Applied Cryptography

Project 2: Encryption for cloud storage

Cloud is untrusted

- The cloud has huge potential when it comes to storing, sharing and exchanging files, but the security provided by cloud services is questionable.
- Users, after uploading their files, have no control anymore about the way their data is handled and the location where it is stored.
- Even worse, users have no means to control access to their data. Considering both corporate and personal data which is often secret and sensitive in nature, one should not blindly entrust it to a cloud storage provider

Challenge of Cloud Storage

- To guarantee integrity, confidentiality and control over all stored data.
- Users should apply security-oriented cloud storage middleware that, on the one hand, keeps the usage of the cloud simple and fast, and, on the other hand, enforces the application of strong cryptographic algorithms and protocols in order to keep private and confidential data from being leaked.

SOME SOLUTIONS

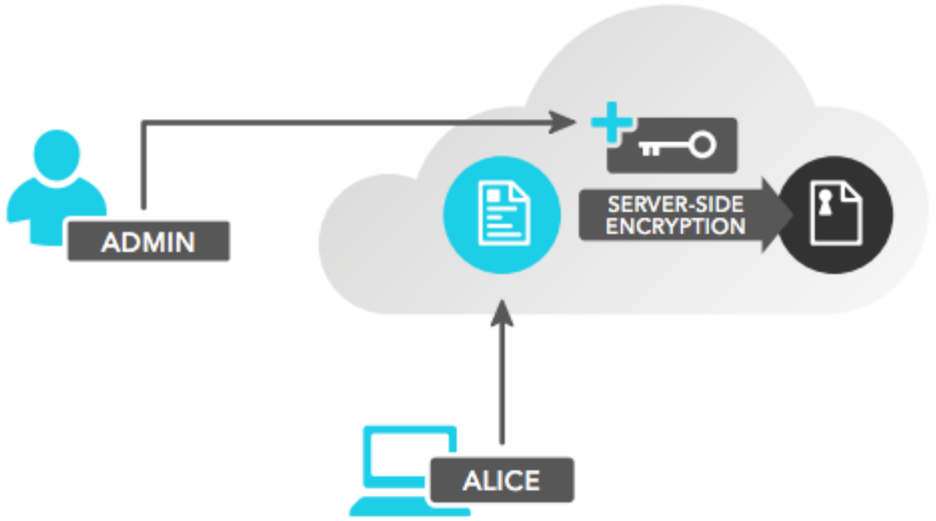
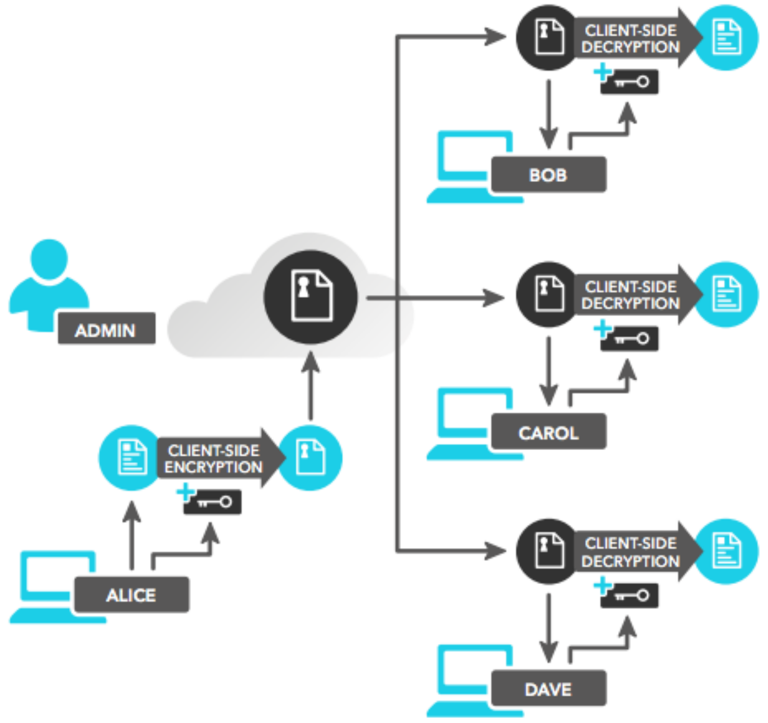


Figure 1 When using traditional cloud storage middleware, documents are visible in plaintext.

