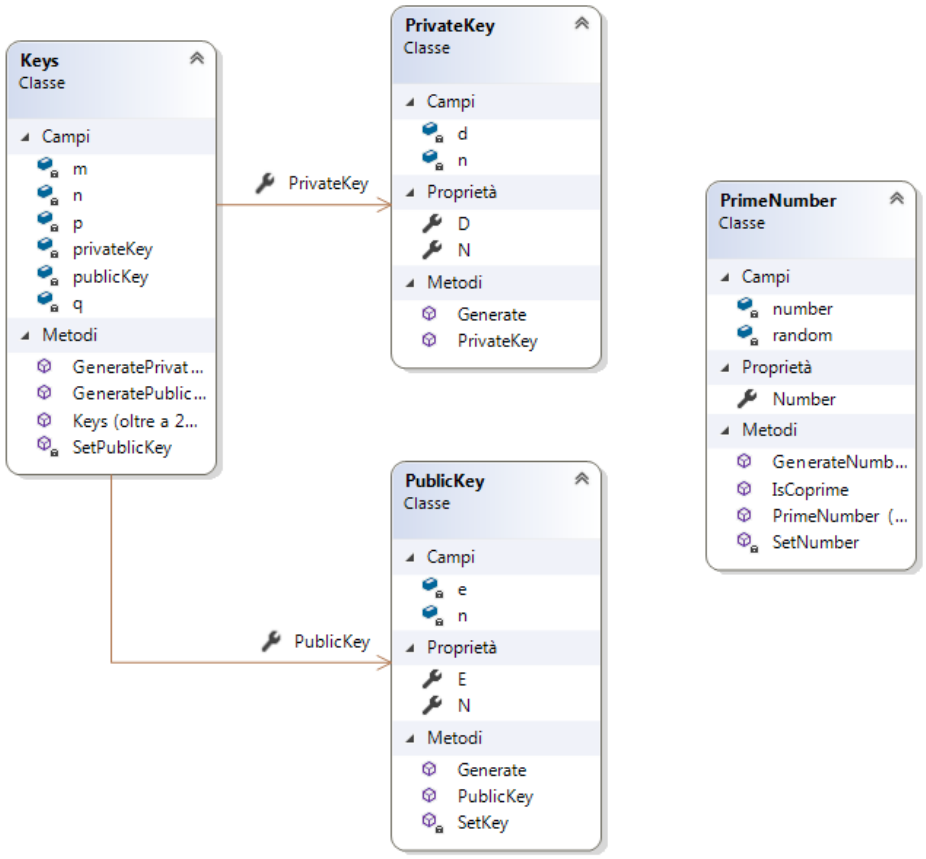
Formaggio Alberto 5E 6/3/19

# Relazione di sistemi e reti

Il testo dell'esercizio richiedeva la creazione di un programma che permettesse di cifrare e decifrare un testo usando l'algoritmo della chiave asimmetrica del protocollo RSA.
Il programma doveva essere realizzato nel linguaggio di programmazione C#.

Inizialmente si è creato, grazie allo strumento "visualizzazione diagramma classi" reso disponibile da Visual Studio, uno schema che rappresentasse le entità (e quindi le classi) presenti nel programma e i collegamenti tra di esse.
Il risultato prodotto è stato il seguente:




**<u>Analisi dello schema</u>**:

Si è partiti dalla realizzazione della classe **PrimeNumber** che come membri possiede il *numero* generato e un oggetto di tipo *Random* per la generazione casuale dei numeri. Il numero membro di questa classe deve essere primo. Troviamo poi due costruttori: uno che non accetta alcun parametro, mentre il secondo che accetta un

numero e un valore booleano. Il primo genera un valore casuale e verifica sia primo, mentre il secondo genera un numero casuale limitato da un valore massimo o prova a settare un valore passato a seconda del valore della variabile booleana.
I metodi SetNumber e GenerateNumber servono per effettuare le operazioni dei costruttori appena descritte.
Troviamo inoltre il metodo IsCoprime che controlla se l'istanza corrente è coprimo con un numero *m* passato.

