Advanced Programming of Cryptographic Methods Project

Luca Dal Bianco e Paolo Piasenti
Ottobre 2020

1 Introduzione

Per svolgere il lavoro assegnatoci abbiamo sviluppato tre diversi progetti denominati: *RECEIVER*, *SENDER* e *DATABASE*.
Il Receiver è scritto in Java mentre gli ultimi due sono implementati in JavaFx, quindi dotati di grafica per interagire con l'utente. I primi due, inoltre, interagiscono con delle librerie scritte in C tramite le JNI. Lo scopo di questo progetto è quello di simulare l'invio di una foto, con i relativi tags, da parte di un utente (Sender) verso un server centrale (Receiver). Il passaggio di tale informazione avviene tramite la rete Web e più precisamente attraverso un Socket, quindi è stato necessario cifrare tale messaggio utilizzando una cifratura AES che a sua volta richiede la cifratura delle chiavi AES tramite RSA. Tutta la parte di crittografia è scritta in C in modo da ottimizzare il processo computazionale richiesto dalla cifratura e decifratura di un messaggio.

2 Le nostre implementazioni

In questo capitolo riassumiamo brevemente le nostre scelte implementative ed esplicitiamo meglio alcuni passaggi non del tutto ovvi. Innanzitutto, visto che abbiamo sviluppato il progetto in uno pc con un'alta risoluzione grafica e succesivamente abbiamo fatto girare il tutto su un pc con una più scarsa risoluzione ci siamo accorti che le dimensioni non erano ottimali per entrambi

i pc e dunque abbiamo inserito a un fattore di scala (scalar_factor) all'inizio delle classi Sender.java e DataBase.java, che può essere modificato da chiunque desideri ridimensionare le finestre interattive. Per chi conosce un po' la matematica, diciamo che il fattore scalar_factor opera come un'omotetia.

2.1 *SENDER*

Il Sender è composto da cinque classi JavaFx: 'Sender', 'MyTextField', 'LoadBtn', 'SubmitBtn' e 'Client'.

- Il 'Sender' è la classe principale (main) e gestisce tutta e solo la parte grafica.
- 'MyTextField' è un'area di testo che può anche azionare l'invio del messaggio tramite il metodo sendPhoto del SubmitBtn.
- Il 'LoadBtn' ha il compito di aprire un'altra finestra di dialogo da cui l'utente può selezionare uno foto da caricare. Tale foto verrà successivamente posta nel rettangolo al centro del Sender e inserita, sotto forma di byte array, in un'apposita variabile che possa essere letta e rielaborata dal SubmitBtn.
- Il 'SubmitBtn' viene azionato solo una volta che la foto e i relativi tags sono presenti. Esso prepara il messaggio che il 'Client' dovrà cifrare e mandare al Receiver. Per fare ciò concatena, in un array di byte, l'array dei tags e quello della foto, inserendo in più, nel primo byte la lunghezza dell' array dei tags così da poter essere facilmente spezzato una volta mandato al Receiver. (A causa dell'utilizzo del byte per calcolare la lunghezza dell'array di tags siamo costretti a mettere un numero massimo di lettere nei tags. Java vede un byte come un numero da -128 a +127 quindi la lunghezza massima dei tags sarebbe 127 ma per comodità abbiamo optato ad un bound massimo di 100 caratteri). Infine chiama i metodi del 'Client' per lo scambio delle chiavi, la cifratura e l'invio del messaggio. La buona riuscita dell'intera operazione viene annunciata tramite un specifico messaggio che compare nella console, viceversa nel caso in cui l'utente abbia scritto male il messaggio dei tags o non abbia ancora caricato l'immagine, nella console compaiono le relative istruzioni necessarie per mandare la coppia (tags—immagine) in modo corretto.

- Il 'Client' opera tramite il Socket per mandare e ricevere i messaggi che il Sender e il Receiver si scambiano volta per volta. Appena istanziato crea un Socket che crea un collegamento con quello del 'Receiver'. Esso inoltre è addetto a tutta la parte crittografica di tale protocollo, infatti è suo compito recuperare le chiavi di RSA dal canale, generare le chiavi di AES, cifrarle e mandarle al Receiver ed infine cifrare l'intero messaggio ottenuto dal 'SubmitBtn' e spedire anche questo sul canale. Per la cifratura del messaggio vengono chiamati dei metodi nativi (scritti in C) tramite l'utilizzo delle JNI.