COMPLETELY SECURE CLOUD COLLABORATION



End-to-end Encryption

- Encryption and decryption are done on the client side. No entity is able to recover the data, except for the owner herself and users authorized by the owner.
- No trust in the cloud storage provider is required.
- Your data stays as safe as if it was stored securely on your own system.

Shared Files are Encrypted

- Files shared between users are still encrypted in the cloud at any time.
- Security and collaboration go hand in hand.

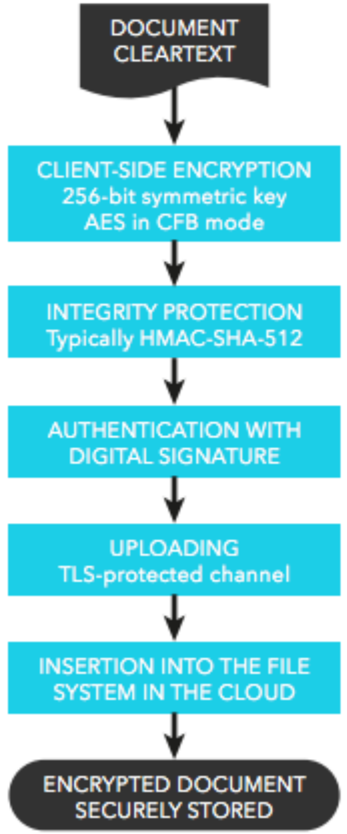
Shareable

The owner can invite anyone with ease to share files with.

Cloud Independent

- It is independent of cloud providers.
- It is possible to store the file on the cloud of the user’s choice, or even to distribute it among several clouds.

Technology


DOCUMENT CLEARTEXT
CLIENT-SIDE ENCRYPTION 256-bit symmetric key AES in CFB mode
INTEGRITY PROTECTION Typically HMAC-SHA-512
AUTHENTICATION WITH DIGITAL SIGNATURE
UPLOADING TLS-protected channel
INSERTION INTO THE FILE SYSTEM IN THE CLOUD
ENCRYPTED DOCUMENT SECURELY STORED

Algorithms and Libraries

- File Name is encrypted using SHA256 hashing algorithm. Library is available on http://create.stephan-brumme.com/hash-library/
- Files are encrypted using AES256 encryption algorithm. C code is available on http://www.literatecode.com/aes256

PreProcess(Original File, PassKey)

1. In this step, File is encrypted by AES 256 algorithm and new file is created with a different name.
2. The name of new file is calculated as SHA256 hash of the original file name + passkey and hence, name of original file can not be recovered based on the new file name.
3. 256 bit Symmetric Key used for encryption is calculated based on a passkey/password

Authorize(Original FileName, PassKey)

1. This step produces 256 bit key used to encrypt the original file, based on a passkey.
2. It also returns the file’s name in cloud based on the original file name and passkey.

Recover(Encrypted FileName, PassKey)

1. Using 256 bit Key produced from combination of filename and passkey, Encrypted file is decrypted.
2. This step attempts to regenerate the original file.

Uploading and storage

- If the user wants to upload a file to cloud, which is an encrypted and integrity protected storage, it is encrypted and uploaded to the cloud by the client-side application.
- The encryption is performed with a fresh 256-bit symmetric key chosen by the clientside application.
- The encryption algorithm client side application applies is AES in CFB mode. The integrity of the files is protected with HMAC.
- Following this, a TLS tunnel is established between the client machine and the cloud. Optionally, Each upload can be authenticated with the digital signature of the uploader.

DEMO: PreProcess()

1. ./PreProcess PassKey OriginalFileName.txt

This will create a new File *in* ***binary format*** with random name.

By looking at the content of the file, you will only see some garbled text.