La classe **PublicKey** contiene i valori che identificano la chiave pubblica, ovvero *n* ed *e*. Il primo è di tipo long, mentre il secondo è di tipo PrimeNumber. Sono poi presenti due metodi **Generate** e **SetKey** che, rispettivamente, generano e settano i valori di una chiave già esistente. Il primo verrà richiamato al momento della creazione della propria chiave pubblica, mentre il secondo per creare la chiave pubblica dell'altro utente (con n ed e quindi già conosciuti). Il costruttore della classe ha il compito di creare una chiave. È stato necessario inserire, oltre ai due numeri, un valore booleano che dica se l'operazione da eseguire è la generazione oppure il set della chiave. Si è fatto ciò dal momento che sia la coppia (n,m) che (n,e) è formata da valori di tipo long (quando *e* arriva dall'altro utente non è di tipo PrimeNumber, ma soltanto un numero) e quindi non era possibile eseguire l'overloading del costruttore.

La classe **PrivateKey** ha una funzione simile a quella precedentemente descritta, l'unica differenza sta nei valori (*n* e *d*) e nel fatto che la chiave Privata può essere solo generata e non settata, dal momento che non potrà mai verificarsi che si riceva da un altro utente la propria chiave privata, la quale sappiamo deve essere custodita segretamente da colui che la possiede.

Infine la classe **Keys** contiene come campi membro i numeri p e q (i quali sono di tipo **Prime Number**), i valori n ed m calcolati in base ad essi e un riferimento alla *chiave pubblica* e a quella *privata*. Nella classe sono presenti poi 3 costruttori: uno senza parametri che genera due chiavi in modo del tutto casuale, uno che invece crea una chiave conoscendo due numeri primi P e Q, già determinati, e infine un costruttore che accetta *n* ed *e* e crea un'istanza soltanto della chiave pubblica. I metodi GeneratePublicKey, GeneratePrivateKey e SetPublicKey servono per effettuare le operazioni dei costruttori precedentemente descritte. Si è deciso di evitare di inserire tutto dentro al costruttore per evitare inutile ridondanza e poiché

sarebbe potuto essere necessario richiamare tali metodi anche all'infuori della creazione delle chiavi.

È stata poi creata una libreria esterna denominata **ExtensionMethods** che contiene delle classi Helper (di cui non può essere creata nessuna istanza) contenenti metodi utili a fornire alcune funzionalità. È buona prassi, infatti, creare librerie esterne che possono essere inserite all'interno di qualsiasi programma qualora sia necessario richiamare i suoi metodi. Nel caso in cui la classe helper sia statica, è detta "Utility class".
Si è creata una classe Helper (PrimeNumberHelper) per fornire supporto al calcolo dei numeri primi.
Al suo interno è presente un metodo per il calcolo del MCD (Massimo Comun Divisore), un metodo per stabilire se un numero è primo o composto e un metodo che effettua l'esponenziazione modulare.

**<u>Scelte progettuali</u>**:

- Numeri primi: Si sono considerati due metodi per stabilire se un numero è primo o meno:
    - Metodo delle divisioni di prova (trial division): Si prende un numero n che bisogna stabilire se è primo o meno e un numero k divisore che parte da 2. Ad ogni iterazione si incrementa k di 1. Nel caso in cui risulti che n è divisibile per k si ritorna falso immediatamente, altrimenti si continua a controllare fino a quando k non diventa maggiore o uguale a n.
    - Metodo di Fermat: Dato un intero *n*, si sceglie un numero intero *a* coprimo di *n* si calcola $a^{n-1}$ modulo n. Se il risultato è diverso da 1, n è composto. Se è 1, allora n potrebbe essere primo. Un valore *a* per cui la relazione è sempre vera è detto "witness" (in inglese).
      Il numero *potrebbe* essere primo perché in realtà esistono dei numeri, detti pseudoprimi, che sono composti, ma che per certi valori di *a* il risultato sarà 1.
      I valori di *a* che applicati alla formula danno come risultato 1 ma che non sono witnesses possono essere al massimo la metà di tutti i numeri minori di n (dimostrato scientificamente). La probabilità, dopo T iterazioni, che non sia uscito nessun witness è pari a $1/2^t$. Quindi dopo

20 iterazioni, la probabilità di non aver trovato nessun witness è 1/1048576. Da qui si deduce che questo metodo probabilistico permette di stabilire se un numero è primo al 99,9%. Con un numero pari a 20 prove.

