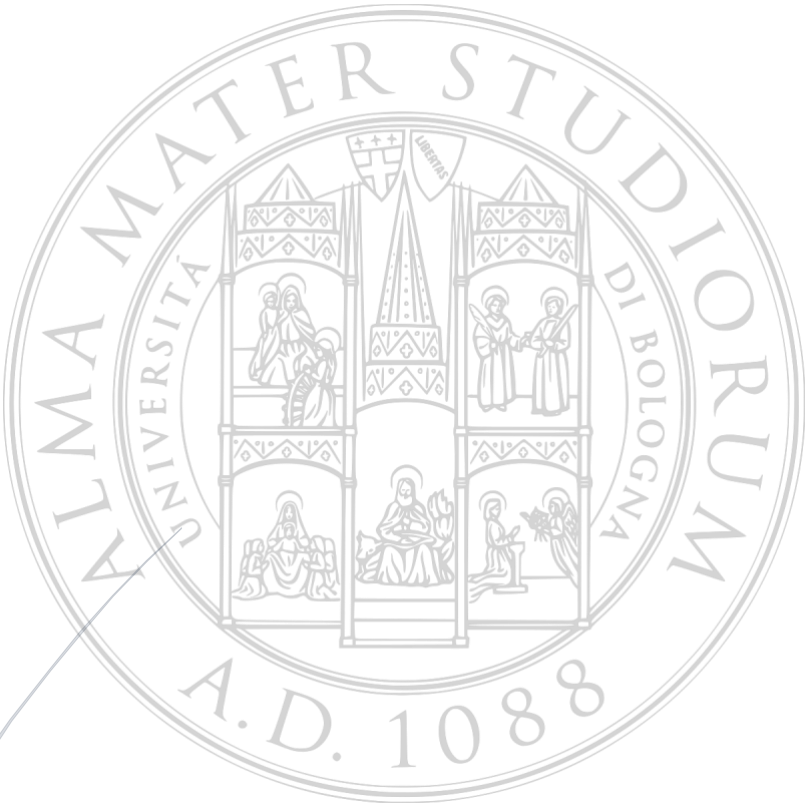
Progetto 2019/2020

JOMANJI

Sistemi operativi



AUTORI
Fantini Omar
Lazar Andreea F.
Tabanelli Simone
Sacchetti Giacomo
Stefani Flavio

Nome Gruppo JOMANJI
Indirizzo mail referente omar.fantini@studio.unibo.it

Componenti:
Fantini Omar 840120
Lazar Andreea Florina 889375
Sacchetti Giacomo 892855
Stefani Flavio 883331
Tabanelli Simone 890230

INDICE

Introduzione

Questo progetto nasce dal corso di Sistemi Operativi per la Laurea Triennale in Informatica per il Management dell'università di Bologna durante l'anno accademico 2019-2020.

All'interno di questo PDF troverete i contenuti utili e le spiegazioni di "Jomanji", un sistema di messaggistica con un'architettura di rete paritaria (peer to peer).

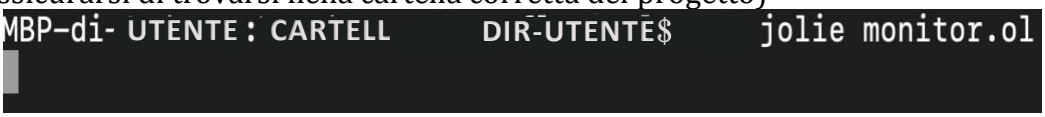
I peer possono scambiarsi messaggi in modalità pubblica o privata, rispettivamente attraverso gruppi o in singolo a patto che la comunicazione sia crittata tra questi.

A continuazione prima della descrizione della rete JOMANJI, potrete visualizzare una DEMO sul come far funzionare il programma.

