Project 2 Part 2

Done by:
Name: Chin Zhi Xian SID: 3033484336
Name: Nathanael S Raj SID: 3033484331

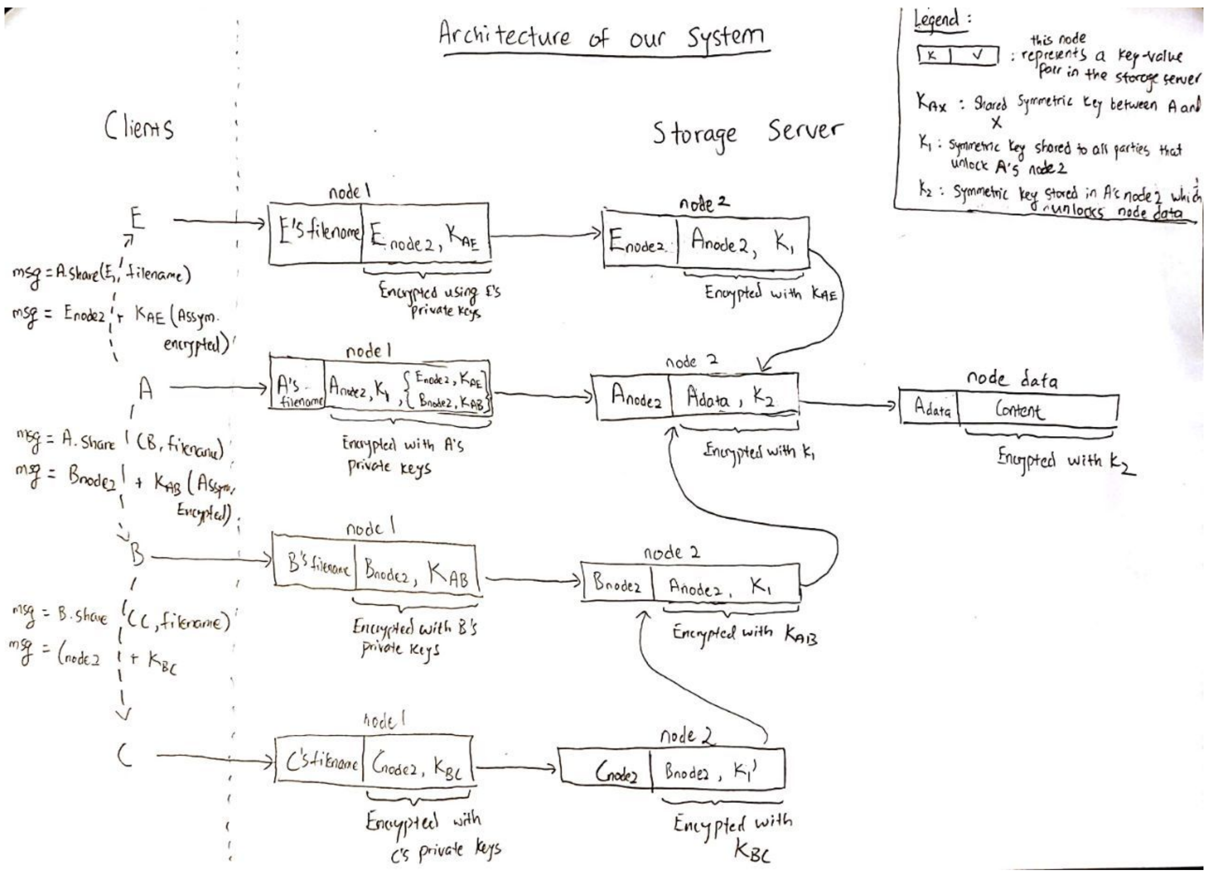
Contents:

1. Introduction

In this design document, we will first go through a high level explanation of the architecture of our system. This is followed by a detailed explanation of the steps involved in the implementation of each function. Finally we explore the security analysis of our system and look at how our system is invulnerable to various attacks.

