Implementazione di una chat in linguaggio Java con crittografia asimmetrica

Progetto del corso di Reti di Calcolatori

Valentino Marano, Amerigo Mancino

Giugno 2015

Indice

1 Presentazione degli obiettivi

Il nostro progetto per il corso di Reti di Calcolatori si è concentrato sullo sviluppo di una applicazione di messaggistica per reti locali (LAN). Si è progettato il programma in modo tale da sfruttare il protocollo UDP per lo scambio dei messaggi, i quali si ipotizzano essere criptati tramite crittografia asimmetrica con chiave pubblica/privata. Per di più, l'applicazione è stata pensata per essere in grado di inviare anche allegati (che definiamo, a questo proposito, come file generici quali immagini, audio, etc), anch'essi criptati. Il programma, teoricamente, deve essere in grado di identificare tutti gli utenti connessi alla LAN, così da rendere possibile scegliere con quale utente si desidera chattare, similmente a quanto avviene con applicazioni note quali Skype o Telegram Desktop. Inoltre deve godere di un'interfaccia grafica intuitiva per semplificare le operazioni.

2 Progetti a confronto

L'idea della chat non è di certo innovativa. Ne esistono di molti tipi da diversi anni e ognuna di loro ha manifestato il proprio momento di gloria, per poi abbandonare i riflettori e dirigere gli utenti verso applicazioni più nuove e immediate.

Nell'ambito dei progetti realizzati in passato per il corso di Reti di Calcolatori, in particolare, nel 2006 uno studente ha proposto un servizio di messaggistica su connessione cifrata con SSL ad architettura p2p, non realizzando tuttavia tutte le premesse e abbandonando quindi l'implementazione della cifratura con SSL e, di conseguenza, la gestione dei certificati. Questo lavoro era stato svolto sfruttando la libreria Axis di Apache Foundation.

Nel 2014 altri due studenti hanno realizzato una chat prevedendo un'unità centrale che potesse immagazzinare i dati strettamente necessari sugli utenti, senza quindi affidare dati ridondanti a tutti i client. Il loro obiettivo era quello di creare un servizio che fosse a metà fra le chat peer to peer e le chat centralizzate.

3 Sviluppo degli Obiettivi

Quasi tutte le parti poste nella Presentazione degli Obiettivi sono state correttamente sviluppate e implementate. La chat realizzata possiede una GUI per l'interazione con l'utente avente il seguente aspetto:

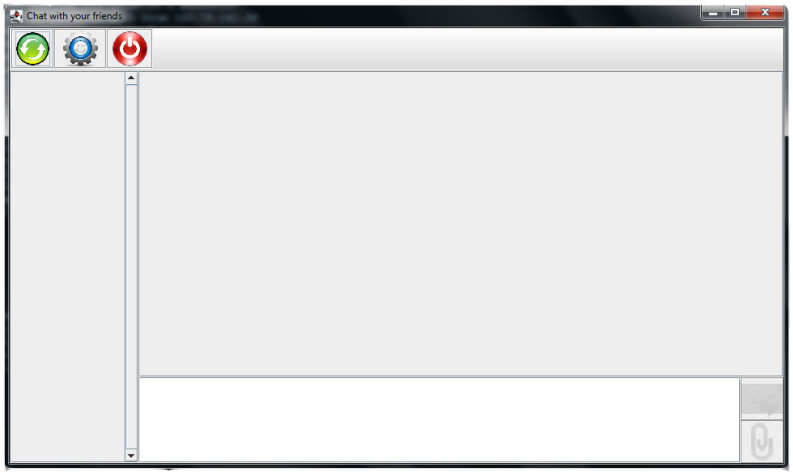

Figura 1: GUI principale

Il riquadro in basso permette la scrittura di testo. I due tasti laterali ad esso affiancati consentono invece di inviare messaggi (criptati) o allegati (non criptati). In alto sono presenti tre bottoni: di essi, quello più a sinistra consente di scandire la rete per la ricerca di eventuali utenti che, in questo momento, stanno utilizzando la chat, tali utenti vengono salvati in una lista di utenti conosciuti. La scansione viene in ogni caso eseguita in automatico all'avvio del programma e ripetuta periodicamente durante la sua esecuzione per aggiornare la suddetta lista. Quando un utente viene trovato, nel riquadro a sinistra dell'applicazione viene inserito un pulsante con il suo nickname. Cliccandoci sopra, diventa possibile chattare con lui e inviargli messaggi. Il bottone al centro, invece, apre un'altra piccola finestra.