DEMO JOMANJI

Per eseguire il progetto Jomanji 2019-2020, seguire i seguenti passi:

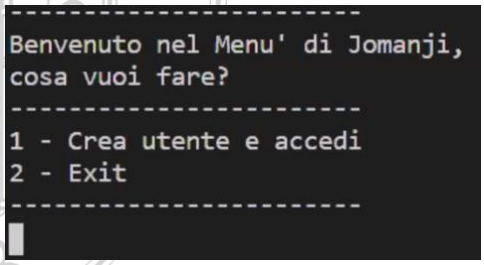
I. Avviare un nuovo terminale e scrivere *jolie monitor.ol.*
(Assicurarsi di trovarsi nella cartella corretta del progetto)



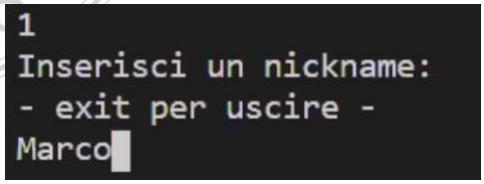
II. Avviare un altro terminale e scrivere *jolie server.ol.*
P.S.: Sul primo terminale avviato, se. Tutto è andato correttamente comparirà il messaggio che vediamo in figura:

III. Avviare infine un numero di terminali pari al numero di utenti che si vogliono collegare alla rete, per ciascuno scrivere *jolie jomanji.ol.*

IV. Per ogni utente creato è possibile scegliere tra crea o accedere al proprio account (*tasto 1*) o uscire dal programma (*tasto 2*).

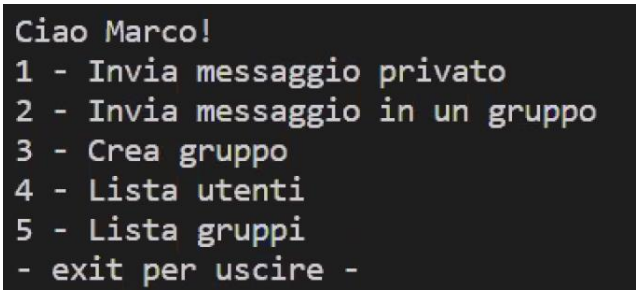


V. Se si sceglie di proceder alla creazione o accesso presso l'account, verrà chiesto un nome per identificarsi (*in questo caso abbiamo Marco che vuole accedere alla rete*).



Una volta eseguito il login con successo, comparirà un menù contenenti le varie operazioni con cui i peer potranno interagire con il programma.

In particolare, le operazioni sono le seguenti:



VI. Selezionando 1, il fruitore potrà inviare un messaggio privato ad un altro peer se quest'ultimo esiste. Per fare ciò, basterà scegliere e poi scrivere, dalla lista che apparirà, il nome della persona con cui si intende iniziare una conversazione.

Marco vuole, nella versione esemplare, inviare un messaggio in privato ad Anna. Infatti preme 1 e poi scrive il nome di Anna sulla console.

```
1
-----------------------
A chi vuoi inviare il messaggio: ?
 - exit per uscire -
-----------------------
Anna
Giulia
Davide
-----------------------
Anna
```

Esempio di conversazione tra i 2:

```
Chat con Anna iniziata
Anna@Marco_Anna:ciao Marco, come stai ?

Tutto bene, grazie tu?
```

```
Chat con Marco iniziata
ciao Marco, come stai ?
Marco@Marco_Anna:Tutto bene, grazie tu?
Marco@Marco_Anna:
```

VII. Premendo il tasto 2, il peer potrà inviare un messaggio in un gruppo se quest'ultimo esiste.
Per fare ciò, basterà scegliere e poi scrivere, dalla lista che apparirà, il nome del gruppo con cui si intende conversare.

```
2
-----------------------
In quale gruppo vuoi scrivere: ?
 - exit per uscire -
-----------------------
Pallavolo Bolo
Family
Progetto Sistemi
-----------------------
Progetto Sistemi
```

Come possiamo vedere dalla figura, Marco sceglie di entrare nel gruppo "Progetto Sistemi".