2. Design

2a. Overview of System Architecture



Architecture of our System
Clients
Storage Server
Legend :
K | V : this node represents a key-value pair in the storage server
KAx : Shared Symmetric key between A and X
K1 : Symmetric key shared to all parties that unlock A's node 2
K2 : Symmetric key stored in A's node 2 which unlocks node data
E
A
B
C
msg = A.Share(E, filename)
msg = Enode2 + KAE (Assym. encrypted)
msg = A.Share (B, filename)
msg = Bnode2 + KAB (Assym. Encrypted)
msg = B.Share (C, filename)
msg = Cnode2 + KBC
node 1
E's filename | Enode2, KAE
Encrypted using E's private keys
node 2
Enode2 | Anode2, K1
Encrypted with KAE
A's filename | Anode2, K1, {Enode2, KAE ; Bnode2, KAB}
Encrypted with A's private keys
Anode2 | Adata, K2
Encrypted with K1
node data
Adata | Content
Encrypted with K2
B's filename | Bnode2, KAB
Encrypted with B's private keys
Bnode2 | Anode2, K1
Encrypted with KAB
C's filename | Cnode2, KBC
Encrypted with C's private keys
Cnode2 | Bnode2, K1'
Encrypted with KBC

Figure A: Architecture of our system

The architecture of our version of Dropbox is designed around each client having access to different 'pathways'. Each pathway is made up of nodes where each node stores the key to unlock the next node in the pathway. If the pathway resolution is successful, the client will be able to access the final data node and is granted access to the data.

In fig A, we look at the architecture of the system which represents client A sharing a file to B and E and client B sharing that same file to C. When A uploads a file, he creates three nodes, node 1, node 2 and node data as shown in fig A. To download the file, he first unlocks node 1, which yields the pointer and key to node 2. Node 2 can then be unlocked, which in turn provides him with the pointer and key to node data. Finally, node data is unlocked, allowing the file to be downloaded.

If A wanted to share a file to B, he first generates K_{AB}, which is a shared key exclusive for A's and B's use only. He creates B's node 2 and encrypts B's node 2 using K_{AB}. B's node 2 will point to A's node 2 and will have the key to unlock A's node 2. Once B has access to A's node 2, B can 'enter A's pathway' and is granted access to the file.

The purpose of including a second tier of nodes is to allow for revocation. During revocation, we need to be able to change both the encryption key and location of data, so that the revoked client will be able to neither locate nor decrypt the file. All other clients whom the file has been shared with will need to be provided this updated pointer and key. This intermediate tier of nodes which both sharer and sharee can access allows the sharer to update the sharee of the changes. Therefore, the procedure of A revoking B will be as follows. A regenerates new node 2 and new node data which will contain new pointers and new keys. A updates E's node 2 with the new pointer and new key to his new node 2. Without the new pointer and new keys, B has no way of finding the file anymore.

In this high-level outline of how the system works, we have deliberately left out implementation details to prevent overcomplicating the explanation. For example, we only mentioned existence of a single key in each node (except node data) when in reality, there are actually 2 keys in each node, an encryption and MAC key. Details like these will be covered in the subsequent part of the report, where we fully describe how the upload, download, share, receive_share and revoke functions have been implemented.

2b. New Upload Function

When a client first uploads a file, he creates three nodes - two key directory nodes and one data node. He will create two new sets of symmetric encryption keys K_{e1} and $K_{e2,}$ and two new sets of MAC keys K_{m1} and K_{m2} :

1. First node (key directory node) - Contains two layers of data
    a. First layer
        i. Second layer data (Encrypted using client's own symmetric key K_a and MAC key K_a)
        ii. Counter value to decrypt the data
        iii. MAC of the plain text data
    b. Second layer
        i. Pointer to the next node
        ii. K_{e1} and K_{m1} required to decrypt the next node and verify the data
        iii. List of all of the users which client will be sharing to, including all the relevant data (see Share function for more details)
2. Second node (key directory node) - This will be the node which other clients which we have shared the data with have access to
    a. First layer
        i. Second layer data (Encrypted using K_{e1})
        ii. Counter value to decrypt the data
        iii. MAC of decrypted data
    b. Second layer
        i. Pointer of the next node
        ii. K_{e2} and K_{m2} required to decrypt the next node and verify the data

3. Third node (data node) - This will be the final node which contains the file
    a. First layer
        i. (Encrypted using K_{e2}) File
        ii. Counter value to decrypt the data
        iii. MAC of file

The address of the first node will be a H(client_name||file_name). The client name is concatenated with filename to ensure that no collisions occur when different clients upload the same file. The name is hashed to fulfil confidentiality of the file and user names.