Questa finestra gestisce le impostazioni e da qui l'utente può regolare:

- ogni quanti secondi far eseguire la scansione della rete;
- ogni quante scansioni svuotare completamente l'insieme degli host conosciuti in modo da non conservare inutilmente host non più presenti nella LAN o che abbiano chiuso l'applicazione;
- cambiare il suono di notifica che verrà riprodotto all'arrivo di un messaggio.

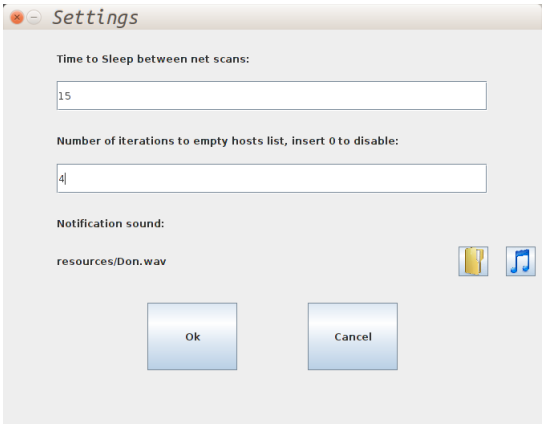

Figura 2: Pannello delle impostazioni

I pulsante più a destra invece consente di chiudere l'applicazione. Prima della chiusura verrà mostrato un messaggio di conferma per chiedere all'utente se vuole davvero terminare l'applicazione. Prima di essere chiuso, il programma manda a tutti gli host conosciuti un messaggio speciale *##DOWN##* in modo che gli altri host sulla LAN sappiano che si sta disconnettendo e lo tolgano subito dalla lista degli host conosciuti. All'avvio del programma si cerca il file di configurazione e si cerca di leggere il nome utente e le impostazioni. In caso non venga trovato il file o il nome utente non sia presente (è il primo accesso fatto dall'utente) compare una schermata di login:

Figura 3: Finestra di Login

In questa scheda l'utente crea il proprio account, scegliendo un nickname da usare in chat. Una volta digitato il nickname, il tasto di conferma diventa abilitato e premendolo si effettua il login, consentendo finalmente l'uso vero e proprio dell'applicazione. Viene poi controllata la presenza della coppia di chiavi e nel caso non siano presenti vengono create.

In questa relazione, analizzeremo a grandi linee la codifica proposta e l'implementazione delle varie classi, con particolare enfasi sul ruolo giocato

da ognuna di esse. Mostreremo, poi, i risultati ottenuti e, infine, il funzionamento dell'intero programma qui presentato, esponendo anche i test fatti in corso d'opera in locale e sulla Virtual Private Network di ateneo. Il lavoro svolto sarà, quindi, accompagnato dalle dovute conclusioni e da un commento sui possibili sviluppi futuri.

4 Il package app

Il codice è stato diviso in due package: uno che si occupa di gestire l'applicazione e la codifica dei messaggi, e l'altro che invece contiene tutte le classi inerenti allo scambio e alla trasmissione dei messaggi stessi. Il package `app` contiene tutte e sole le classi che si occupano del corretto funzionamento dell'applicazione, a partire dalla definizione della schermata di login fino ad arrivare alla gestione delle chiavi. Analizzeremo più dettagliatamente nei prossimi sotto-paragrafi l'uso delle singole classi.