2.2 *RECEIVER*

Il 'Receiver' è formato da una sola classe Java. La sua funzionalità è quella di simulare un server che, una volta acceso, aspetta che un qualsiasi Sender si connetta e invii dei messaggi. Per questo motivo l'avvio di tale 'Receiver' è la prima cosa da fare per lanciare l'intero progetto, infatti è a suo carico la creazione del Socket su cui il 'Sender' può comunicare. Una volta che il 'Sender' comunica l'intenzione di voler mandare un nuovo messaggio, esso compie le seguenti operazioni:

- genera le chiavi private e pubbliche di RSA tramite l'utilizzo dei metodi nativi per poi mandare al 'Sender', per mezzo del Socket, solo la chiave pubblica.
- Aspetta che il 'Sender' mandi nell'ordine: le chiavi di AES cifrate tramite RSA che solo lui può decifrare(ancora una volta tramite le JNI), l'IV e l'intero messaggio cifrato tramite AES.
- Decifra l'intero messaggio tramite il metodo nativo decAES scritto in C ed inserisce l'intero messaggio in un array di byte.
- Visto come è stato costruito il messaggio da parte del 'Sender', facilmente è possibile ricostruire la Strigna dei tags e il file dell'immagine che verranno rispettivamente salvate in un'apposita cartella presente nel Database denominata 'New Message'.

2.3 *DATABASE*

Il DataBase è composto da sette classi JavaFx: 'DataBase', 'MyNode', 'MyTree', 'ClearBtn', 'Attention' , 'SearchBtn' e 'ImageViewer':

- Il 'DataBase' è la classe principale (main) e gestisce tutta e solo la parte grafica.

- 'MyNode' crea i nodi da inserire nell'albero 'MyTree'. Ognuno è formato da un tag e dall'insieme delle foto collegate a quel tag. Abbiamo usato un TreeSet ed in particolare un Set per raggruppare le immagini in modo che se uno stesso tag viene ripetuto più volte per descrivere la stessa foto quest'ultima non verrà ripetuta tra la lista delle immagini che il Database mostrerà. In questa classe vengono implementate altri metodi di supporto tipo il toString, l'Equals e il Compare.

- Il 'MyTree' è l'estensione della classe Java TreeSet ed ha come nodi gli oggetti Java 'MyNode'. Questa classe, tramite metodi specifici, gestisce l'aggiornamento dell'albero stesso andando a leggere le righe del file tags&names.txt presente nel cartella New Messages. Ogni riga è composta dai tags (separati da uno spazio) e dal nome dell'immagine, quindi per ogni tag controlla la presenza o meno del relativo nodo all'interno del 'MyTree' per aggiornare la lista delle immagini o per crearne uno nel caso il tag/MyNode sia nuovo. Un apposito metodo permette di scrivere tutti i dati presenti nell'albero in un file data.txt presente nella cartella Database in modo da poter ricostruire il 'MyTree' una volta chiusa l'applicazione DataBase.

- Il 'ClearBtn' è un bottone, posto sulla parte bassa del 'DataBase', che apre un'ulteriore finestra di dialogo con l'utente chiamata 'Attention' ed impedisce l'uso normale del Database se prima non si è consumato uno dei due eventi dell'Attention.

- La classe 'Attention' domanda all'utente se vuole o meno eliminare l'intero database delle foto con i relativi tags. Per questo motivo al suo interno sono presenti due bottoni che confermano o meno la volontà dell'utente di resettare il 'Database', cancellando sia le foto e i tags presenti nell'albero che quelli presenti nei nuovi messaggi.

- Il 'SearchBtn' è un altro bottone che permette di ricercare le foto relative ai tag precedentemente inseriti nel TextField presente nel Database. Questa classe permette la ricerca sia di un solo tag che l'unione o l'intersezione di più tags. Per far ciò abbiamo utilizzato uno dei nuovi strumenti presenti da Java8 in poi: gli Stream. Tutte le foto ottenute

da questa ricerca saranno disposte su tre colonne all'interno del rettangolo centrale presente nel DataBase. Anche in questo caso un'eventuale errore nella scrittura dei tags da parte dell'utente, sarà segnalata da un apposito messaggio di errore. Prima di ogni nuova ricerca viene aggiornato il MyTree con eventuali messaggi presenti nella cartella 'New Message'

- ImageViewer' è il quadrato su cui verrà disposta la disposta l'immagine. La particolarità di questa classe è che permette di aprire la foto. Per far ciò abbiamo dovuto caricare a mano una particolare libreria di Java ed inserirla nello stesso folder del progetto per poi poterla compilare insieme ad esso. Tale azione sarà possibile una volta che l'utente ha eseguito un doppio click sull'immagine presente.

3 Il percorso del messaggio

Ora spieghiamo brevemente l'intero percorso del messaggio. Esso viene costruito ad opera del Sender(SubmitBtn), che crea nel seguente modo un array di byte: trasforma la stringa dei tags in un array di byte per poterla concatenare con l'array di byte ottenuta dall'immagine. Infine, in posizione zero, scrive la lunghezza dell'array dei tags così da poterla facilmente spezzare una volta che il Receiver ha ottenuto il messaggio.

Questo messaggio, sotto forma di array di byte, viene passato ai metodi nativi scritti in C. Essi restituiscono un array di byte contenente il messaggio cifrato. Visto che la cifratura tramite AES richiede che la lunghezza del messaggio sia multipla di 16 byte abbiamo dovuto implementare implementare una funzione specifica (padding) che permette di aggiungere il numero necessario di zeri per ottenere la lunghezza desiderata. Per tenere traccia degli zeri aggiunti e quindi per poter ricostruire il messaggio in modo corretto, abbiamo dovuto inserire un contatore chiamato 'res', in posizione zero del messaggio, che ci permette di tenere traccia degli zeri aggiunti. Solo dopo questa operazioni di 'padding' siamo finalmente pronti a cifrare il messaggio tramite cifratura AES.

