# Sistema di anonimizzazione

## Obiettivo

L'obiettivo del nostro progetto è stato creare un semplice sistema di anonimizzazione basato sull'onion routing che permettesse lo scambio di messaggi in modo half-duplex, ovvero trasmissione bilaterale alternata, tra un client ed un server passando per un nodo intermedio.

## Strumenti

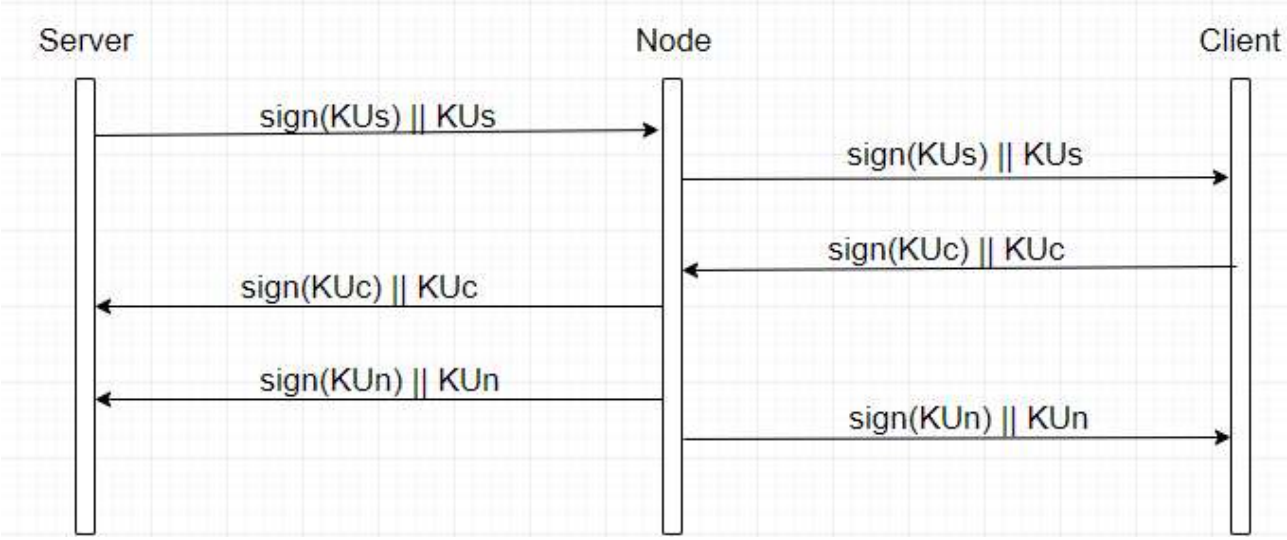
Il sistema è stato sviluppato in Java utilizzando l'ambiente di sviluppo Eclipse ed è composto da un unico progetto contenente le tre classi che una volta eseguite andranno a comporre l'insieme (Server.java, Node.java, Client.java) e tre classi create con lo scopo di semplificare alcuni procedimenti (Communication.java, SendRSA.java, AES.java). La comunicazione è stata creata utilizzando il protocollo UDP.

