Key Agreement in Peer-to-Peer Collaboration Systems

Charles Parzy--Turlat

School of Computer and Communication Sciences

Decentralized and Distributed Systems lab

Semester Project

February 2019

Responsible
Prof. Bryan Ford
EPFL / DEDIS

Supervisor
Kirill Nikitin
EPFL / DEDIS

Contents

1 Introduction

Nowadays, online platforms providing editing of files shared with other users are widespread and used by lots of people. The most popular example is Google Doc. It allows the people to create and modify online documents while collaborating with other users in real time. It simplifies the lives of its users by dealing with all the concurrency issues introduced by having several users contributing to the same file at the same time.
However, despite its usefulness, it relies on a central server to operate properly. Thus it is a **centralized system**. These kind of systems have some flaws.

First of all, all the users data are stored in a central server, which the users have to trust with their data. Thus, there is a privacy issue. When the users used centralized systems, they share their private and sensitive data with the third party operating the system. For example, Google owns every documents created by its users.

Then, the second issue with centralized systems is that they are a single point of failure. First, they are fault intolerant. Indeed, they are more likely to fail accidentally as they rely on a single component. Moreover, they are not really resistant to attacks. Indeed, there is only one point of attack that would disarm the system. This makes the system resistance depends on the resistant of this one point. Thus, centralized systems can be considered fragile.

Unlike centralized systems, decentralized systems can offer more guarantees regarding privacy. Indeed, the users own their data and do not entrust them to any third party. In addition, decentralized systems rely on many separate components making them more fault tolerant and attack resistant.

This project focuses on the Peerdoc platform which is an alternative to centralized collaboration tools like Google Docs. The Peerdoc platform is a decentralized version of these tools. It allows the users to create and collaborate together on documents, while being a decentralized system.

2 Goals

The goal of this project is to develop a key agreement system for the Peerdoc platform. The aim of such system is to provide mechanisms for different users collaborating on the same file to devise a shared key. This shared key is then used to encrypt the packets exchanged by the users regarding the state modification of the document.

The trivial way to implement such system would be to use asymmetric encryption. When a peer is added into the network of collaborators, it

broadcasts its public key. Then, whenever a peer in the network wants to send a packet to the other peers, it sends to each of these peers the packet encrypted with the destination peers public key. This way, when one of the destination peer receives the packet, it can decrypt it using its own private key. This way of encrypting the packets would certainly work. However, it will not scale with the number of peers in the network. Indeed, the more peer in the network, the slowest it will take to send a packet, due to the sending peer encrypting the packet with each destination peers public key before sending it. The number of encryption operation to broadcast one packet is linear with the number of peer in the network.

A more efficient and scalable solution is to use Diffie-Hellman tree in the key agreement system. The idea behind it is that each peer maintain a tree. The leaves of the tree are the peers collaborating to the document. The root of the tree contains the group key which each peer agrees on. Thus, all peers have the same group key. This way, whenever a peer has to broadcast a message, it just encrypts it with the group key. With this method, only one encryption is necessary.

Figure 1 represents a Diffie-Hellman tree maintained at peer A. The peers collaborating, A, B and C, are the leaves of the tree. Their nodes contain their public keys. The node for A contains also its private key as this tree is stored and maintained by A. The intermediate node between A and B, IN_{AB} contains the shared key between A and B. Both A and B agreed on the same shared key. The intermediate node between C and IN_{AB}, IN_{ABC} is the root. It contains the shared key between node A, B and C, which is also the group key.

