Corso di Programmazione ad Oggetti – a.a. 2013/2014

Corso di laurea in Ingegneria e Scienze Informatiche – Università di Bologna

Relazione relativa all'ingegnerizzazione di ISI-Crypt: un'applicazione di crittografia e steganografia

Componenti del gruppo: Eugenio Severi e Filippo Vimini

1 Analisi del problema

È stata realizzata un'applicazione per la gestione sicura dei dati personali combinando crittografia simmetrica e asimmetrica, oltre ad alcune tecniche di offuscamento dei dati. In particolare offre quattro funzioni principali:

- Crittografia di un file, eventualmente in forma compressa, e cancellazione sicura del file di origine una volta prodotto il file cifrato. L'integrità dei dati è garantita da algoritmi di hashing.
- Steganografia di informazioni all'interno di un'immagine.
- Un password manager che consente di salvare in un database cifrato i dati sensibili dell'utente.
- Scambio file attraverso la rete tra due istanze del programma, in forma cifrata.
- Selezione di temi (schemi di colori per i componenti grafici) predefiniti per l'applicazione.

Nella realizzazione della GUI è stato tenuto presente l'obbiettivo di rendere l'utilizzo del programma relativamente semplice anche per l'utente medio. Per questo motivo il programma si avvia su una start screen dalla quale l'utente può scegliere la funzionalità desiderata. Inoltre alcune operazioni compiute dal programma sono nascoste all'utilizzatore, che attraverso l'interfaccia grafica interagisce solo con gli elementi maggiormente ad alto livello dell'applicazione.

I dati inseriti dall'utilizzatore sono salvati su file nella cartella "home" dell'utente e ricaricati al successivo avvio dell'applicazione.

2 Progettazione architetturale

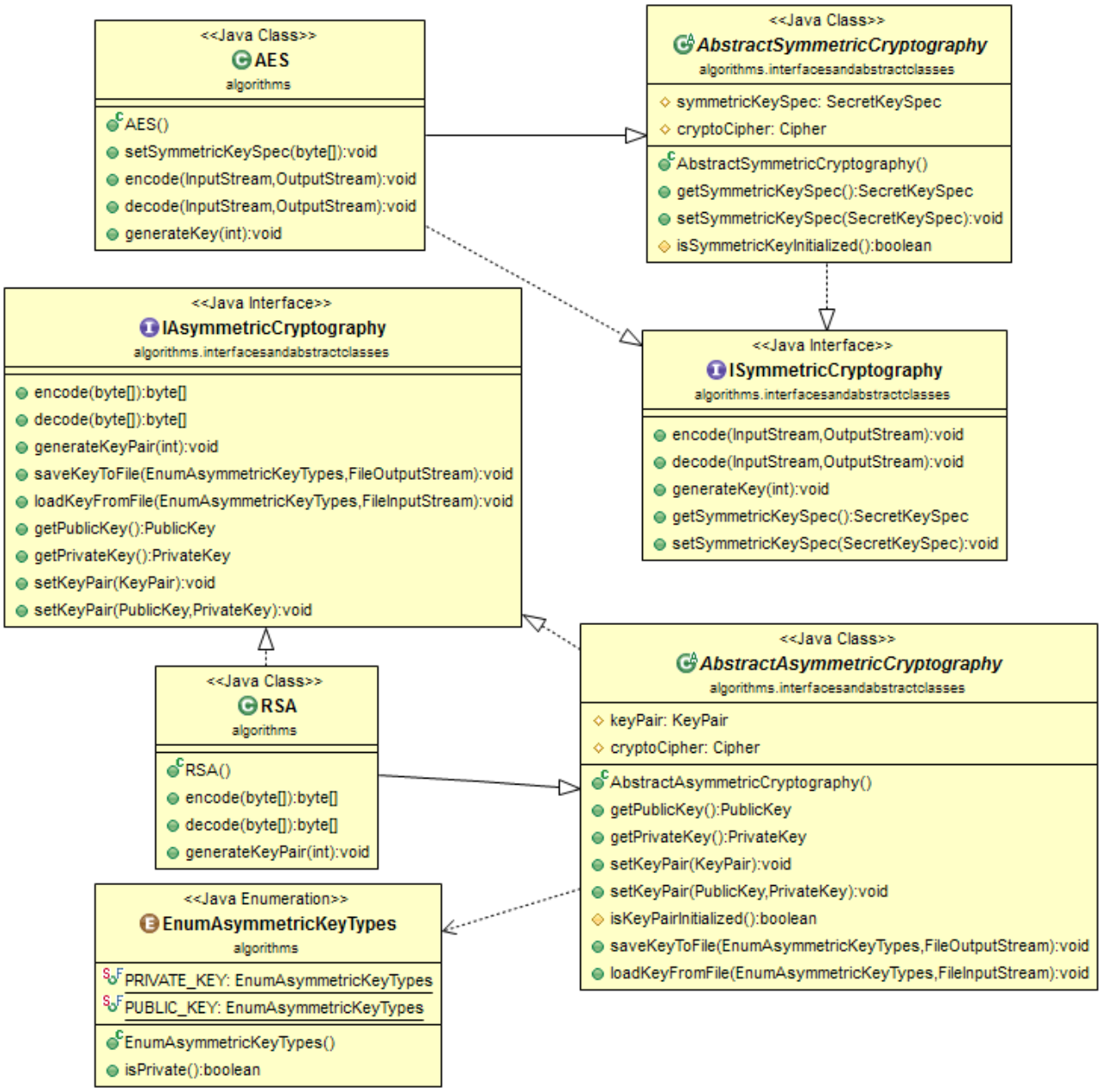
In questa fase è stato definito lo schema delle classi e dei package allo scopo di raggruppare entità correlate e favorire il riutilizzo del codice. Per questo motivo sono stati realizzati sistemi di interfacce e classi astratte.