## Funzionamento

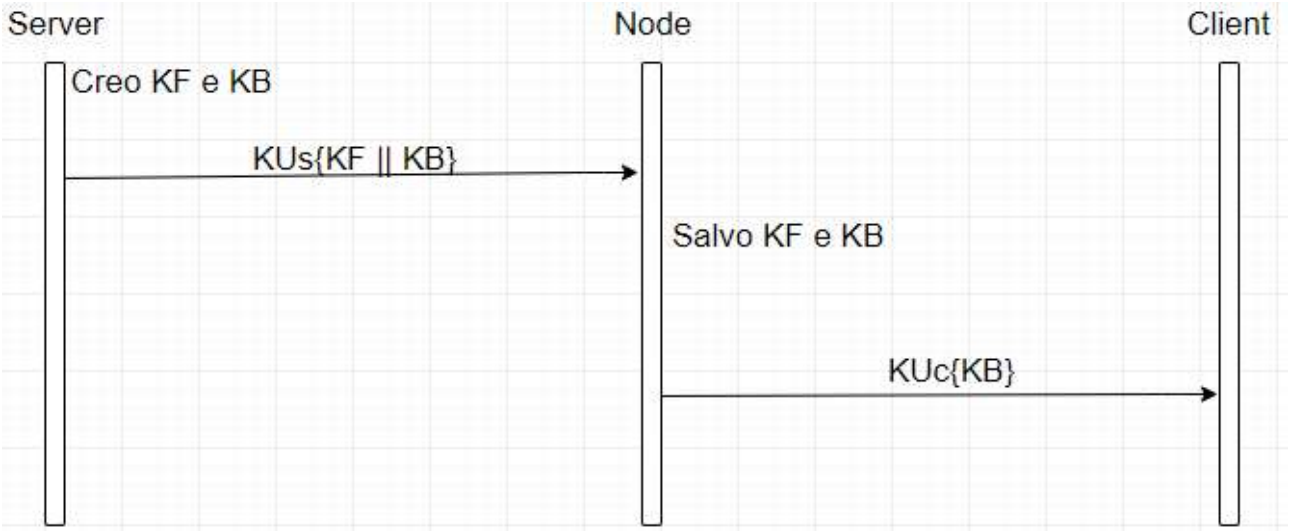
Come già detto il sistema consente una comunicazione client-server passando per un nodo intermedio. Essendo basato sull'onion routing i messaggi da server a client verranno cifrati utilizzando le chiavi segrete di forward, mentre la comunicazione da client a server sfrutterà le chiavi segrete di backward. Queste chiavi vengono utilizzate per cifrare i messaggi mediante algoritmo di cifrature AES-ECB.

La chiave di forward e quella di backward vengono entrambe generate in modo casuale dal server e comunicate al nodo e al client crittandole con le rispettive chiavi pubbliche, in modo che solo loro possano decifrarle utilizzando le chiavi private e garantendo la confidenzialità delle chiavi sfruttando la cifratura asimmetrica RSA.

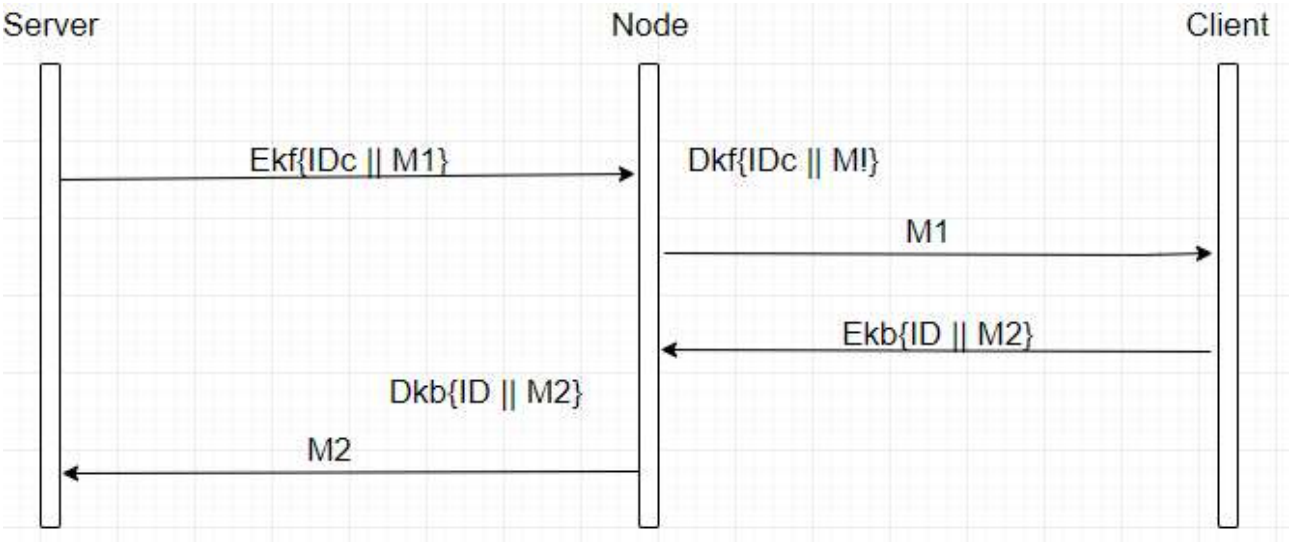
Prima di tutto però avviene la creazione delle chiavi pubbliche e private da parte di ognuno dei nodi e lo scambio delle chiavi pubbliche tra loro. Durante lo scambio la chiave pubblica viene concatenata al suo hash (SHA-256) per garantire l'integrità del messaggio ricevuto. Le fasi di scambio sono le seguenti:



