

Design

Part 1

In order to store a file on the server, a client first needs three symmetric keys: k to encrypt the filename, k_e to encrypt the file content and k_a to authenticate the file. In my design, I use k for all files and I use one set of (k_e, k_a) , i.e. for file i I use the key set (k, k_{e_i}, k_{a_i})

The key k is generated specifically for each client when the client is initialized, that is in constructor method I call *make_unique_symmetric_key()*. This key is then encrypted asymmetrically using the client's public key and stored along with its signed encryption using the client private key on the server *client/info*, i.e. we store *client/info* : $(E_{K_{pub}}(k), Sign_{K_{pri}^{-1}}(E_{K_{pub}}(k)))$

Whenever we want to upload a file, we generate a key set (k_e, k_a) . Similarly, this key set is encrypted, signed as $(E_{K_{pub}}((k_e, k_a)), Sign_{K_{pri}^{-1}}(E_{K_{pub}}((k_e, k_a))))$. To store this encryption on the server, we have to make a path specific for the encrypted filename. That is we first have to encrypt the filename into $E_k(filename)$ using the symmetric key k , then the path and the (k_e, k_a) encryption on the server is *client/ $E_k(filename)$ /keys* : $(E_{K_{pub}}((k_e, k_a)), Sign_{K_{pri}^{-1}}(E_{K_{pub}}((k_e, k_a))))$

After having the key set (k, k_{e_i}, k_{a_i}) for file i , we encrypt the file content using k_e and k_a , AES-HMAC with CBC mode and store the encryption on the server *client/ $E_k(filename)_i$* : *AES_{CBC} - HMAC _{k_{e_i}, k_{a_i}} (filename _{i} , file-content _{i})*

Whenever we download some file or get keys from the server for using, we have to verify the signature (for keys) or check the integrity (for file content) using the corresponding methods 'digital signature' or 'HMAC' before decrypting it.

Part 2

If client 1 want to share a file i with another client, say *client_2*, client 1 follows 4 steps:

1. Share the key set (k_{e_i}, k_{a_i}) with the client 2 by making the encrypted and signed key set available under the client 2 name, that is client 1 create and put onto the server:

$$client_2/E_k(original\ filename)/keys : \left(E_{K_{pub_client2}}((k_e, k_a)), Sign_{K_{pri_client1}^{-1}}(E_{K_{pub_client2}}((k_e, k_a))) \right)$$

2. Send the client 2 the key set (k_{e_i}, k_{a_i}) and the encrypted original filename $E_k(original\ filename)$ or *eof* via an encrypted and signed message *msg* containing

$$\left(E_{K_{pub_client2}}((k_e, k_a, eof)), Sign_{K_{pri_client1}^{-1}}(E_{K_{pub_client2}}((k_e, k_a, eof))) \right)$$

3. Make the link to the file content *client_2/eof* : "[*POINTER*]client_1/eof"
4. Create and MAC a list of clients the file is share with
client_1/ $E_k(original\ filename)$ /shared_with : [*client 2*]

When client 2 receives the share, he/she check the integrity and authentication of the message, then creating two links:

$client_2/E_{k_{client_2}}(new\ filename)/keys : "[POINTER]\ client_2/E_{k_{client_1}}(original\ filename)/keys"$
 $client_2/E_{k_{client_2}}(new\ filename) : "[POINTER]\ client_2/E_{k_{client_1}}(original\ filename)"$

Client 2 also has to check the integrity and authentication and re-sign the keys using his/her private key before using the keys.

When client 1 revokes the client 2's access, he/she deletes the links created at step 1 and 3 above, pop client 2 out of the share list, generate a new set of key (k'_{e_i}, k'_{a_i}) for file i , re-encrypt and HMAC the file content and recursively go through the share lists to re-distribute the new key set to the other client. The re-distribution is done by implementing step 1 repeatedly. Since the original client has to sign the key set using his/her private key, the other clients have to check and, if necessary, re-sign this key set whenever they want to use it.

The original client can check the integrity and access the share lists recursively because all the share list are HMACed using the share key k_{e_i} . Thus, whenever the clients are still in a sharing relation wrt a file, the sharer can access the check the integrity and access sharee's share list.

The key points of this design are:

1. Whenever we put any thing on the server, we have to encrypt and sign or HMAC it, and whenever we download something from the server, we have to verify the signature or check its integrity before decrypting it. We may not encrypt the share list because the specs says the share list can be known by third party but we still have to HMAC it to make sure it is not modified.
2. The original client has access to all the shared key sets to modify then in the case of revoke. When he/she re-distributes the new key set, he/she signs it under his/her private key, so the other user must check and re-sign the new key set before using it.
3. The revoke is done by 5 steps: 1. Delete the file link; 2. Delete the key link; 3. Remove the sharee from the share list; 4. Re-encrypt the file with the new key set, and 5. Re-distribute the new key set

Security Analysis

Attack 1:

A malicious server can modify anything on the server to break the integrity of keys, file contents, share lists. This is impossible because of the key point 1 above: whenever we put anything on the server, we encrypt and sign or HMAC it.

Attack 2:

An attacker can eavedrop the message. However, he cannot decrypt it nor modify it since the message is assymmetrically encrypted by public key and signed by private keys.

Attack 3:

An attacker may want to swap files but it is impossible since we encrypt filename with file content.

Attack 4:

A malicious server or attacker may want to steal any keys or filename/file content. It is impossible because the all the information are decrypted.