4.1 Avvio dell'applicazione

La classe `Main` è la classe principale dell'applicazione. Possiede un unico metodo `main` che si occupa di:

1. caricare il nickname da file assieme alle impostazioni o richiederlo tramite la schermata vista in precedenza se non è presente;
2. controllare la presenza delle chiavi e se necessario crearle;
3. controllare la presenza di tutte le directory necessarie all'esecuzione e creare quelle mancanti;
4. eseguire il metodo `run` per far partire l'applicazione vera e propria.

4.2 Interfaccia Grafica

La classe `App`, che implementa `Runnable`, si occupa della creazione e della gestione della finestra grafica della chat. La GUI, in particolare, è stata realizzata mediante Swing, un framework per Java orientato allo sviluppo di interfacce grafiche. I principali metodi qui presenti si occupano, quindi, di generare la GUI, ma non solo: le conversazioni avvenute fra gli utenti vengono salvate in locale e, se la conversazione con un utente già noto viene ripresa in un qualche futuro, allora i precedenti messaggi scambiati con lui sono caricati da un file del tipo

`nickname.txt`

e mostrati nella schermata principale dell'applicazione grazie al metodo `openChat()`. Il file di configurazione `.chat`, invece, è un file che permette di tenere traccia del nickname dell'utente. Quando avvia l'applicazione per la prima volta, infatti, compare la schermata grafica di login mostrata in Figura 3 per l'inserimento di un username. Il tasto `OK` è disabilitato fintanto che l'utente non ha scritto almeno una stringa valida, ossia una stringa più lunga di zero caratteri. Premendo il tasto di conferma, viene salvata nel file con estensione `.chat` una stringa del tipo:

`NICK:` *nickname*

insieme ad altre informazioni di configurazione, descritte in precedenza.

4.3 Cifratura dei messaggi

La classe principale che implementa la cifratura dei messaggi è la classe `Encryption`. La codifica sfrutta l'algoritmo a chiave pubblica RSA (dal nome di coloro che lo hanno proposto, Rivest, Shamir, Adleman), il quale utilizza operazioni in modulo per generare le chiavi. Per violare la chiave privata bisognerebbe effettuare un attacco a "forza bruta" generando quindi un problema computazionalmente non trattabile. Per realizzarle, è stata usata la classe `KeyPair` di Java, che permette di generare una coppia di chiavi (pubblica e privata). Queste vengono estrette e salvate in locale su due file nascosti `.public.pem` e `.private.pem` e usate poi per criptare e decriptare i messaggi. Ogni utente possiede una coppia di chiavi, quindi se un utente A deve spedire un messaggio ad un utente B, è necessario che A possieda la chiave pubblica di B con la quale cripterà quindi il messaggio in modo che solo B possa decriptarlo usando la sua chiave privata.

5 Il package net

Nel package `net` sono organizzate tutte le classi che si occupano di effettuare e gestire lo scambio dei messaggi e di ricercare altri utenti nella rete.

5.1 Ricerca di utenti

La classe che si occupa della scansione è la classe `Scan`. Per effettuare la ricerca viene utilizzato `Nmap` (http://nmap.org/), un software libero distribuito con licenza GNU GPL creato originariamente per effettuare port scanning, cioè mirato all'individuazione di porte aperte su un computer bersaglio o

anche su range di indirizzi IP, in modo da determinare quali servizi di rete siano disponibili. Un tipico esempio dell'uso di `Nmap` è il seguente:

```
nmap www.google.it

Starting Nmap 6.40 ( http://nmap.org ) at 2015-06-05 09:47 CEST
Nmap scan report for www.google.it (74.125.206.94)
Host is up (0.026s latency).
Not shown: 998 filtered ports
PORT    STATE SERVICE
80/tcp  open  http
443/tcp open  https
```

Nello specifico, poi, per quanto concerne la nostra applicazione, `Nmap` viene utilizzato principalmente per individuare quali host sono presenti nella LAN. Per eseguire `Nmap` viene adoperata la classe `Process` di Java. Quando viene avviata una nuova scansione si invia a tutti gli host raggiungibili in LAN un messaggio del tipo *##NICK:nickname##* in cui "nickname" rappresenta il nickname dell'utente. Chi lo riceve risponderà con il proprio nickname. Tutte le coppie degli host raggiungibili che hanno risposto vengono salvate tra gli host conosciuti in una mappa a doppia chiave come coppia

$$\langle \text{Nickname, Indirizzo} \rangle$$

e vengono mostrati nella barra laterale sinistra della GUI, come già anticipato sopra. Osserviamo in particolare che la classe `Scan` estende la classe `Thread` costituendo quindi un thread a parte rispetto al resto dell'applicazione.