Il successivo scambio delle chiavi di forward e backward invece avviene nel seguente modo:




Dopo tutti questi scambi sarà possibile scambiare messaggi in modo anonimo tra Client e Server. La comunicazione avviene quindi nel seguente modo:




Per terminare l'esecuzione basta mandare un messaggio '**exit**' dal server o dal client

## Sviluppo

La classe Communication.java permette di creare delle connessioni UDP semplificate specificando solo quali sono le porte che si vogliono utilizzare, infatti per inviare e ricevere messaggi sarà sufficiente utilizzare i metodi `send(String s)` e `receive()`, mentre con `getReceive()` si accede al contenuto di ciò che è arrivato. Utilizzando questa classe creiamo un collegamento tra Server e Nodo e tra Nodo e Client. UDP permette di mandare messaggi in entrambe le direzione in modo alternato sfruttando lo stesso canale.

Una volta creati i canali ogni elemento utilizzerà la classe SendRSA.java per creare in modo casuale le proprie chiavi pubbliche e private, una volta generate quella pubblica verrà resa nota agli altri nodi nel modo precedentemente illustrato. Questa classe fornisce i metodi per cifrare con chiave privata e decifrare con chiave pubblica (`encrypt(String plaintext), decrypt(String ciphertext)`) e viceversa(`encryptTo(String plaintext), decryptFrom(String ciphertext)`). La seconda coppia di metodi è stata ideata per far in modo che un nodo A possa mandare un messaggio a B

cifrando il messaggio con la chiave pubblica di B e garantendo così la confidenzialità del messaggio. La chiave pubblica è condivisa con gli altri nodi spedendo in un unico messaggio sia il modulo che l'esponente utilizzando i metodi `getMod()` e `getExp()`.

Dopo che le chiavi pubbliche sono state scambiate il Server genera casualmente delle chiavi da 256 bit per l'algoritmo di cifratura AES-ECB. Per semplificare il procedimento sia di generazione delle chiavi (di backward e forward) che di cifratura e decifratura è stata creata la classe AES.java che offre tutti i metodi necessari. Infatti il costruttore senza parametri permette di creare un oggetto che generi casualmente le chiavi, mentre quello che permette di inizializzarle con valori esterni sarà usato dai nodi che dovranno ricevere per messagio le chiavi. Per sapere i valori delle chiavi si usano i metodi `getForward()` e `getBackWard()`, mentre per cifrare e decifrare ci sono i metodi `encrypt(byte[] key, byte[] text)` e `decrypt(byte[] key, byte[] text)`. In questo caso però le chiavi non sono state mandate insieme come si voleva fare inizialmente ma sono state mandate con due messaggi separati dato che RSA non riusciva a decifrare una lungezza maggiore a 64 byte, ma grazie alla cifratura con chiavi pubbliche la confidenzialità durante il trasferimento permane.

Per finire, una volta che sono state scambiate le chiavi, inizia la comunicazione vera e propria. Per far partire il procedimento di scambio di chiavi spiegato sopra è necessario inserire il primo messagio che il Server spedirà al Client. Questo verrà poi cifrato con la chiave di forward, spedito al Nodo che lo decifrerà e manderà il messagio in chiaro al Client. Dopodichè il Client potrà mandare un messaggio al Server utilizzando lo stesso procedimento, ma in questo caso verrà utilizzata la chiave di backward. Per terminare tutti o processi basta spedire dal Client o dal Server il messaggio **exit**