VIII. Scegliendo 3, l'utente avrà la possibilità di creare un nuovo gruppo pubblico con il nome designato da lui.

```
3
-----------------------
Come vuoi chiamare il gruppo? Progetto Sistemi
Creato il gruppo Progetto Sistemi
-----------------------
```

```
3
-----------------------
Come vuoi chiamare il gruppo? Pallavolo Bolo
Creato il gruppo Pallavolo Bolo
-----------------------
```

A continuazione una simulazione tra Anna, Davide e Giulia sul gruppo "Pallavolo Bologna" – stiamo visualizzando la chat di Giulia:

```
Chat nel gruppo Pallavolo Bolo iniziata
Stasera partita?
Davide@Pallavolo Bolo:Yess, al campetto? ( Verified )
Anna@Pallavolo Bolo:Vengo anche io, alle 17? ( Verified )
Perfetto
```

IX. Optando per il tasto 4, si potrà prendere visione della lista di tutti gli utenti registrati.

Possiamo vedere a destra la lista dei quattro utenti collegati alla rete in quel momento. →

```
4
-----------------------
Ci sono 4 utenti
-----------------------
Marco
Anna
Giulia
Davide
```

X. Preferendo infine il tasto 5, si potrà visualizzare la lista di tutti i gruppi presenti.

Nella foto affianco possiamo vedere che una volta cliccato su 5, Marco può vedere i tre gruppi diversi presenti. →

```
5
-----------------------
ci sono 3 gruppi
-----------------------
Pallavolo Bolo
Family
Progetto Sistemi
-----------------------
```

XI. Concludendo, se si desidera uscire in qualsiasi momento da un menù si potrà usufruire della chiave "*exit*".

```
exit
-----------------------
Disconnessione in corso
```

N.B: Si esce completamente dall'account e si dà la possibilità all'utente di iscriversi nuovamente in una nuova socket con un username diverso.

Struttura della Rete

La prima idea da cui siamo partiti era quella di realizzare una rete totalmente decentralizzata dove ogni client fosse anche un server, dove i peer fossero dei nodi su una struttura assimilabile ad un grafo, in tal senso avevamo la completa decentralizzazione della rete.
Ci siamo quindi posti alcuni quesiti di implementazione riguardo a questa prima idea, nello specifico ci siamo domandati:

"Come gestire le iscrizioni e le posizioni all'interno della rete?"
"Come gestire le chat singole e di gruppo?"
"Come gestire il salvataggio su file?"
"Come gestire le liste degli utenti?"

In principio avendo una rete totalmente decentralizzata avevamo ideato una struttura dove ogni peer fosse in grado di garantire l'accesso alla rete ad ogni altro nuovo utente in arrivo. Si assegnava un identificativo e un indirizzo al nuovo peer della rete e comunicandolo a cascata agli altri fruitori della stessa, si aggiornavano quindi le posizioni comuni sul file di supporto.
In altre parole, volevamo consegnare a ogni peer una lista con tutte le posizioni degli altri utenti della rete in modo che ognuno avesse la facoltà di comunicare a proprio piacimento con chiunque fosse collegato. Dunque, ci siamo imbattuti nel primo problema, ossia:

se due o più nuovi utenti cercano di entrare in contemporanea nella rete, cosa succede?

In tal caso le liste si aggiornano in contemporanea e nell'invio ad ogni peer vi è la possibilità che una delle liste vada persa, che si sovrascriva la lista contemporaneamente o ancora peggio che metà dei peer abbiano una lista e l'altra metà un'altra, si suddividerebbero in due sotto-reti e non potrebbero più comunicare tra di loro.

Una possibile soluzione a questo problema sarebbe l'implementazione di un semaforo (non dentro ai peer, altrimenti il problema sussisterebbe), una "waiting list" dove i possibili nuovi utenti possono essere inseriti solo uno alla volta.
Una soluzione semplice dato che in tal modo solo un peer alla volta può essere utilizzato per i nuovi accessi. Questa soluzione fa perdere però di efficacia al sistema perciò abbiamo deciso di scartarla.
In seguito, abbiamo anche affrontato il problema dal punto di vista della sicurezza, dato che per il nostro team è uno dei punti su cui ci siamo maggiormente concentrati.
Se tutti gli utenti hanno la possibilità di scrivere e così modificare la lista delle posizioni degli altri peer per aggiungere i nuovi utenti, cosa impedirebbe a un possibile male intenzionato di modificare gli indirizzi degli utenti o addirittura aggiungere posizioni fittizie all'interno della rete?