5.2 Scambio di messaggi

Lo scambio di messaggi non sfrutta, come si è intuito dai paragrafi precedenti un'architettura client-server, ma il peer-to-peer. Fondamentalmente, tutti i nodi sono equivalenti e possiedono un thread server sempre in ascolto per la ricezione dei messaggi ed un client predisposto, invece, all'invio dei messaggi stessi. Abbiamo creato dunque una classe `Message` che implementa l'ADT (Abstract Data Type) *messaggio*, contenente fondamentalmente un campo di tipo stringa per il testo, un campo per la data e un capo `tipo` posto a 0 se il messaggio è stato ricevuto e posto a 1 se inviato.

Lo scambio vero e proprio è implementato invece nelle classi `Chat_manager` e `Server`, cuore dell'intero programma. Nella prima vengono svolti vari compiti:

- caricamento, e eventualmente aggiornamento, del file audio che verrà poi riprodotto per notificare l'arrivo di un messaggio o di un allegato;

- avvio dei thread relativi alla classe `Server` e alla classe `Scan`;
- creazione, aggiornamento e gestine della struttura dati *HashBiMap* usata per memorizzare gli host conosciuti;
- invio dei messaggi e degli allegati (alla pressione dei pulsanti mostrati nella Figura 1) tramite i metodi `send()` e `attach()`;
- la gestione di una mappa a doppia chiave, realizzata anch'essa tramite la classe `HashBiMap`, per contenere i messaggi inviati che si è sicuri dell'arrivo. A ogni invio la coppia

  $$\langle \text{Destinatario, Messaggio} \rangle$$

  è aggiunta alla mappa e verrà tolta dalla struttura dati solo alla ricezione di un ack da quel destinatario per quel messaggio. L'invio di qualsiasi tipo di messaggio (messaggio normale, allegato, messaggio "speciale") è effettuato tramite la classe `DatagramSocket` via UDP.

I messaggi speciali sono dei messaggi usati per la sincronizzazione tra gli host e la gestione della rete:

- *##NICK:nickname##* Come visto prima questo messaggio viene inviato dalla classe `Scan` a tutti gli host raggiungibili in modo che conoscano il nickname dell'utente;
- *##NICK:nickname#$#* Questo messaggio viene inviato in automatico dalla classe `Server` come risposta alla ricezione di un messaggio *##NICK:nickname##*, è codificato in maniera diversa dal precedente in modo che il Server della controparte non risponda a tale messaggio;
- *##RESZ:size##* Questo messaggio viene inviato appena prima dell'invio di un allegato in modo che il Server del destinatario allarghi in automatico il buffer di ricezione abbastanza da ricevere l'allegato. Tale buffer verrà poi in automatico reimpostato al default di 256 Byte;
- *##RCVD:message##* È l'ACK nominato sopra che il Server invia in automatico alla ricezione di un messaggio (né allegato né messaggio speciale) al mittente in modo da fargli sapere che il messaggio è stato ricevuto;
- *##DOWN##* Come detto precedentemente è il messaggio che l'applicazione manda a tutti gli host conosciuti appena prima di essere chiusa.