You can upload the new file to cloud and share the PassKey and Name of the original file (OriginalFileName.txt) with your friend (with whom you want to share the file)

DEMO: Authorize()

2. ./Authorize PassKey OriginalFileName.txt

Your friend should run this to find out the file in cloud that he needs to download.

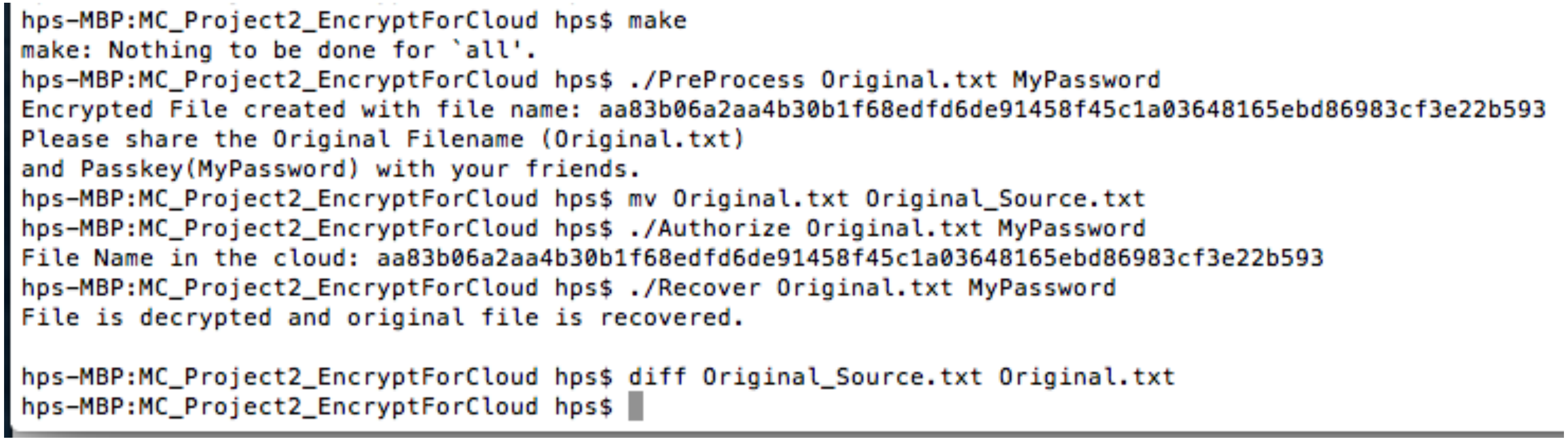
This will generate a 256 bit key to be used in next step and also, it will give you actual name of the file in cloud.

DEMO: Recover()

3. ./Recover OriginalFileName.txt EncryptedFileName 256Key

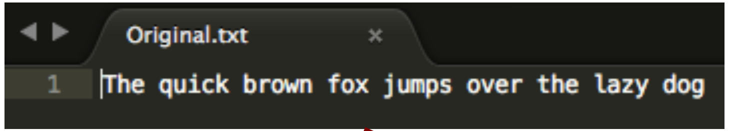
This step will recreate the original file from encrypted file based on 256 bit key generated in the previous step and original file name.

DEMO: Screenshot



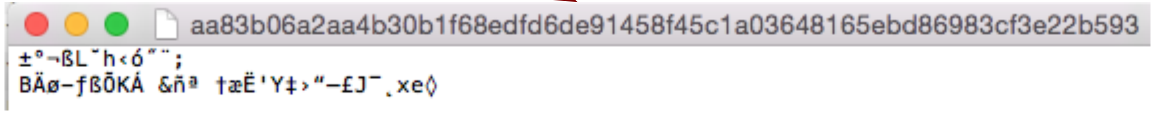
```
hps-MBP:MC_Project2_EncryptForCloud hps$ make
make: Nothing to be done for `all'.
hps-MBP:MC_Project2_EncryptForCloud hps$ ./PreProcess Original.txt MyPassword
Encrypted File created with file name: aa83b06a2aa4b30b1f68edfd6de91458f45c1a03648165ebd86983cf3e22b593
Please share the Original Filename (Original.txt)
and Passkey(MyPassword) with your friends.
hps-MBP:MC_Project2_EncryptForCloud hps$ mv Original.txt Original_Source.txt
hps-MBP:MC_Project2_EncryptForCloud hps$ ./Authorize Original.txt MyPassword
File Name in the cloud: aa83b06a2aa4b30b1f68edfd6de91458f45c1a03648165ebd86983cf3e22b593
hps-MBP:MC_Project2_EncryptForCloud hps$ ./Recover Original.txt MyPassword
File is decrypted and original file is recovered.

hps-MBP:MC_Project2_EncryptForCloud hps$ diff Original_Source.txt Original.txt
hps-MBP:MC_Project2_EncryptForCloud hps$ 
```