Ci spieghiamo meglio con un esempio:

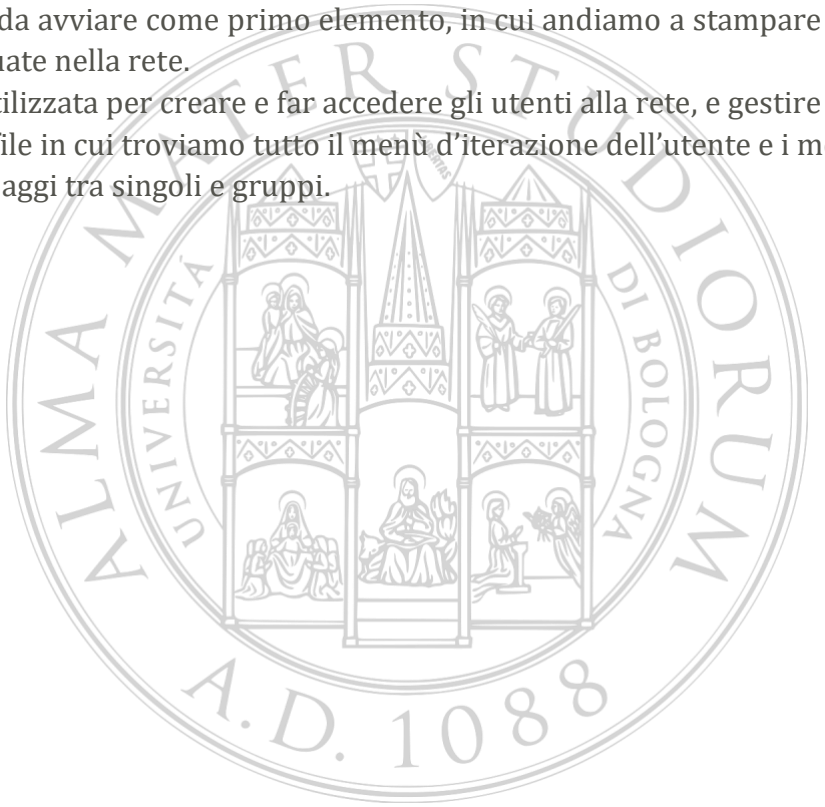
> Mettiamo il caso in cui Trudy riesce a impersonificarsi in BOB, e quindi riesce a far risultare che l'indirizzo corrisponda sempre a lui, anche se in realtà la chat è gestita da lei.

> Se Trudy inviasse la lista a tutti i peer della rete, BOB sarebbe escluso da essa in quanto non potrebbe più ricevere messaggi dagli altri utenti.
> Mettiamo anche il caso in cui ALICE utilizzando questa lista, messaggi a BOB, inconsapevole di star scrivendo a una persona incognita, e a sua insaputa gli vengano sottratti dati sensibili.

La soluzione che proponiamo è dunque quella di costruire un server di supporto, sul quale salvare la lista, e anche se un peer decidesse di modificare la stessa, verrebbe comunque effettuato un controllo dal server che dà garanzia della veridicità degli utenti. Nella costruzione finale del progetto abbiamo superato anche questo modo di vedere la rete per poter eliminare vari controlli al livello server e ridurre il tempo di esecuzione del progetto.

In sintesi, abbiamo deciso di strutturare il nostro progetto con tre file principali:

- **Monitor**: da avviare come primo elemento, in cui andiamo a stampare ogni operazione effettuate nella rete.
- **Server**: utilizzata per creare e far accedere gli utenti alla rete, e gestire i vari controlli.
- **Jomanji**: file in cui troviamo tutto il menù d'iterazione dell'utente e i metodi per inviare i messaggi tra singoli e gruppi.