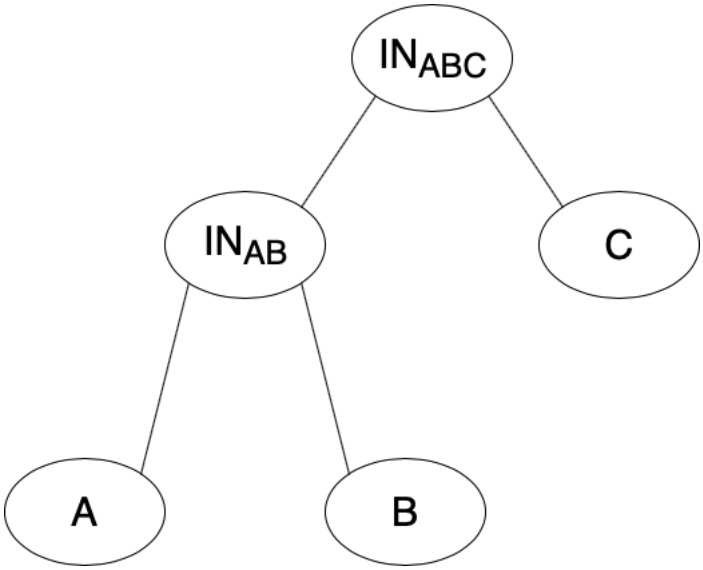

Figure 1: Example of a Diffie-Hellman tree between three nodes, A, B and C

3 Background

This section will present Peerdoc in more details, with the modules that have already been implemented before starting the implementation of the key agreement module.

Peerdoc is a decentralized system offering to the users to create documents and collaborate on them together. The frontend is a web interface written in HTML/CSS along with JavaScript, while the backend is written in Go.

The backend is splitted into several components:

- The ABTU module
- The Access Control module
- The Management module

The ABTU module handles the concurrency issues introduced by multiple users collaborating to the same file at the same time. The Access Control module handles the right of the users for each document. The Management module is the entry point of the system and is binding all modules together.

Figure 2 gives an idea on how the different modules are organized (with the key agreement module).

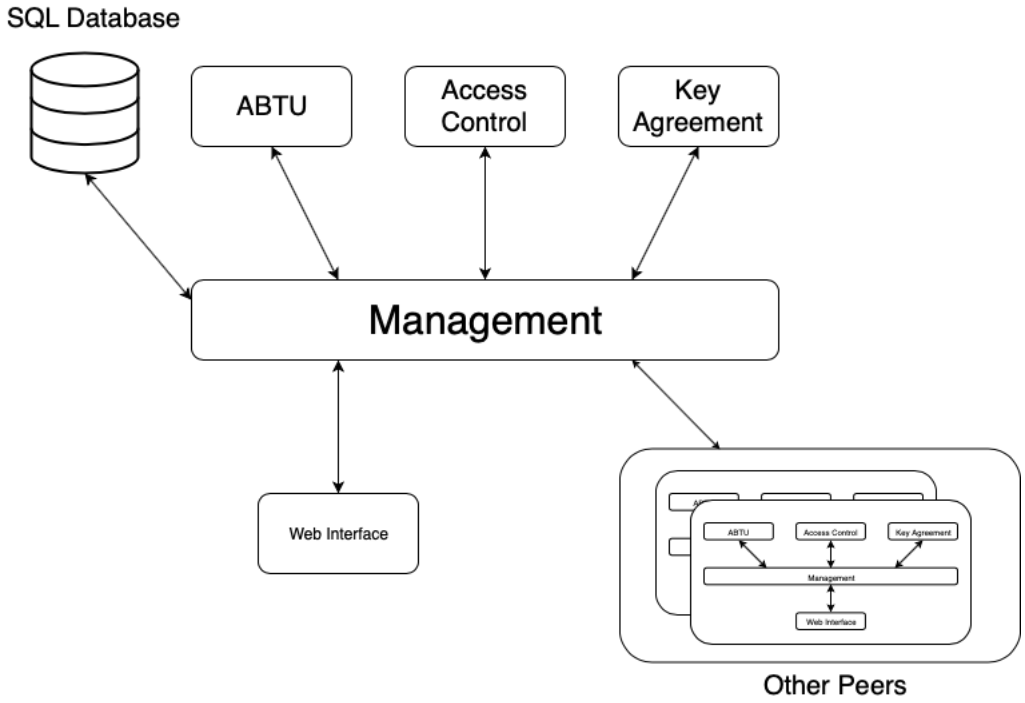

Figure 2: Organization of Peerdoc modules

4 Documentation Phase

During the documentation phase, 3 papers, each of them presenting a different implementation, have been studied.