2c. Share Function

Used when a client ("Alice") shares with another client ("Bob"). Alice will first create a new node which points to Alice's second key directory node. Alice will also create a new set of symmetric encryption key K_{eAB} and K_{mAB} which Bob can use to decrypt the data of the new node. The pointer to this new node will be P_b. Alice will create a new node whenever she chooses to share the data with a new user.

1. Bob's second node
    b. First layer
        i. (Encrypted using symmetric key K_{eAB} and MAC key K_{eAB}) Second layer data
        ii. Counter value to decrypt the data
        iii. MAC of the decrypted data
    c. Second layer
        i. Pointer of the next node (Alice's second node)
        ii. K_{e1} and K_{m1} required to decrypt the next node (Alice's second node) and verify the data

Alice will then add on the data to the User List within the second layer of her first node. She will keep track of each user's name, the pointer of the node she has created for them, and both the symmetric encryption and MAC keys she used to encrypt the data within the second node.
Alice then sends msg M = P_b || K_{eAB} || K_{mAB} to Bob by asymmetrically encrypting M and sending it to him through an out-of-band channel. This is performed by accessing the public key server to retrieve Bob's public key, which is then used to asymmetrically encrypt M. She also performs RSA signing of msg using her own RSA key before sending the resultant product to Bob.

$$D = (E_{Kb}(P_b \,||\, k_{e1} \,||\, k_{m1}))$$

$$C = Sign_{KRa}(P_b \,||\, k_{e1} \,||\, k_{m1})$$

$$M = D \,||\, C$$

2d. Receive Share

Used when a client ("Bob") chooses to receive share from another client ("Alice"). Bob will receive M from Alice through an out-of-band channel. Bob will first verify the integrity and authenticity of the message using $Verify_{KRa}(D, C)$ by retrieving Alice's RSA key from the public key server. Once he has established that the message has came from Alice untampered, he will decrypt D using his own private key of K_b(D) to access the message contents. Bob will create his own node using his own symmetric key K_b and MAC key K_b to maintain confidentiality and integrity respectively:

1. Bob's First node (key directory node)
   d. First layer
      i. (Encrypted using client's own symmetric key K_b and MAC key K_b) Second layer data
      ii. Counter value to decrypt the data
      iii. MAC of the decrypted data
   e. Second layer
      i. Pointer of the next node
      ii. K_{eAB} and K_{mAB} required to decrypt the next node (Bob's second node which Alice created) and verify the data

After the node is created, Bob now has all the decryption keys and pointers required to access and modify the file located within the final data node.

Bob's Pathway: Bob's First Node → Bob's Second Node (created by Alice) → Alice's Second Node → Alice's Third Node [DATA]

User sharing a file that has been shared to him

Whenever a user ("Bob") chooses to share the file with another user ("Dave"), he will create a second node (key directory node) for the other user which points to his own second node. Dave will then receive the message which Bob has sent to him, create his own first node and will now have a pathway to access the file.

Dave Pathway: Dave's First Node → Dave's second Node (created by Bob) → Bob's Second Node (created by Alice) → Alice's Second Node → Alice's Third Node [DATA]

2e. Download Function

The client will use the user's own symmetric key K_{eX} and MAC key K_{mX} to access the first node. This function will then resolve all of the key directory nodes to reach the final data node created by the owner. Each key directory node only contains sufficient data to access the next node (Location of next node, symmetric encryption key and MAC key of the next node).

2f. Revoke

Used when the owner of a file ("Alice") wishes to remove the read/write permissions of another user ("Bob") while maintaining read/write permissions for other users she has shared to ("Carol"). Revoke will take the following steps:

1. Saves the updated user list (less the revoked user)
2. Run upload function on the current data file again to create new nodes

    a. The client will take note the pointer for Alice’s second node when Upload is called
    b. The client will save the updated user list into Alice’s new first node
3. Edit the data within the second nodes of all non-revoked users which Alice has shared to. The pointer in all of these second nodes initially pointed to Alice’s original second node but it now has to change it to point to Alice’s new second node instead.