A questo punto il Sender può mandare il messaggio cifrato al Reciver, ma prima invia la lunghezza di questo array di byte così che il Receiver possa

preparare in anticipo un array byte della giusta lunghezza per salvare il cifrato mandato dal Sender.

Da qui in poi entra in gioco il Receiver. Esso passa il cifrato alle librerie native scritte in C che permettono di cifrare il messaggio. Innanzitutto vengono decifrati solo i primi 16 byte così da poter recuperare il valore del 'res' e tramite ciò, preparato un array di byte della giusta lunghezza dentro cui salvare il messaggio decifrato e accorciato degli zeri di troppo (quest'ultima azione è ad opera del metodo 'depadding').

Da qui il messaggio verrà spezzato in due array di byte, uno contenente i tags sotto forma di byte e l'altro l'intera immagine. L'array dell'immagine viene letta come tale e salvata un'apposita sottocartella del Database denominata 'New Photos' contenuta nella cartella 'New Message'. In quest'ultima cartella è presente un secondo file, denominato 'tags&names.txt' che viene aggiornato per tenere traccia dei tags relativi all a foto appena mandata. Ogni riga di questo file è formata dai tags e dal nome della foto.

Ogni volta che il Database vuole aggiornare il suo albero controlla se il file 'tags&names.txt' non sia vuoto. In questo caso costruisce un 'MyNode' per ogni tags che trova su una stessa riga con all'interno il nome dell'immagine appena mandata. In più sposta tutte le foto dalla cartella 'New Message/New Photos' alla cartella 'Database/Photos'.

Alla chiusura del 'DataBase' il file 'data.txt' all'interno della sottocartella 'Database' viene rigenerato a partire dall'albero 'MyTree'. Le righe di questo file sono costruite nel seguente modo: Ogni nodo ha una sua riga formata da un tags seguito dal nome di tutte le foto relative a quel tag. Per ricostruire l'albero 'MyTree' basta semplicemente leggere il file riga per riga, creare il 'MyNode' corrispondente ed aggiornare l'albero.

3.1 Lo scambio delle chiavi

Anche il processo dello scambio delle chiavi per ottenere un segreto in comune (che poi nel nostro caso è la chiave di cifratura AES) non è così banale.

Una volta che il Sender comunica di voler mandare un nuovo messaggio, è il Receiver che comincia a calcolare le chiavi private e quelle pubbliche di

RSA. Visto l'alto livello computazionale richiesto è verosimile che sia proprio il Receiver (un server con elevata potenza computazionale) a cimentarsi in questa operazione. Esso chiama un metodo nativo denominato genKeyRSA che tramite l'utilizzo degli mpz di C produce una stringa formata dalla concatenazione, mediante il carattere :, delle cinque chiavi private di RSA. Solo dopo essere ritornati a livello Java il Receiver manda tramite Socket la stringa contenete solo la chiave pubblica RSA.
La genKeyRSA ha la necessità di trovare due numeri primi (p,q) con un centinaio di cifre e per far ciò deve innanzitutto recuperare un random della medesima lunghezza. La funzione random da noi utilizzata è la srand(time(0)) di C che permette di produrre un numero pseudorandom utilizzando come seed il tempo corrente.
A questo punto il Sender recupera la chiave pubblica di RSA sotto forma di Stringa da cui ottiene i valori n ed e sempre sotto forma di stringa. Contemporaneamente genere la chiave e l'IV di AES producendo 16 bytes random ottenuti grazie alla classe SecureRandom di Java. Tornando in C tramite le JNI siamo in grado di cifrare con RSA l'array di byte della chiave trasformando quest'ultima direttamente da array di Java a un numero in mpz di C. Analogamente anche le stringhe n ed e vengono convertite in mpz.
La chiave cifrata è ottenuta sotto forma di Stringa così da essere facilmente mandata tramite Socket, al Receiver. L'operazione di trasformazione in stringa è stata una delle poche idee sensate per gestire un tipo di dato (intero) così grande: fino a che non si scende al livello di C e della libreria GMP non è possibile trattare questi tipi di dati davvero come interi. A questo punto il Sender ha tutte le carte in regola per mandare il vero messaggio cifrato sul Socket.

Il Receiver per decifrarlo ha bisogno innanzitutto di decifrare la stringa contenente la chiave AES precedentemente cifrata dal Sender. Esso, chiamando la funzione nativa decRSA su questa stringa e sulle stringhe relative alle chiavi private di RSA ottiene la chiave di AES già sotto forma di array di byte. In questo caso le JNI trasformano tutte le stringhe in mpz, decifrano la chiave e la riconvertono in un array di 16 bytes già pronto per essere usato dal cifrario AES.