- *Communication-Efficient Group Key Agreement*, Yongdae Kim, Adrian Perrig and Gene Tsudik [1]
- *Simple and Fault-Tolerant Key Agreement for Dynamic Collaborative Groups*, Yongdae Kim, Adrian Perrig and Gene Tsudik [2]
- *On Ends-to-Ends Encryption*, Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican and Kevin Milner [3]

These three papers will be slightly presented in the next subsections.

4.1 Paper 1: *Communication-Efficient Group Key Agreement*, Yongdae Kim, Adrian Perrig and Gene Tsudik [1]

This paper proposes a revisited version of the STR (Steer et al.) group key agreement protocol [4]. The authors extend STR to handle dynamic groups. It consists in building a linear Diffie-Hellman tree at each peer.

First of all, this protocol does not create a hierarchy between the peers participating in the key exchange. Whenever a new peer is added, a new root is created whose left child is the former tree and right child is the new node. This makes the join operation simple and its complexity constant with the number of peers in the network. Whenever a node leaves or is removed from the network, potentially all the keys up to the root have to be recomputed which makes the complexity of this operation linear with the number of peers in the network.

4.2 Paper 2: *Simple and Fault-Tolerant Key Agreement for Dynamic Collaborative Groups*, Yongdae Kim, Adrian Perrig and Gene Tsudik [2]

This second paper was written by the same authors as the first one. It is a slightly modified version of the first paper [1]. Indeed, instead of building a linear tree, each peer builds a balanced tree. Because the tree has to be balanced, more computations are required when a peer is added into the network. The complexity of this operation is logarithmic with the number of peers in the network. However, the leave operation is more efficient in this paper than in the first one. As the tree is balanced, instead of being linear, the complexity of the leave operation is logarithmic as well with the number of peers in the network. These changes in the complexities are the main differences between these two papers.

4.3 Paper 3: *On Ends-to-Ends Encryption*, Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican and Kevin Milner [3]

This paper also proposes a tree-based group key exchange protocol, but focuses on **post-compromise security** property.
A protocol between Alice and Bob provides **post-compromise security** if Alice has security guarantee about communication with Bob, even if Bobs secrets have already been compromised.
In a protocol that does not provide **post-compromise security**, it is possible for an adversary who compromised one participant to be able to intercept group messages indefinitely.
This protocol can be seen in two phases. First, the initiator of the group creates the tree and computes the keys allowing the other members to determine the group key. Then, each member can update its key.
Thus, this paper can be considered as an extension of the previous ones because it describes mechanisms that can be added to the Diffie-Hellman tree constructed as presented in the first two papers.

5 Design and Implementation

5.1 Implementation Choice

A quick and small survey was conducted in order to determine how people use document collaboration tools like Google Docs, especially whether they add more often contributors than they remove. 21 people participated to this survey.
First, Figure 3 shows that 61.9% of the participants have already removed some contributors. Moreover, during the study, only 1 participant out of 21 never added a contributor to a Google Docs. It seems that adding contributors is more common than removing them. It is confirmed by Figure 4, which shows the ratio addition/deletion. Almost half of the participant add 5 times more contributors than they remove. In average, the participants add 8.7 times more contributors than they remove.
According to this survey, addition of contributors seems more widespread than deletion of them. Thus, the addition operation has to be optimized. The deletion operation could be less efficient as it is less used by the people. Thus, the first paper [1] has been selected to the detriment of the second one [2] because its addition operation is done in constant time with the number of peers in the network.
The third paper [3], being an extension to the first two papers, has not been selected but is considered as a possible improvement that could be added to the key agreement module in the future.