Peer

I peer sono associati al numero di utenti che vogliono utilizzare l'applicazione.
Ogni peer identificherà un utente.
La struttura che abbiamo costruito voleva essere il più semplice possibile, infatti i fruitori della nostra rete sono indentificati da 2 principali caratteristiche:

- Un nome utente.
- La capacità di mandare e ricevere messaggi.

Ogni peer per essere definito tale nella rete Jomanji, deve rispettare le sopracitate caratteristiche.

Parlando della creazione dei peer la nostra idea è la seguente:

Step 1:
La rete è vuota, soltanto il server e il monitor sono accesi e perciò non vi sono peer nella rete.

Step 2:
Viene avviato *jomanji.ol*, da qui si potranno iniziare a inserire uno alla volta per terminale, gli utenti nella rete. Ogni volta che viene inserito un nuovo nome si effettua un controllo per vedere se è disponibile.
In caso affermativo, *jomanji.ol* assegna un identificativo al nuovo fruitore e una socket nascosta. Ci assicuriamo così che l'utente non verrà mai a conoscenza della sua vera posizione nella rete e non potrà mai modificarla.

Step 3:
A questo punto il peer viene salvato sulla lista in memoria nel server e può iniziare a fruire dei servizi da essa offerti.

La nostra struttura non permette però la rimozione degli utenti, ossia ogni utente può effettuare il logout ma la socket e il suo account non verranno liberati fino alla fine della sessione del server. Questo significa che per poter usufruire nuovamente del servizio Jomanji dallo stesso terminale, bisogna registrarsi con un nuovo nome utente.
Questo ci garantisce due cose:

In primo luogo, la tutela dei "dati" dell'utente.
In secondo luogo, non liberando la socket evitiamo di esporre la rete ad attacchi di terzi.
Poniamo infatti il caso in cui:
un utente, che chiameremo Bob, sia iscritto alla rete e decida di rescindere i termini di utilizzo, se l'account fosse rimosso e la socket liberata, la rete sarebbe esposta a un attacco. In particolare, se la socket non viene riempita subito con un nuovo indirizzo e la lista aggiornata, vi è la possibilità che un utente ostile tenti di riempire la socket vuota con delle credenziali fittizie e sottrarre dei dati sensibile agli altri utenti.

Gestione delle chat

Ogni peer dopo l'accesso, può scegliere di inviare un messaggio (guardando le apposite liste) ad un utente privato oppure a un gruppo.

```
define _stampaListaGruppi
{
    NumeroGruppi@serverPort()(num);
    println@Console("ci sono "+num+" gruppi")();
    println@Console( "----------------" )()  ;
    for(i=0,i<num,i++){
        PrendiNomeGruppo@serverPort(i)(gruppo);
        println@Console(gruppo)()
    }
    println@Console( "----------------" )()
}
```

```
define _stampaListaUtenti
{
    NumeroUtenti@serverPort()(response);
    println@Console("ci sono "+response+" utenti")();
    println@Console("---------------")();
    for(i=0,i<response,i++){
        PrendiNomeUtente@serverPort(i)(utente);
        println@Console(utente)()
    }
    println@Console("---------------")()
}
```

Le chat singole come le chat dei gruppi vengono salvate su file.
Nel caso di chat private, il salvataggio è avviene su un file unico, ogni file rappresenta una chat.