La classe `Server` ha come scopo principale quello di avviare il server inizializzando poi il campo `myIP` e permettendo dunque ad altri utenti di poterlo identificare. Dopodiché si mette in attesa di eventuali datagrammi. Alla ricezione di ogni datagramma controlla se è criptato (messaggi) o in chiaro (allegati e messaggi speciali) e se necessario lo decripta. I messaggi speciali, come visto precedentemente, vengono gestiti in automatico dal Server, per i messaggi e per gli allegati viene invece interpellata la classe `Chat_manager` che riprodurrà il suono di notifica e salverà nel file *nickname.txt* il messaggio o il nome dell'allegato. In caso di allegato, aprirà automaticamente una finestra per chiedere all'utente dove salvare l'allegato ricevuto. Se invece si tratta di un messaggio e la chat con l'host mittente è già aperta viene aggiornata la schermata e aggiunto il messaggio alla chat visualizzata. Nel caso non sia attualmente aperta la chat con l'host mittente oltre alla riproduzione del suono di notifica viene colorato di rosso il bottone laterale.

6 Conclusioni e sviluppi futuri

Il lavoro svolto ha permesso un completo sviluppo di un'applicazione di messaggistica pienamente funzionante e operativa su reti locali. I test fatti in locale e su VPN hanno dato risultati soddisfacenti permettendo un pieno scambio di messaggi e file (si è testato, ad esempio, l'invio di una semplice immagine e di qualche file di testo). L'uso dell'interfaccia grafica realizzata con SWING - siamo certi - permette una visualizzazione rapida dei messaggi e semplifica molto l'uso dell'applicazione anche ad utenti meno avvezzi all'uso di un terminale. Inoltre durante le ultime prove abbiamo verificato ulteriormente la correttezza del nostro lavoro provando a catturare i pacchetti con *Wireshark* (`https://www.wireshark.org/`) e abbiamo osservato che i messaggi in chiaro erano leggibili mentre gli altri messaggi erano effettivamente criptati e quindi illeggibili a chiunque faccia sniffing della rete.

Nonostante l'applicazione abbia raggiunto una certa solidità, numerosi sono i possibili interventi da attuare per migliorarla.

- Sarebbe interessante criptare anche gli allegati oltre che i messaggi al fine di garantire una maggiore riservatezza;
- I file contenenti le chat sono memorizzati sul PC dell'utente in chiaro, il che li rende vulnerabili. Di fatto, si potrebbero cifrare e decifrare ogni volta che viene chiamato il metodo `openChat` (ossia ogni volta che viene premuto un pulsante nella barra a sinistra dell'applicazione), introducendo tuttavia in questo modo un overhead che, durante lo sviluppo,

abbiamo deciso di non aggiungere per non appesantire ulteriormente il programma;

- Un discorso molto simile vale per il file di configurazione che pur essendo nascosto potrebbe essere modificato;

- Si potrebbe aggiungere un supporto Android per l'applicazione, utilizzandola per far comunicare più utenti nella stessa rete Wi-fi;

- La sicurezza della chat è migliorabile se si aggiunge anche un controllo per mezzo di un certificato oppure un supporto per l'inserimento di una firma digitale durante lo scambio di messaggi;

- Un altro problema presente che potrebbe essere risolto in futuro è legato ai nickname: la cronologia dei messaggi viene salvata, come detto precedentemente, in un file `nickname.txt`; Questa decisione sta in piedi fintanto che non esistono due utenti con lo stesso nickname (visto che ipotizziamo che un utente non possa cambiare nickname se non modificando il file *.chat*), nel qual caso una sovrapposizione è inevitabile, portando alla perdita di entrambi gli storici delle conversazioni che si mischierebbero in un unico file. Una soluzione ipotizzata comporta l'inserimento di un server centrale predisposto alla gestione dei nickname;

- Altri miglioramenti attuabili riguardano poi prettamente l'interfaccia grafica del programma, inserendo la possibilità di cambiare colori, trasformando i messaggi in nuvolette, cambiare lo sfondo della conversazione e altre modifiche per rendere l'applicazione più user-friendly;

- Potrebbe essere utile dare la possibilità all'utente di visualizzare le informazioni relative a ogni messaggio come il timestamp di invio e ricezione, la lettura di tale messaggio da parte dell'utente;

- Una modifica utile sarebbe l'introduzione dei gruppi, come in qualsiasi applicazione di messaggistica.