Carol’s pathway: Carol’s First Node (Original) → Carol’s Second Node (Original but with altered pointer) → Alice’s Second Node (New) → Alice’s Third Node [DATA] (New)

Bob’s pathway:
Bob’s First Node (Original) → Bob’s Second Node (Original) → Alice’s Second Node (Original) → Alice’s Third Node (Original)

Bob no longer has access to the new data and further modifications made to the file. Users which Bob has shared to will no longer have access to the new data as well as they will need to go through Bob’s Second Node to reach Alice’s Data node. Take the case of another user (“Dave”) which Bob shared the file to before Bob’s rights were revoked.

Dave’s pathway:
Dave’s First Node (Original) → Dave’s Second Node (Original) → Bob’s Second Node (Original) → Alice’s Second Node (Original) → Alice’s Third Node (Original)

3. Security Analysis

3a. Choice of Encryptions

3ai. Key Directory Nodes

In our key directory nodes, we use symmetric key system. Symmetric keys are used as they are generally easier to generate and there is a secure way of key distribution. We distribute the symmetric keys by (1) placing them in the nodes and (2) sending it in a ‘share’ message. In (1), each node is securely encrypted with the symmetric key of the previous node and will not be able to be accessed directly. In (2), the ‘share’ message is encrypted using an asymmetric key system and the data contained within cannot be accessed by eavesdroppers. Using a symmetric key system will help us fulfil the following criteria:

i) Confidentiality

We use AES-CTR$_k$(n) for the value within each node. This provides confidentiality for the data within each node as AES permutates the data randomly each time after dividing it into blocks. The symmetric keys and the nonce for the IV are chosen randomly with get_random_bytes() during client initialisation and every time a new node is created.

ii) Integrity

We use SHA256-HMAC$_k$(n) for our MAC to achieve integrity verification for our input while maintaining confidentiality. The MAC is appended with the encrypted data of each node so that the user will be able to check that the data has not been tampered with. HMAC is able to provide a pseudorandom MAC for the data such that it will not leak any information as well.

3aii. Message

We use asymmetric key encryption in our message. This will help us solve three different problems:

i) Confidentiality

We use El-Gamal encryption as it is a trapdoor function. By implementing ElG$_k$(data) for our message, we are essentially implementing a one-way trapdoor function using the difficulty of the Discrete Log Problem. Only the user who possesses the private key to ElG$_k$ will be able to access the data within the message given non-exponential time.

ii) Integrity and Authentication

We use PCKS #1 standardized RSA key for our integrity and authentication check. The sharer will sign the message off before sending it off to the person whom he has shared with. The RSA signing process is also a one-way trapdoor function as well - only the one who possesses the RSA private key can create a RSA signature to be verified in non-exponential time.

ii) Scalability - Key distribution Problem

Assuming we have n users in our file sharing system, if we use a symmetric encryption system each user will need a new key each time he is the target of a share message and this will take O(n^2) keys.

If we instead use an asymmetric encryption system, each user will only need a single public and private key. This will take O(n) space. It is obvious that even though keys are harder to generate in an asymmetric encryption system, the asymmetric encryption system is much more scalable.

3b. Potential Attack Vectors

Attack 1. Man in the middle attack
This attack can occur when Alice desires to pass a 'share' message to Bob. Mallory intercepts the message and replaces it with his own hoping that Bob will access Mallory's file instead of Alice's file. We protect against this attack by having Bob verify that the signature of the message received is created by whomever has shared to the file with him. Mallory will be unable to create Alice's RSA signature in non-exponential time.

Attack 2. Eavesdropping attack
This attack can occur when Alice desires to pass a 'share' message to Bob. Eve picks up on the message and tries to access Bob's second node. We protect this attack by encrypting the message using an El-Gamal encryption system. Since Eve does not have access to Bob's private key she will not be able to decrypt the message in non-exponential time.

Attack 3. Malicious server swapping data between a user with access and a revoked user (without access)
The attack can occur if a malicious server controlled by Mallory (and used by Alice) switches the values stored at two names. Alice has initially uploaded (a, b) and has revoked Mallory and updated the file such that it is now (a, b'). Mallory switches all the values located within the server and this will take O(n^2) time, where n is the number of values stored within the server. We protect against this attack in two ways: (1) Alice will not be able to download the wrong file as the MAC of the file has to be verified before the download function returns the file. (2) Mallory will not be able to access b' as it is encrypted with a new symmetric key which Mallory has no access to.