Tra i due metodi appena descritti si è scelto di applicare il secondo dal momento che con un numero di prove sempre uguale è possibile stabilire per qualsiasi numero, qualsiasi sia la sua dimensione, se esso è primo no. Questo permette di non porre alcun limite al momento della generazione del numero casuale. Infatti, con il primo metodo descritto, nel caso in cui il numero sia infinitamente grande, il programma rallenterebbe e potrebbe trascorrere fin troppo tempo per la generazione di un numero primo.
Il metodo di Fermat era stato scelto perché inizialmente si era pensato di adottare come tipo di dato numerico il BigInteger, il quale non ha alcun limite relativo alle dimensioni se non la memoria del computer che dovrà immagazzinarlo. Purtroppo però la classe Random fornita dal .NET Framework non permette la generazione di un numero che non sia di tipo Int32.

- Esponenziazione modulare: è l'algoritmo che permette di effettuare il calcolo $x^k mod n$. Viene applicato sia nel test di Fermat che nella cifratura e decifratura del testo da inviare. Deve essere eseguito perché la complessità di un'elevazione a potenza con base ed esponenti elevati può essere alta. Ad esempio $32^{77}$, numeri che sono anche piuttosto piccoli per il nostro programma, verrebbe un numero con più di 100 cifre in base 10 e che quindi per essere generato occuperebbe troppo tempo. Questo metodo permette quindi di diminuire i tempi di calcolo.
  Questo metodo però viene applicato soltanto nel test di Fermat perché nella fase di cifratura e decifratura viene applicato il metodo ModPow() della classe BigInteger. È stato necessario fare ciò perché i valori in fase di cifratura e decifratura sono troppo elevati per essere trattati come Int32.

- Generazione chiave privata: il metodo suggerito durante le ore di lezione per la generazione dell'esponente privato d sembra funzionare solo con valori piccoli: applicando la formula d=[(k*m)+1]/e e poi la formula (d*e)%m==1, con valori piccoli il ciclo terminava velocemente, mentre con valori grandi, risultava un ciclo lungo che talvolta mandava in eccezione il programma. Dopo svariati tentativi e prove e non riuscendo a risolvere il problema sollevato, si è

deciso di applicare l'algoritmo di Euclide esteso per la generazione dell'esponente privato.
Questo metodo inoltre può risultare più efficiente rispetto al precedente perché con numeri relativamente grandi un metodo eseguito "per tentativi" può risultare piuttosto dispendioso in termini di tempo e risorse.

Per l'implementazione del codice appena realizzato, si è deciso di creare un programma di chat in tempo reale che all'invio e alla ricezione, cifra e decifra il testo del messaggio.

Per la realizzazione del programma è stato necessario documentarsi in Internet sul funzionamento dei Socket (Indirizzo IP e Porta) e delle classi messe a disposizione dal .NET Framework che permettessero la loro implementazione per realizzare la comunicazione, la quale deve avvenire in modo asincrono: il mittente infatti può spedire il messaggio e continua ad effettuare le proprie operazioni senza dover attendere la risposta da parte dell'altro utente.

Uscendo un dal contesto realizzativo del programma, una trasmissione è sincrona quando i clock sono allineati tra mittente e ricevente: il messaggio viene trasmesso solo in istanti di tempo predefiniti. Nelle trasimissioni asincrone invece i tempi di trasmissione dei dati sono casuali e i clock di mettente e ricevente sono indipendenti l'uno dall'altro.

Sempre per quanto concerne i fondamenti teorici per la realizzazione di questa parte di programma, si è dovuto comprendere come avviene esattamente la comunicazione tra client e server: nel client è presente un TcpClient che si connette attraverso uno Stream, il quale controlla il flusso di informazioni, il quale a sua volta invia i dati al socket che li spedisce fuori da una porta. Questi dati poi entrano in una porta (quella del server), escono dal socket, passano per lo Stream e arrivano al TcpListener (che è quella parte del server in ascolto passivo in attesa che arrivino informazioni).

All'interno del programma sono stati inseriti 2 backGroundWorker (uno per la lettura e uno per la scrittura) che rimangono in continuo ascolto di avvenimenti all'interno del programma e che si attivano quando qualcosa accade.

