**Overview**

This project goal consists of a messaging app that allows its users to send messages, the focus here being that they are end-to-end encrypted, not only are we focused on the message’s privacy but also other concepts such as key storage, privacy, availability, and encrypted message searching. To help us achieve this, we used a Java Framework called SpringBoot that helps us abstract the communication configuration for HTTPS, database access and management, server deploying, configuration, and many other features that help us separate the components. Talking about components, we developed 3 main components q to help us achieve our end goal, here follows a small overview of each one of them:

* Client: Component that is responsible for the authentication with the PKG, communication between users (including group/topic conversations), message replication and encryption, key storage, and replication (via Shamir Secret Sharing), the client allows its users to Search Keywords into the NAS (Network Attached Storage) with the property that these queries remain private and their results are encrypted, and last but not least responsible for encapsulating secret Keys through an IBE Scheme and it’s the first backup from a friend client.
* Server (Public-key generator): The main server is responsible for signing up and authenticating clients into the system, but its most important role is that it behaves as a Public-Key generator, it generates public and private keys related to client-side encryption, for this, we used encryption schemes such as Identity-based encryption, used for the communication between 2 users; and Key-policy Attribute-based encryption used for the group/topic chats.
* Cloud (Network Attached Storage): This component is mostly related to remote storage, specifically shares related to Shamir Secret Sharing, Encrypted Message Storage, and Index Storage related to the Cash’14 Dynamic Searchable Encryption Scheme and also the related search capability, since it’s shared storage it validates with a token from the PKG who triggered a storage action, finally it also generates KPABE and encapsulates secrets keys with it, to help users create group chats.

Note that we created the Cloud and Server Components to be highly scalable since multiple instances can be easily added, with the only need for a common database (In the delivery we only used one instance of each component).

To persist our data in a local database that uses project file scope to store our messages, the database connection to be used can be easily changed by changing the configuration file and specifying the information needed for the connection, like URL, user info, certificates, ETC.

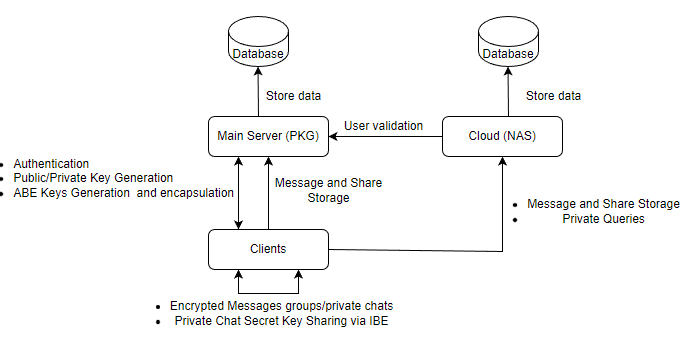
To visualize the results and functionality of the System we provided the Client with a REST interface that allows an easy implementation for a local front end, to visualize and use the interface we will provide later a brief tutorial on how to use the application through the API that the client offers, and also provide a Postman Collection.

Figure 1: Diagram Overview of the system components and interactions

**Authentication**

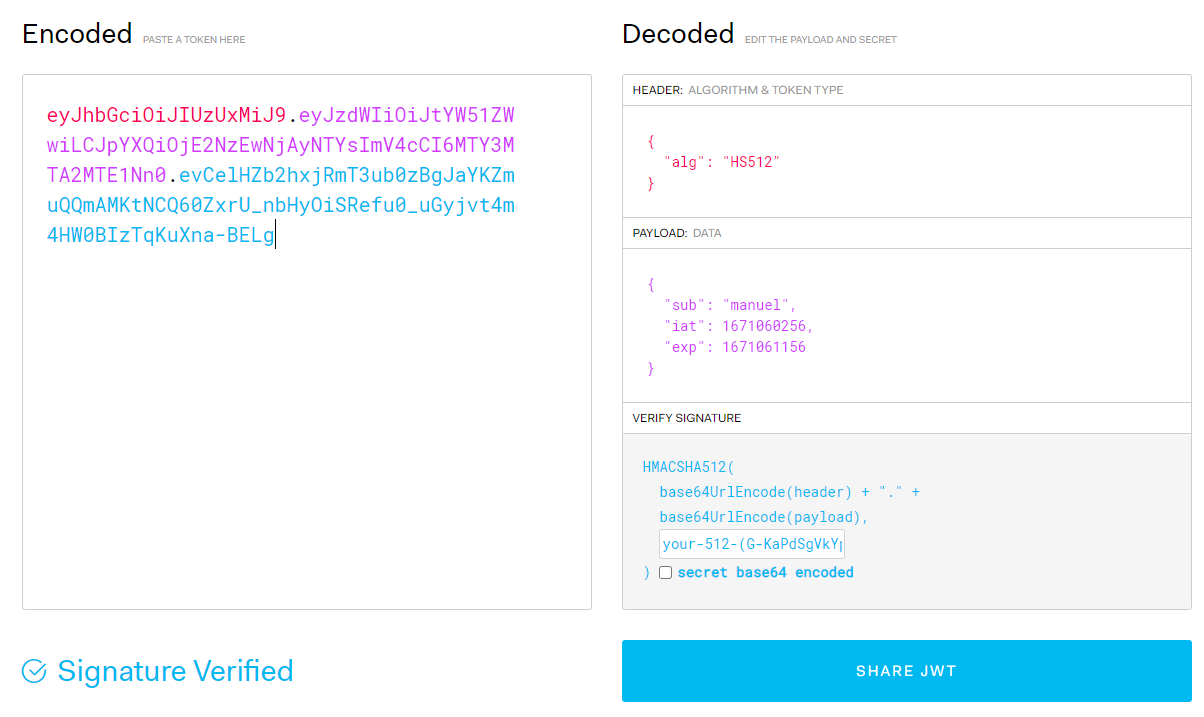
Our Authentication is based on JSON web tokens (JWT), step-by-step we will introduce how it was done and what security guarantees it provides. “*JSON Web Tokens are an open, industry-standard RFC 7519 method for representing claims securely between two parties*”, A JWT is composed of 3 parts: header; payload, and signature. The Header contains 2 fields, the algorithm used for the signature(“HMAC-SHA256”), the payload will contain the userId; Issue date, Expiration date, and finally, the signature is the HMACSHA512(base64UrlEncode(header) + "." + base64UrlEncode(payload), 512-bit-secret), this is an HMAC algorithm that uses a SHA hash function with 512 bits will improve the strength of the signature, this signature will provide the tokens integrity and user´s authentication.

