and tested on

- Linux Ubuntu 12.04/14.04 32/64 bit architecture
- Used Crypto++ library
- Used hexadecimal format

Output

```
vd@ubuntu:~/Desktop/Project2$ bash script.sh
Auto-excute [Y/N] N
Command Syntax:

        ./PreProcess key.txt file.txt filename.txt
        ./Authorize key.txt filename.txt
        ./Recover efile.txt efilename.txt fkey.txt

vd@ubuntu:~/Desktop/Project2$ bash script.sh
Auto-excute [Y/N] Y
Setting Up Environment
Environment Set
Preprocess Phase
Performing HMAC using SHA256 on file...efile.txt
Preprocess Phase complete
Authorize Phase
Authorize Phase complete
Recover Phase
Generating RSA keys...
Sending public key to user...
User encrypting fk and HMAC of uploaded file with public key...
Encryption Complete.
User now sharing the encrypted fk and HMAC of uploaded file over unsecure channe
l...
Peer receives the encrypted files.
Decrypting the files using his private key...
Checking HMAC of downloaded file ...
File authenticated and integrity maintained!
Recover Phase complete
```

Justification of Correctness

- AES scheme is used to encrypt the file contents and produce the encrypted file Efile.txt
- Only the user with the correct fkey.txt which contains the corresponding Key (K), IV and KFN file can decrypt the Efile.txt to get the plaintext file Sfile.txt
- This ensures confidentiality of file contents
- The filename (Fn1) is obfuscated using a random hex generator to get Efilename.txt (Fn2). This prevents leaking any information about the file.

Justification of Correctness

- During the PreProcess phase HMAC of file *efile.txt* (F2) with key ***Kmac*** and store it in filename *hmac_efilename.txt*.
- The key ***Kmac*** for HMAC is also stored in the *key.txt* file.
- RSA scheme is used to transfer symmetric keys FK and HMAC to the user in the recover phase via secure channel
- Once the user receives the symmetric keys, ***HMAC*** is used to verify for integrity by using the key ***Kmac*** stored in ***FK***.
- Once we verify that the integrity of the file is not compromised, we move forward with decryption of the file.
- This ensures a secure channel to transfer symmetric keys to the user using RSA
- Integrity of file contents are ensured using HMAC scheme.
- After decryption Sfile.txt == file.txt and

  Sfilename.txt == Efilename.txt

Justification of Privacy

- AES algorithm is used to ensure privacy of data.
- *Efilename.txt*(Fn2) is generated, which is different from the plaintext *filename.txt(Fn1)*. This has been done so that there is no leakage of information about the contents of the file.
- A function of key(***K***) and *filename.txt*(Fn1) is generated. This is stored in *fkey.txt*.
- Thus only those who posses the correct key and the respective filename will be able to decrypt the encrypted file *efile.txt*.
- *Filename.txt(Fn1)* is XORed with a random hex generator. This changes each time the function is run. This generates *efilename.txt(Fn2)* which leaks no information about the original filename.
- Thus the pair (f2,fn2) leaks no more information about f1 than the filename fn1 and the length of f1.
- In this manner privacy of data is ensured.

Justification of Integrity

- During the preprocess stage ***HMAC*** of the encrypted file, *efile.txt* is generated and stored in a text file, *hmac_efilename.txt**.***
- During the recover stage, ***HMAC*** of the *efile.txt* is verified using the key, ***Kmac*** received from the file *fkey.txt.*
- Only if there is a match do we proceed to decrypt the file.
- This detects if there was any modifications in the file uploaded in the cloud.

Extra Credit

- Please find “Shah-Shah-extracredit.pdf” PDF document attached.