A Cryptographically Authenticated and Secure File store

Sharath HP 19111082

Jaydeep Meda 19111039

1. Introduction

This document presents a paradigm of design considerations to be incorporated in implementing a simple, *cryptographically authenticated and secure file store*. The whole system is designed taking care of the fact that Client machine can be shut down inadvertently. Because of this, every bit of critical information (cannot be generated otherwise) is stored on the Data Store (DS) Server & in a secure manner, given that the Server is Untrusted. It is also to be noted that the Usernames corresponding to different users & their respective Public Keys (Public Key - Private Key Cryptography) are stored on a Trusted Key Server (believed to be secure).

This system ensures the following two properties during its use:

Confidentiality – Every piece of information stored on the DS is protected by a cryptographically secure Encryption scheme based on a Key Unique to every user (& file). It is important to note herein that a malicious user wouldn't be able to make a targeted attack on any particular user record/file block, as the associated MAC (see below) is itself stored as an independent record.

Integrity – Aside storing the above information, it is also protected by a MAC. Owing to the nature of hash functions, we don't encrypt this MAC, as it is not possible to derive the original information back from the MAC.

2. Design Considerations

2.1 File Storage Structure

While having a UNIX-like INODE implementation is one way to store the file blocks, we chose to differ from this convention & instead have come up with an intuitive way to store & access the file blocks. This scheme is also file size agnostic i.e., the file size that can be stored on the DS is limited only to the capacity of the DS itself. Working under the constraints & requirements of the system, this approach involves automatic generation of keys to the respective file blocks, which can be fetched on demand. The description to follow will further make it clear to the implementer. Of course all of these are stored in Encrypted form & protected by MACs as well. Access to such a file is possible only by an authorized user, who has been granted access to the file (who has knowledge of a shared secret).

2.2 Data Structures

The following data structure is used to store the User Data, the context of which is available to every call of any method invoked by the User.

```
User {
   Username: string
   Password: string
   PrivK: rsa.Privatekey
}
```

The following data structure is used to store information pertaining to a file. This is created on a per file basis whenever a User (owner) creates the file.

```
FileRecord {
   Owner: string
   FileSize: int
}
```

The following data structure is used to share file related information with collaborators

```
SharingRecord {
   Signature: [ ]byte
   EncMsg:  [ ]byte
}
```

The next section gives a comprehensive description of individual procedures needed to enable the system. These try to cover various aspects that are needed to be kept in mind during implementation while being flexible to choose the specific underlying details (such as choice of key generation, encryption algorithm, suitable handling of erroneous situation etc). Below is a list of symbols and definitions that will help better understand the description that follows:

E_K(something): Encrypt 'something' using 'K' as a key (to the encryption algorithm chosen)

D_K(string): Decrypt 'string' using 'K' as a key

MAC_K(string): Hashes 'string' using hmac and 'K' as a key

A Cryptographically Authenticated and Secure File store

Sharath HP 19111082

Jaydeep Meda 19111039

FileKey: Symmetric key used for encryption and decryption of file data blocks. It is unique for every file. (Non - deterministic key)

UK: Unique Key created for a user based on his/her username and password.

GetDS/KS (key):
Fetches value from DS/KS corresponding to **key** if it exists else Raise Error

SetDS/KS (key, value):
Stores the 'key': 'value' on the DS/KS

Fetch_MAC_and_Compare:
Mac <- GetDS (E_K{$Entity}Mac)
if [Mac not equal to $MAC_{\$FileKey}$({$Value})] then Raise Error
$Entity – DS Key used in previous step (before encryption)
$Value – DS Value returned in previous step (Encrypted)
$FileKey – Encryption Key used in previous step

GenUK (UserName, Password): Generates a Unique Key using the input parameters.

GenFileKey (FileName): Generates a random key based on the FileName in the context of the invoker (user)

GetFileKey (FileName, UK):
EncFileKey <- GetDS (E_{UK}(FileNameKey))
if [Fetch_MAC_and_Compare fails] then Raise Error
return D_{UK}(EncFileKey)

GetFR (FileKey):
EncFR <- GetDS ($E_{FileKey}$(FRAddr))
if [Fetch_MAC_and_Compare fails] then Raise Error
return $D_{FileKey}$(FR)

Raise Error - Exit procedure returning an error

3. Procedures

Initialize User

Result: Populates the User Data Structure (UDS) & stores it on Data Server (DS) securely. Also stores User Public Key (PK) on Key Store (KS).