Poiché l'istanza del programma deve essere la stessa, bisogna prevedere sia il caso in cui il programma si comporta da Server, sia quando si deve comportare da Client.

Ciò è reso possibile con 2 bottoni: btnStart che avvia l'istanza del server e btnConnect che invece permette di collegarsi ad un'istanza del server già attiva.

Al momento dell'avvio della comunicazione vengono subito scambiate le chiavi pubbliche tra server e client. Quelle private devono rimanere segrete.
Terminato anche questo passaggio, possono essere scambiati messaggi di testo tra server e client liberamente, senza necessità che uno dei due risponda al messaggio inviato dall'altro.

Durante la comunicazione l'utente può generare delle nuove chiavi casuali, oppure inserire lui stesso dei valori di p e q sulla base dei quali verranno generate le chiavi private e pubbliche.
Bisogna tenere in considerazione però che il prodotto tra p e q non può essere inferiore a 256 dal momento che questo non permetterebbe una visualizzazione di tutti i valori della tabella ASCII esteso(a 8 bit): il problema si genererebbe in fase di invio o ricezione quando si prende il numero elevato e si esegue il modulo con *n* (che è, appunto, il prodotto tra p e q).
Si poteva anche limitare il programma all'utilizzo dei soli caratteri della tabella ASCII normale a 7 bit, ma questo avrebbe impedito l'utilizzo di caratteri accentati, i quali sono invece frequentemente utilizzati nella comunicazione.

Si è deciso che lo scambio di informazioni debba avvenire con un file .JSON, applicando così trasversalmente le nozioni acquisite anche in TPSI. Per poter però serializzare un oggetto in formato JSON è necessario creare una classe (ElementToSend) che conterrà il tipo di dato inviato (key o text) sulla base del quale si interpreteranno i valori inseriti all'interno di *values*.

Per la serializzazione si è dovuto importare la libreria Newtonsoft.Json.

Questa parte di realizzazione del programma si è rivelata essere molto più complicata della prima parte (quella dello sviluppo dell'algoritmo RSA vero e proprio) anche perché questo argomento non è mai stato affrontato nel programma scolastico di informatica, anche se comunque possedevo già delle nozioni molto generiche relative alla comunicazione asincrona grazie al periodo di ASL svolto (tuttavia, in quel caso si era trattato il tema in linguaggio Javascript che, essendo linguaggio di programmazione Web, si presta maggiormente a questo tipo di operazioni).

In seguito è stata integrata al programma anche la parte relativa alla gestione degli utenti. Per farla si sono creati due nuovi Form: uno per la creazione di un nuovo Utente, mentre un altro per il login di un utente già esistente.
Al momento della creazione dell'account si controllano il valore inserito nel campo password: esso infatti deve contenere almeno una lettera maiuscola, minuscola e un numero ed essere lunga almeno 8 caratteri. Se questi requisiti sono soddisfatti, allora si procede con l'inserimento dei dati nel Database. Come chiave è utilizzato il campo Username, quindi non è possibile inserire due utenti con lo stesso nome e nel caso in cui si provi a far ciò verrà ritornato un messaggio di errore.

Per quanto riguarda il login invece, si fanno inserire all'utente username e password e, al click sul bottone di login, verrà fatta un'interrogazione al database che ritornerà i valori delle chiavi inserite al momento della registrazione (quindi il valore *n*, l'esponente pubblico *e* e l'esponente privato *d*). Tali valori verranno inseriti all'interno del programma e inviati all'altra istanza del programma collegata per permettere di aggiornare le sue chiavi.

Il database utilizzato è un dataBase Access, il quale però presenta una problematica: può lavorare solo in locale. Quindi, nel caso in cui il programma venga copiato su un altro computer, le modifiche apportate al dataBase di un programma non si rifletteranno anche sull'altra istanza: una possibile soluzione a questo problema è l'utilizzo di DataBase online. Purtroppo, per effettuare le operazioni necessarie ai fini del programma, è necessaria la registrazione con anche i dati di fatturazione. Nel caso in cui ciò fosse possibile, le operazioni effettuate da un Pc sarebbero visualizzabili a tutti gli utenti che faranno uso del programma.