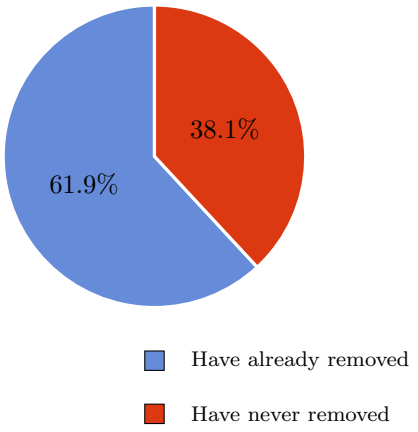

Figure 3: Proportion of people who have already removed contributors from a document.

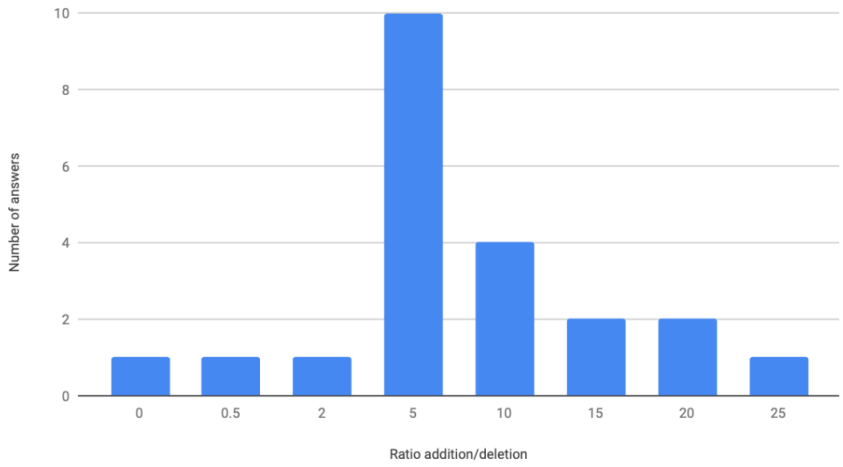

Figure 4: Number of answers according to the ratio addition/deletion

5.2 Code Integration

A key agreement module has been created. It contains all the code necessary to implement the key agreement procedure described in the first paper [1]. In this module, a structure representing an instance of the key agreement system was implemented. This instance holds and maintains the Diffie-Hellman tree. Each document has a key agreement instance attached to it meaning that each document has its own Diffie-Hellman tree, which is convenient because a peer does not always collaborate with the same peers on all its documents. Whenever a peer is added or remove from a document, the key agreement instance updates the Diffie-Hellman tree and recomputes the shared group key.

5.3 Implementation

5.3.1 Assumption

First of all, only the document owner has the right to add or remove people. Thus the document owner is the administrator and is equivalent to the sponsor in paper 1 [1].
This introduces a simplification of the protocol described in the paper 1 [1]. Indeed, the sponsor is always the same node in our case: the administrator. Thus, it does not have to be selected before processing any operation of the Diffie-Hellman tree.
The administrator being the peer that created the document, it is also its first contributor. Thus, the administrator is the leftmost leaf of the tree.
Except for this assumption, paper 1 [1] was faithfully implemented.

5.3.2 Group Key Computation

In the project, **elliptic-curve Diffie-Hellman** is used so that the peers agree on a group key.
How the group key is agreed between peers will be described in this section.