È stato largamente utilizzato il pattern Singleton per l'implementazione degli algoritmi di hashing, compressione e wiping, a causa della natura a singola istanza di quelle classi. L'interfaccia grafica è stata realizzata secondo il pattern MVC (Model-View-Controller), allo scopo di separare gli aspetti puramente grafici dalla logica dell'applicazione e da quelli di modellazione dei dati, favorendo il riuso del codice in scenari differenti.

Per ognuna delle quattro funzioni principali dell'applicazione, essendo separate e distinte tra loro, è stata realizzata un'interfaccia grafica autonoma, alla quale si accede a partire da una schermata iniziale. Per questo motivo sono state realizzate classi MVC distinte per ognuna di esse.

Segue ora la presentazione di alcuni degli aspetti più rilevanti relativi all'architettura del sistema. In primo luogo saranno descritti gli algoritmi implementati, e successivamente le varie funzioni del programma.

Diagramma UML delle classi relative agli algoritmi crittografici implementati nell'applicazione.

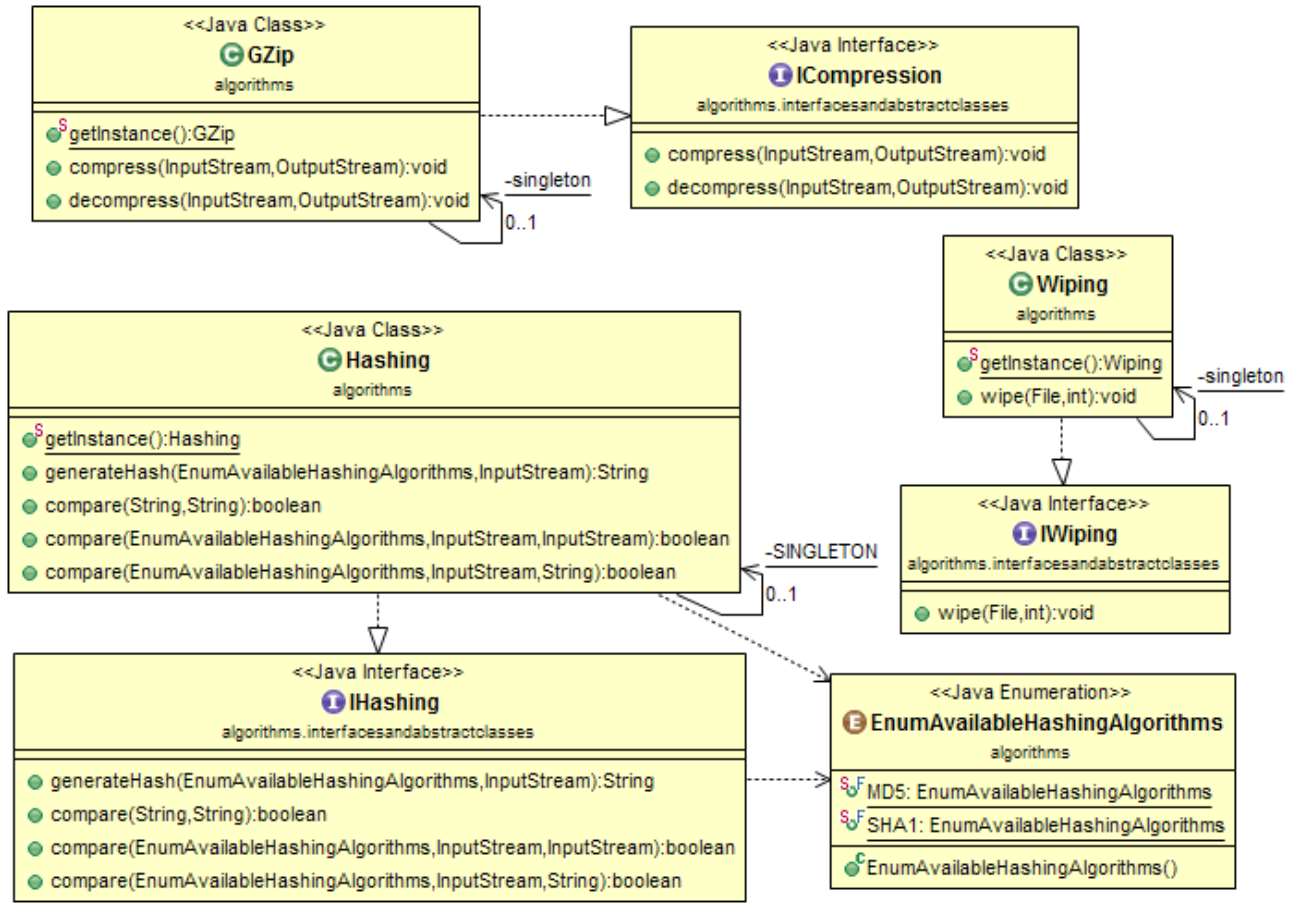


Descrizione degli aspetti principali:

- Gli algoritmi crittografici sono divisi in due categorie: a chiave simmetrica e asimmetrica. Due interfacce distinte (**ISymmetricCryptography** e **IAsymmetricCryptography**) rappresentano tale distinzione, ed esplicitano i metodi comuni che devono essere implementati da ciascun algoritmo delle specifiche categorie. È rilevante che gli algoritmi simmetrici lavorino a livello di stream, mentre quelli asimmetrici a livello di array di byte: questa scelta è stata dettata dalle minori prestazioni degli algoritmi asimmetrici per cifrare grandi quantità di dati. L'idea degli sviluppatori è di utilizzare sempre algoritmi di crittografia simmetrica per cifrare i dati dell'utente, e ricorrere a quelli asimmetrici solo per aumentare la sicurezza di quelli simmetrici, cifrandone la chiave.
- Per ognuna delle due categorie di algoritmi di cifratura è stata realizzata una classe astratta (**AbstractSymmetricCryptography** e **AbstractAsymmetricCryptography**) nella quale sono stati implementati alcuni metodi comuni a tutti gli algoritmi della categoria: le classi che implementeranno gli algoritmi veri e propri estenderanno da queste.

- Al momento è stato implementato un solo algoritmo per categoria: **AES** per la crittografia simmetrica; **RSA** per quella asimmetrica. In futuro si prevede di aggiungerne altri.