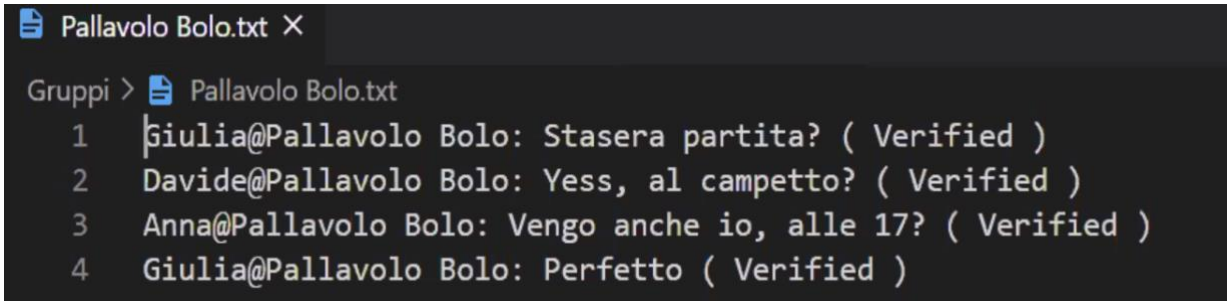
Il metodo *sendMessagge* (foto sottostante), invia il messaggio al destinatario della chat scelta dalla lista delle persone disponibili. Lo stato serve per gestire i terminali in cui vengono stampati i messaggi durante i cambi di conversazione.

```
[ sendMessage( request )( response ) {
    if ( is_defined( global.tokens.( request.token ) ) ) {
        NomeChat = global.tokens.( request.token ).NomeChat;

        //questo foreach gestisce l'invio dei messaggi nella chat privata quindi verra eseguito una sola volta
        foreach( u : global.chat.( NomeChat ).users ) {
            jomanjiPort.location = global.chat.( NomeChat ).users.( u ).location;
            if ( u != global.tokens.( request.token ).username ) {
                for(i=0,i<#global.utenti,i++){
                    if(global.utenti[i].nome==u && global.utenti[i].stato == NomeChat){
                        aa=true
                    }
                }
                if(aa){
                    with( msg ) {
                        .message = request.message;
                        .chat_name = NomeChat;
                        .username = global.tokens.( request.token ).username
                    };
                    setMessage@jomanjiPort( msg )
                }
                undef(aa)
            }
        }
    }
    else {
        throw( TokenNotValid )
```

La gestione dei gruppi è effettuata appoggiandosi sul server, dove le chat sono pubbliche. Gli utenti possono creare dei gruppi pubblici nei quali inviare i messaggi.
Proprio come per i peer, chiunque avendo il riferimento può scrivere messaggi sul gruppo e al contempo, se è all'interno di esso, può leggere i messaggi.

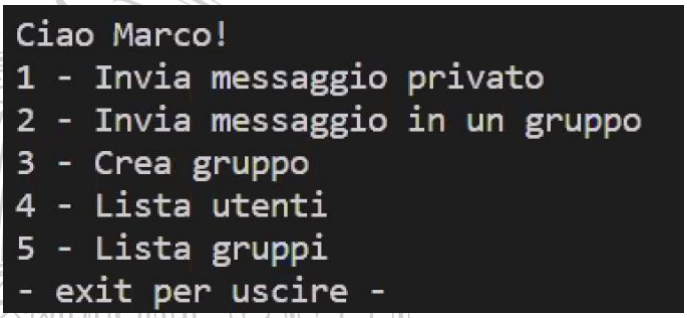
Le chat vengono salvate su un file unico senza necessità di fare salvataggi per ogni utente che partecipa alla conversazione (vedi foto sottostante).



Appena avviato un file *jomanji.ol*, l'utente viene posto davanti alle due scelte:

1. Creare un utente oppure accedervi
2. Uscire.

Una volta riusciti a loggarsi i peer sono davanti a un menù come il fax-simile visto a fianco.



- *Premendo 1*, il peer può inviare un messaggio a un altro utente collegato.
- *Premendo il tasto 2* il peer può inviare un messaggio in un gruppo se quest'ultimo è esistente.
- *Premendo il tasto 3* l'utente può creare un gruppo pubblico.
- *Premendo il tasto 4* si può visualizzare la lista degli utenti.
- P*remendo 5* si può visualizzare la lista dei gruppi.
- Infine, scrivendo *exit* l'utente può disconnettersi.

Crittografia

Abbiamo utilizzato la crittografia mista, a chiave *pubblica* e *privata*, sia per i messaggi di gruppo che per lo scambio di messaggi tra gli utenti delle singole chat.
In particolare, potete trovare l'algoritmo Rivest, Shamir e Adleman (RSA) e lo Secure Hash Algorithm (SHA-256).