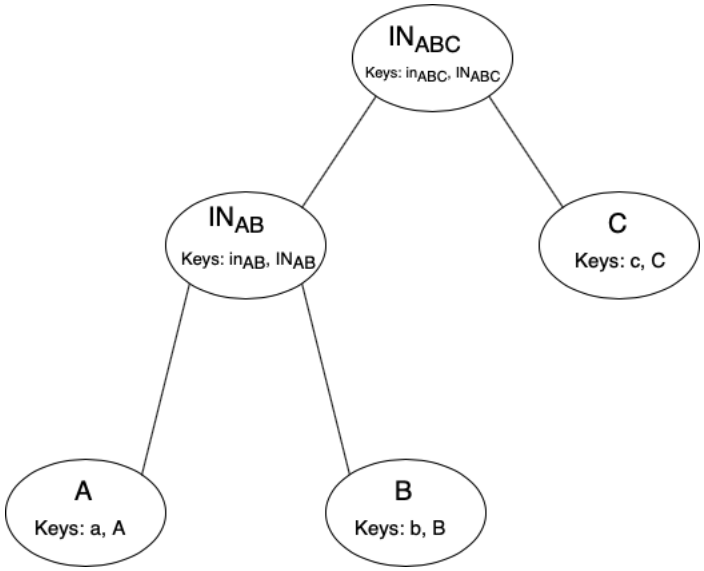

Figure 5: Diffie-Hellman tree with the keys at each node

The tree represented in Figure 5 above shows which keys are known by which peers. It is not a tree maintained at any peer. Indeed, a given peer does not know the private keys of other peers.
Each peer knows its own private and public keys. For example, peer A knows a and A which are respectively its private and public keys.
In addition to its own keys, each peer knows the keys contained in the intermediate nodes above itself. For example, peer A knows the private and public keys of nodes IN_{AB} and IN_{ABC}.
However, a peer only knows the public keys of the other peers and of the intermediate nodes below it. For example, peer C knows the public keys of A, B and IN_{AB}, but not their private keys.

Now, the process of agreeing on the same shared group key will be described.
First of all, A and B agree on a key shared only by themselves: in_{AB}. A computes in_{AB} by using its own private key a and B's public key B. B does the other way around.

$$in_{AB} = a \cdot B = b \cdot A$$

The public version IN_{AB} of in_{AB} is directly determined from in_{AB}. Indeed,

IN_{AB} is computed as following:

$$IN_{AB} = in_{AB} \cdot G$$

, where G is the generator (as known as the base point). It is a parameter of the elliptic curve.

Now, peers A and B compute the group key in_{ABC} from their own private shared key in_{AB} and C's public key C.
C computes in_{ABC} from the public version of the key shared between A and B IN_{AB} and its own private key c.

$$in_{ABC} = in_{AB} \cdot C = c \cdot IN_{AB}$$

5.3.3 Join Operation

As broached in subsection 4.1, the join operation is very simple.
When a node is added into the network, the administrator updates its own Diffie-Hellman tree.
First, it creates a new root node whose left child will be the former root of the tree. Its right child will be the new node to be added.

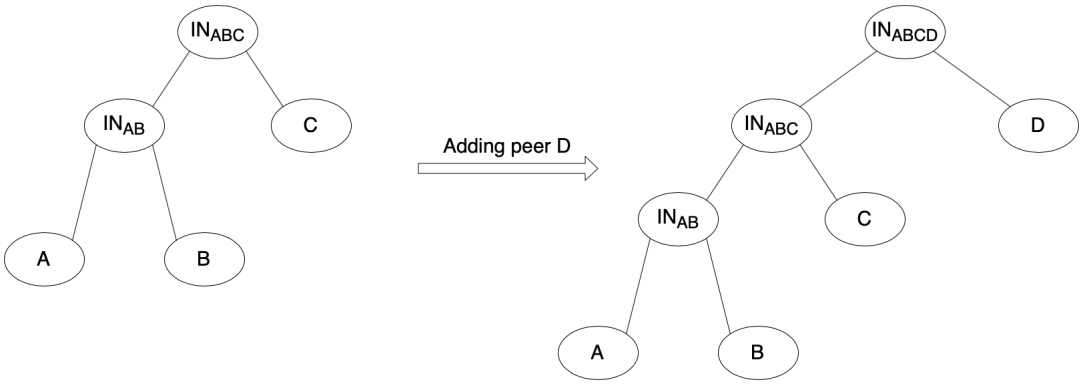

Figure 6: New node D is added into the Diffie-Hellman tree

Then, A computes the new group key as following:

$$gk_{new} = gk_{old} \cdot D$$

, with gk_{new} being the new group key, gk_{old} being the old group key (before the new peer D was added) and D being the public key of the newly added peer D.
Then, A produces a public version of its Diffie-Hellman tree. The public tree only contains public keys. This public tree is then sent to the newly

added peer D, so that it can compute the shared group key. D computes the group key as following:

$$gk_{new} = d \cdot IN_{ABC}$$

, where gk_{new} is the new group key, IN_{ABC} is the public key corresponding to the node $\mathrm{IN_{ABC}}$ (it is actually the public version of the former group key), and d is the private key of the peer D.
Regarding the other peers that was already in the network before D was added, they apply the same computation as A in order to determine the new group key.