Figure : System JWT exemple

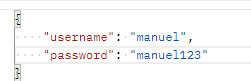
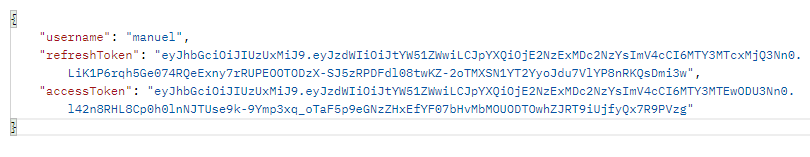
Now looking at how the flow works, first the client signs up to the server providing his username, password, and address. Then the server uses a library called Bcrypt that uses strong cryptography to hash and salts passwords based on the Blowfish cipher, to store our passwords. After the sign up the user is now able to log in, and when this happens the server creates 2 JWT from 2 different master keys for generation and validation, one JWT is called the access token and the other is called the refresh token, the access token allows the users to access functionalities(if the server validates the token) and the refresh token allows the user´s to refresh access tokens from time to time, this is useful because if an attacker captures the user token, he has only a small window of time to exploit since it expires in 7 minutes, and it can also be revoked from the database, and if an attacker, for instance, manages to capture a refresh token and get access to infinite access tokens when his activity is detected that refresh token can also be revoked by removing it from the database. This access token also allows users to identify themselves across the system since for important information like share retrievals and key sharing the entity that is receiving the request always validates the token with the PKG to prove the requester’s identity. Note: minute by minute the access token is refreshed and this timer can change between [1,6] minutes.

Figure 3: Signup/Signin Request(left) and Signin response(right)

**Private Message Chat (Between 2 People)**

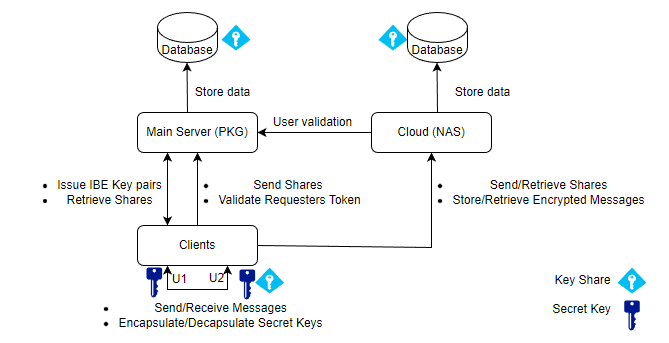
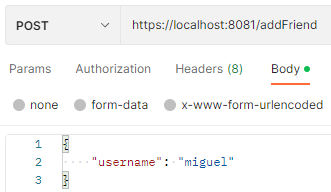
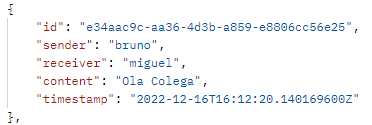
To develop this feature, we used the Identity-based encryption scheme, the way this works is, after the client authenticates himself with the server, then he probes the server to check for its health state, then if everything is OK, the client requests the server (PKG) the public key and his private key generated from the id that his access token contains, after this, the client will have in his side these pair of keys, it’s important to mention that IBE needs a secure channel for the key sharing so for that we used HTTPS that is a secure protocol that runs on top of TLS. When a user(U1) opens the chat with another user (U2) two things will happen, first, the U1 will request U2 for an encapsulated Secret Key, if U2 has it, he will send it to U1 will decapsulate it by using his IBE private key, after the key extraction U1 will use the Shamir secret sharing scheme and will split the secret key into 3 shares (U2, PKG, NAS) and send it to their holders. If U2 does not have this encapsulated Secret Key, U1 will try to retrieve the secret key shares from the NAS and the PKG and will reconstruct the secret key. Finally, if he cannot retrieve the necessary key shares he will encapsulate a new secret key, by using the U2 id along with the public key, in order to encrypt it using the IBE scheme. It’s important to mention that at the end of this process U1 will try to first retrieve the encrypted messages from U2 and in the last case it will retrieve them from the NAS. Note: in the future, more NAS can be added and it’s only necessary to change the number of shares and if needed the threshold.