Input: UserName, Password

UK <- GenUK (UserName, Password)
PrivK, PubK <- KeyGenAlgorithm ()
> Populate UDS
EncUDS <- E_{UK}(UDS)
EncUserName <- E_{UK}(UserName)
> Store EncUDS on DS
> Compute and Store MAC of EncUDS
Store User's Public Key on KS with key as UserName

Fetch User

Result: Fetches UDS of a Valid User

Input: UserName, Password

UK <- GenUK (UserName, Password)
if UserName doesn't exist on KS then Raise Error
> Fetch UDS from KS
if Passwords match then
    return UDS
else
    UDS = null & Raise Error

Store File

Result: Stores a newly created File on DS

Input: FileName, Data, UDS (context)

if Data not a multiple of BlockSize then Raise Error
UK <- GenUK (UDS.UserName, UDS.Password)
if FileName doesn't exist
    FileKey <- GenFileKey (FileName)
    EncFileKey <- E_{UK}(FileKey)
    Store EncFileKey on DS
    Compute and Store MAC
else
    Fetch existing FileKey from DS
> Populate File Record (FR)
EncFR <- $E_{FileKey}$(FR)
> Store FR on DS with MAC

A Cryptographically Authenticated and Secure File store

Sharath HP 19111082

Jaydeep Meda 19111039

Append File

Result: Append supplied data to the end of a file, if it exists.

Input: FileName, Data, UDS (context)

If Data isn't a multiple of BlockSize then Raise Error
UK <- GenUK (UDS.UserName, UDS.Password)
FileKey <- GetFileKey (FileName)
FR <- GetFR (FileKey)
For each block of Data (of size BlockSize) do
    EncData <- $E_{FileKey}$(DataBlock)
    Store EncData on DS along with MAC
EncFR <- $E_{FileKey}$(FR)
> Store EncFR on DS with corresponding key.
> Compute and Store MAC of EncFR on DS.

Load File

Result: Loads the specified data block of the given file

Input: FileName, DataOffset, UDS (context)

if DataOffset > FS then Raise Error
UK <- GenUK (UDS.UserName, UDS.Password)
FileKey <- GetFileKey (FileName, UK)
EncData <- GetDS ($E_{FileKey}$(FileKey{$DataOffset}))
if [Fetch_MAC_and_Compare fails] then Raise Error
return $D_{FileKey}$(EncData)

Share File

Result: Returns a secret message to share a given file with the intended Recipient

Input: FileName, RecipientName, UDS (context)

UK <- GenUK (UDS.UserName, UDS.Password)
FileKey <- GetFileKey (FileName, UK)
RecvPubK <- GetKS (ReceipientName)
EncMsg <- $E_{RecvPubK}$ (FileKey)
Signature <- $Sign_{UDS.PrivK}$(EncMsg)
> Populate Sharing Record with Signature & EncMsg
> Compute MAC of Sharing Record
return (Sharing Record + MAC)

Receive File

Result: Receive a secret message to access a given file

Input: FileName, SenderName, SecretMsg, UDS (context)

If [Verify MAC/Signature of SecretMsg fails] then Raise Error
EncMsg <- StripMAC(SecretMsg)
SenPubK <- GetKS (SenderName)
UK <- GenUK (UDS.UserName, UDS.Password)
FileKey <- $D_{UDS.PrivK}$($D_{SenPubK}$(SecretMsg))
EncFileKey <- E_{UK}(FileKey)
> Store EncFileKey on DS with corresponding key
> Compute and Store MAC of EncFileKey

StripMAC – SecretMsg has a fixed format. Initial few Bytes (mutually agreed upon) correspond to MAC, which is removed.

Revoke File

Result: Changes File encryption & stores it in a different location

Input: FileName, UDS (context)

UK <- GenUK (UDS.UserName, UDS.Password)
FileKey <- GetFileKey (FileName, UK)
FR <- GetFR(FileKey)
if UDS.UserName not Owner then Raise Error
Do
    FileKeyNew <- GenFileKey (UK)
Until [FileKey not equal FileKeyNew]
> Decrypt File Data Blocks with FileKey, Encrypt with FileKeyNew (verifying MAC, refer Store File) and store on DS.
EncFileKeyNew <- E_{UK}(FileKeyNew)
> Store EncFileKeyNew on DS
> Compute and Store MAC of EncFileKeyNew