5.3.4 Leave Operation

When the administrator removes a peer from the network, it first updates its Diffie-Hellman tree. It removes the node corresponding to the leaving peer from the tree, as well as its parent. Then, the left sibling of the removed node takes the place of its parent.
Figure 7 shows how the tree is modified after deleting peer C from the network. The left sibling IN_{AB} of C takes the place of its parent IN_{ABC} and thus becomes the left child of IN_{ABD} (which is actually IN_{ABCD}.

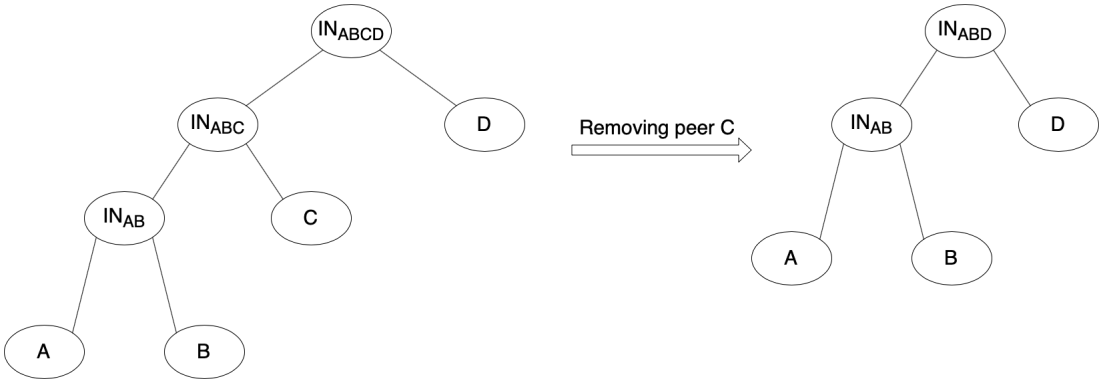

Figure 7: Peer C is removed from the network

Then, the administrator (i.e., peer A) has to recompute the shared keys up to the root and broadcast a public version of its Diffie-Hellman tree to the other peers so that they are able to compute the new group key.
The complexity of the process of computing the keys up to the root is linear according to the number of peers. This is why the leave operation has a linear complexity with the number of peers in the network.

6 Benchmarks

Some benchmarks were conducted in order to analyse how the key agreement system behaves according to the number of peers in the system.
First, 100 peers have been added to the network. For each of them, the time needed for the admin to compute the new group key has been recorded. Figure 8 shows that the time does not increase with the number of node and is quite constant.

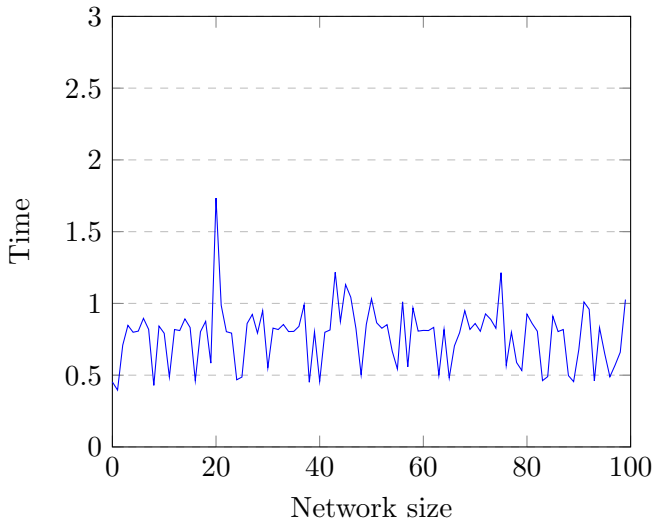

Figure 8: Time to compute the group key according to the number of peer already in the network

Then, the leave operation was also benchmarked.
After adding all the 100 peers into the network, they have been removed starting from the first peer that was added. As Figure 9 shows, the more node in the network, the longer it takes to remove one node (which is located at the bottom of the tree).