DEMO: Original vs Encrypted File



Original.txt

```
The quick brown fox jumps over the lazy dog
```



aa83b06a2aa4b30b1f68edfd6de91458f45c1a03648165ebd86983cf3e22b593

```
±º¬ßL˘h‹ó˝¨;
BÄø–ƒßÕKÁ &ñª †æË'Y‡›“–£J¯˛xe◊
```

If tried with wrong PassKey

- 256 bit AES Decryption key is calculated as 256 bit SHA256 hash of combination of Original file name and PassKey.
- If you don’t know both of them, you can’t crack the file encrypted with AES256.

Extra Credit Work 1 - Cloud Storage Servers

	Amazon AWS	Rackspace	Google App Engine	Microsoft Azure
Deployment Model	Public Cloud	Hybrid Cloud Private Cloud Public Cloud	Public Cloud	Public Cloud
Service Model	Infrastructure as a Service	Infrastructure as a Service	Platform as a Service	Infrastructure as a Service
Control Interface	Web Based Application/ Control Panel API Graphical User Interface	Web Based Application/ Control Panel API Command Line	Web Based Application/ Control Panel API	Web Based Application/ Control Panel API Command Line

Extra Credit Work 1 - Cloud Storage Servers

cont..

	Amazon AWS	Rackspace	Google App Engine	Microsoft Azure
Features	Auto Scaling Cloud Storage DNS Management Messaging Services VPN Access	Auto Scaling Cloud Storage Block Storage DNS Management System Monitoring	Horizontal Scaling	Auto Scaling Cloud Storage Block Storage Disaster Recovery Horizontal Scaling Messaging Services Vertical Scaling VPN Access

Extra Credit Work 2 - File Name Generator

- We have used SHA256 hash of the original file name as the new file name that should be sent to the cloud.
- With this convention, it is impossible for the cloud to find out original filename because of the way how hashing algorithms work.

Work Distribution

- Harshit Sanghvi (hs2619)
  - PreProcess(): Encrypting file with AES256.
  - Extra credit work 1: SHA256 hash of original file name
- Neel Shah (nhs272)
  - Authorize and Recover Function
  - Extra credit work 2: Cloud storage servers.

References

[1] Cloud Security Alliance, “Top Threat to Cloud Computing V1.0,” March 2010. https://cloudsecurityalliance.
org/topthreats/csathreats.v1.0.pdf

[2] “Microsoft accuses former employee of cloud data theft”, February 2011. http://www.zdnet.com/microsoftaccuses-former-employee-of-cloud-data-theft-4010021691/

[3] J. Mutch, “How to Steal Data from the Cloud - An Easy Guide for IT Admins”, Vol. 1, Issue 7, 2010, Cloudbook
Journal, http://www.cloudbook.net/resources/stories/how-to-steal-data-from-the-cloud

[4] I. Lám, Sz. Szebeni, L. Buttyán, “Invitation-oriented TGDH: Key Management for Dynamic Groups in an Asynchronous
Communication Model”, International Workshop on Security in Cloud Computing (CloudSec),
Pittsburgh, PA, September 2012

[5] I. Lám, Sz. Szebeni, L. Buttyán, “Tresorium: Cryptographic File System for Dynamic Groups over Untrusted
Cloud Storage”, International Workshop on Security in Cloud Computing (CloudSec), Pittsburgh, PA,
September 2012