Figure 4: IBE Scheme and Messaging Flow

Relatively to the Messaging part of this, when U1 wants to send a message to U2 he encrypts it using Symmetric Encryption (AES/GCM/NoPadding) and a 16-byte IV, then sends it to U2 and the NAS, if U2 is offline he only sends it to the NAS and U2 will later be able to retrieve it from it. AES in Galois/Counter Mode provides authentication, integrity, and privacy by using AES block cipher in counter mode along with a polynomial MAC based on Galois field multiplication. By using this encryption algorithm, when U2 receives a message and tries to decrypt it, he can always know its authenticity of it by analyzing the MAC (assuming that U1 is the only one with the secret key). After receiving or sending a message the User tokenizes the string into keywords and updates the Dynamic SSE scheme maps in order to be able to search them in the future, this will be explored in greater detail later.

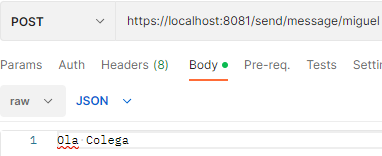




Figure 5Chat opening; Message Sharing and encrypted message (BASE64 Encoded)

Topic/Group Message chat

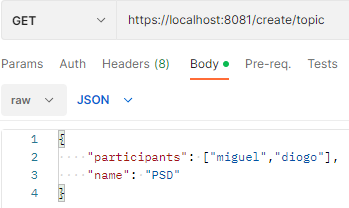
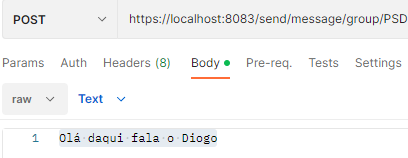
To develop this feature, we used Key-Policy Attribute-based encryption, first, the server starts by generating a Master and Public key for the algorithm, as the name says, the private keys from the users will contain the policy that defines the attributes a cipher has to have in order to the key be able to decrypt it. Since we used the library cloud crypto, the attributes and policy need to be integers, so for that, we needed to hash our string attributes in an integer interval, so collisions are a weakness of this solution, for the key policy, we opted for using the hashed(topic\_category) AND hashed(participantID), and our cypher-text will have the attributes related to the hashed(topic\_category) as well as all the hashed names for all the participants of the topic. Now, this is how it works, a user requests the PKG to create a topic/group, by providing the topic name and participants, the PKG will prepare an encapsulated key for the users containing the topic and all the usersIds that will participate in the chat, after this the users that participate in the group need to subscribe to the topic (including creator), the PKG then validates that the user has access to the topic and build his private key with the policy specified above, then this key is returned along with the encapsulated secret key and all the participant addresses for communication. Every time a user subscribes to a topic, he will request the NAS for any messages that exist, this way if other people send messages before he joined, he will never lose them. When the user wants to send a message to the topic he encrypts it the same way as the private chats, and sends the encrypted message to every online person on the topic, when a user receives/sends a message, he tokenizes the string into keywords and updates the Dynamic SSE scheme maps in order to be able to search them in the future, this will be explored in greater detail later.



Figure 6: Group Creation; subscription; Message Sharing

Dynamic Searchable encryption

This functionality is present in the NAS and Client components and to develop it we were inspired by the Cash’14 Dynamic Searchable Encryption Scheme. The scheme starts with the NAS creating an empty index map in the database and the client creates a counterMap that stores keywords and their count. When a client receives/sends a message he will tokenize it into keywords, for each keyword, the client will create k1 and k2 from a PRF(K,w||”i”), then the client will create the index label from a PRF(k1||c) where c is the current keyword count, we constructed the PRF through the use of the” HmacSHA256” algorithm, after this the client creates the index value, wich is the encrypted MessageID with K2 using the “AES/GCM/NoPadding” and a random IV, then the client sends the encoded indexLabel and indexValue to the NAS, that will store them in the DB, to execute the search the client generates K1 and K2 and sends them along with the keyword to search. Then the NAS starts with a counter set to 0 and he applies PRF(k1|C) and while he finds indexLabels for increments of C he will decrypt the index value with K2, and will store the message in a response list, after he no longer finds index labels, he will stop searching and send the encrypted messages to the client. Although we could achieve forward privacy with re-encryption instead we used a non-optimal solution that uses different keys for the same keyword based on the conversation that the keyword and document are related to