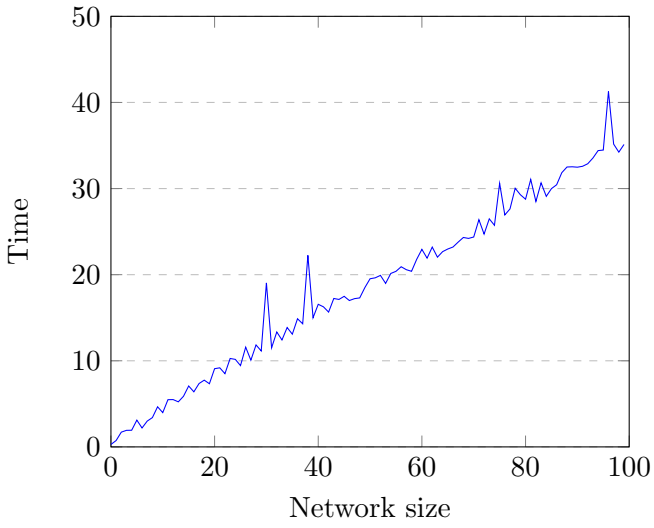

Figure 9: Time to compute the group key according to the number of peer already in the network

7 Additional Tasks

7.1 The Save Button

Before, when the user wrote some text in the web interface, every operation was saved directly into the database, which introduced some latency.
Now, the changes are kept in memory and persisted into the database when the user clicks on the save button.
Note that a listener has been set to the web interface so that it sends a request to persist the state whenever the web page is unloaded. This way, if the users forget to clicks on the save button, their changes will be persisted anyway when leaving the web page.

7.2 Invitation Management

Before when the administrator invited a new peer, it did not send an invitation to the new peer. The database had to be copied and pasted to the new peer so that it can collaborate.
Now, when the administrator invites a new peer, it sends it a new packet containing the necessary information to collaborate to the document:

- **DocId**: the id of the document
- **DocTitle**: the title of the document
- **CreatorId**: the id of the creator of the document

- **CreatorAddr**: the IP address of the creator
- **AcOperationContent**: the Access Control Operation corresponding to the invitation
- **DHRoot**: the root of the Diffie-Hellman tree
- **DHNodes**: the nodes of the Diffie-Hellman tree

Note that the root and nodes of the Diffie-Hellman tree are needed by the joining peer to be able to compute the group key.

8 Next Step

The next step to improve the key agreement system would be to implement the third paper [3] in order to ensure the **post-compromise security** property. This way, the users will have stronger security guarantees even if a peer's secret has been compromised by an adversary.

References

[1] Yongdae Kim, Adrian Perrig, and Gene Tsudik. Communication-efficient group key agreement. In Michel Dupuy and Pierre Paradinas, editors, *Trusted Information*, pages 229–244, Boston, MA, 2001. Springer US.

[2] Yongdae Kim, Adrian Perrig, and Gene Tsudik. Simple and fault-tolerant key agreement for dynamic collaborative groups. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*, CCS '00, pages 235–244, New York, NY, USA, 2000. ACM.

[3] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 1802–1819, New York, NY, USA, 2018. ACM.

[4] D. G. Steer, L. Strawczynski, W. Diffie, and M. Wiener. A secure audio teleconference system. In *Proceedings on Advances in Cryptology*, CRYPTO '88, pages 520–528, Berlin, Heidelberg, 1990. Springer-Verlag.