Chat di gruppo
La crittografia delle chat di gruppo è molto semplice, in quanto utilizziamo un algoritmo di SHA256 per la firma digitale. L'obiettivo che ci siamo prefissati è stato quello di lavorare non solo sulla sicurezza ma anche sulla semplicità di maneggevolezza del progetto; in altre parole non volevamo usare un algoritmo troppo pesante che rallentasse in maniera significativa il lavoro di invio e ricezione messaggi. Abbiamo perciò scelto di gestire nel seguente modo i messaggi pubblici.
Una volta che un utente invia un messaggio in un gruppo allega la propria firma digitale (in automatico), al messaggio che viene inviato. Quando il messaggio viene ricevuto il server accerta che sia stato effettivamente inviato da parte dell'utente (il quale dichiara di averlo scritto), e non da una terza persona nella conversazione.

La prima fase è il *Digest* (metodo del codice) dove un algoritmo di SHA-256 prende in ingresso un token e il messaggio del *sender* e li trasforma in una stringa alfanumerica.
Dopodiché il messaggio in chiaro viene criptato con la chiave privata del *sender* in modo da certificare che sia stato lui a mandare il messaggio e soprattutto per non mandare in chiaro il messaggio.
Il server riceve il messaggio criptato e una stringa criptata con SHA-256, applica la chiave pubblica del *sender* al messaggio criptato e lo confronta con la stringa criptata.

Ci spieghiamo con un esempio:

> Mettiamo il caso in cui MARCO vuole inviare un messaggio nel gruppo pubblico "Sistemi Operativi", all'interno di questo gruppo possiamo trovare anche Anna, Davide e Giulia. Quando MARCO invierà il messaggio, questo verrà criptato con lo SHA256 facendo diventare il plaintext in una stringa alfanumerica. Dopodiché, il plaintext viene criptato con la chiave privata di MARCO, in questo modo quest'ultimo dimostra di aver inviato lui il messaggio ai suoi amici. Anna, Davide e Giulia nel ricevere il messaggio di Marco e la stringa alfanumerica, applicano la chiave pubblica del sender al chipertext e ottengono così il plaintext. Una volta ottenuto il messaggio in chiaro si riapplica lo SHA e se la nuova stringa alfanumerica coincide con quella iniziale di MARCO, comparirà tra parentesi la conferma dell'avvenuto controllo del messaggio di quest'ultimo – VERIFIED.

Chat privata

La crittografia delle chat private è strutturata in maniera medesima alle chat di gruppo. L'idea di partenza era quella di lavorare solo nel senso della crittografia asimmetrica con un algoritmo di AES.
Un algoritmo che, nonostante sia simmetrico, è molto robusto ma particolarmente sensibile ad attacchi brut force e laterali.
Sorge poi anche il problema delle chiavi simmetriche, *come possiamo scambiarci la chiave in modo sicuro? Implementiamo una sola chiave di default per tutte le chat?*
Ovviamente non possiamo procedere in tal senso, poiché significherebbe distruggere tutta la sicurezza su cui ci siamo concentrati fino ad ora. Basterebbe scoprire la chiave di cifratura di una chat per conoscere poi il contenuto di tutte le intere chat della rete.

L'idea di base è sfruttare i punti di forza di RSA.
Gli utenti possono criptate usano la chiave pubblica del *receiver* in modo che il messaggio sia illeggibile a tutti tranne che al *receiver* che, con la propria chiave privata, riesce a "sbloccare" il messaggio.

Ci spieghiamo meglio con un esempio:

> Mettiamo il caso in cui MARCO vuole parlare con ANNA, decide di scriverle un messaggio in privato. Nel momento in cui lui invia il messaggio, un metodo lo cripta con la chiave pubblica di ANNA. Quest'ultima nel momento in cui riceve il messaggio, lo decripta con la sua chiave privata. Se non ci sono state alterazioni nel mezzo la fase di decriptazione va a buon fine e ANNA può leggere il messaggio iniziale di MARCO.