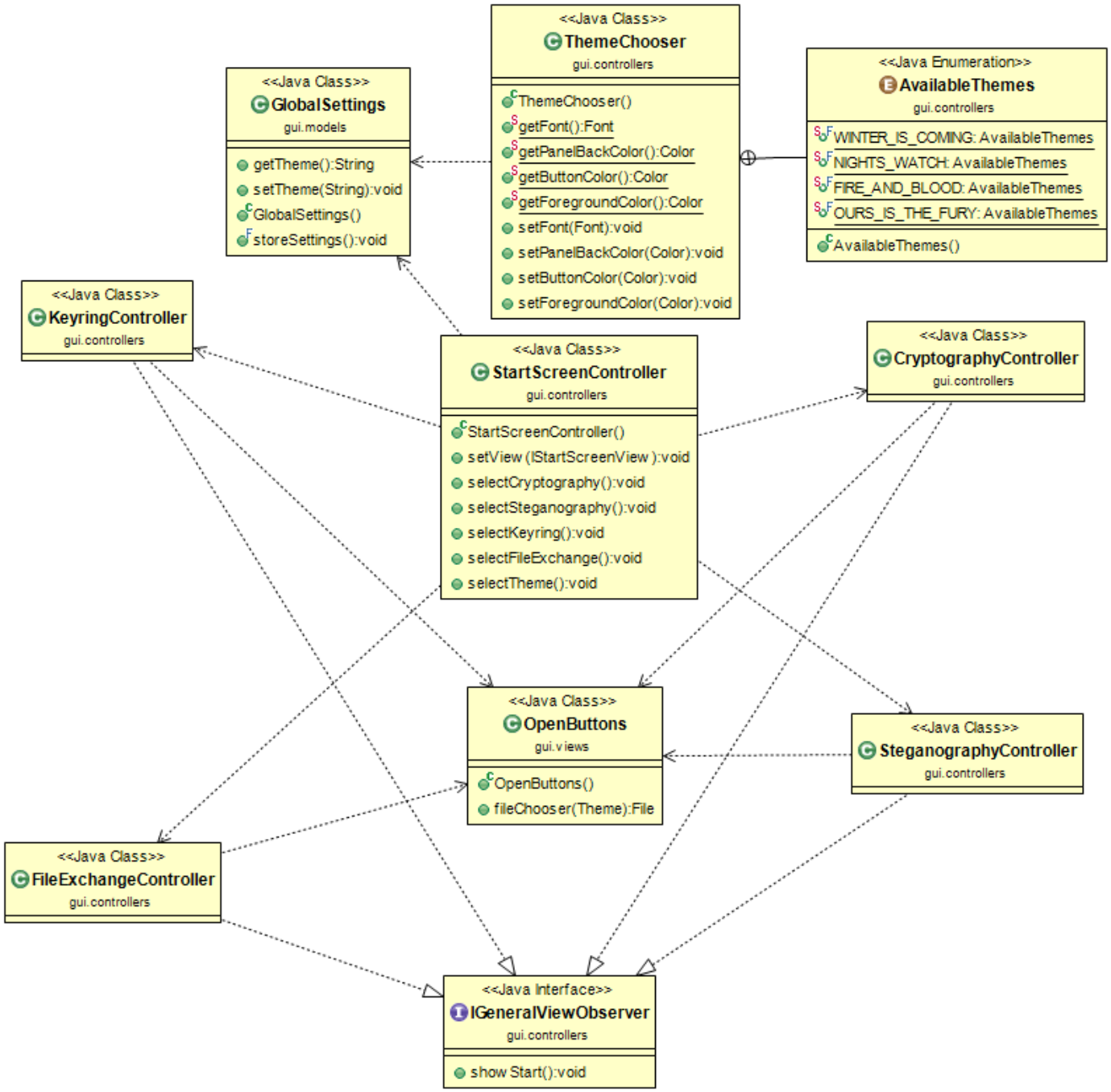
Diagramma UML delle classi relative agli algoritmi realizzati con pattern singleton.



Descrizione degli aspetti principali:

- Queste classi e relative interfacce sono state create con il pattern singleton, in quanto è ragionevole pensare che non sia necessaria più di un'istanza di ogni ognuna contemporaneamente. Inoltre nessuno di questi algoritmi necessita di mantenere proprietà sullo stato dell'istanza.
- Per gli algoritmi di compressione è stato al momento implementato un solo algoritmo (**GZip**). Tutti gli algoritmi di compressione forniscono i metodi *compress()* e *decompress()*, definiti nell'interfaccia **ICompression** che consento rispettivamente di comprimere e di decomprimere uno stream.
- Anche per gli algoritmi di wiping (**Wiping**) è al momento stato implementato un solo algoritmo, basato su un numero di sovrascrittura del file selezionato con dati generati casualmente stabilito dall'utente. Il metodo *wipe()* è definito nell'interfaccia **IWiping**, implementata da **Wiping**.
- Gli algoritmi di hashing (**Hashing**) richiedono uno stream del quale calcolare un checksum. L'algoritmo di hashing può essere scelto dall'utente tra quelli disponibili in **EnumAvailableHashingAlgorithms**. Al momento sono presenti MD5 e SHA-1. L'interfaccia **IHashing**, implementata da **Hashing**, definisce metodi sia per generare l'hash di uno stream, sia per comparare tra loro stream e checksum generati precedentemente.

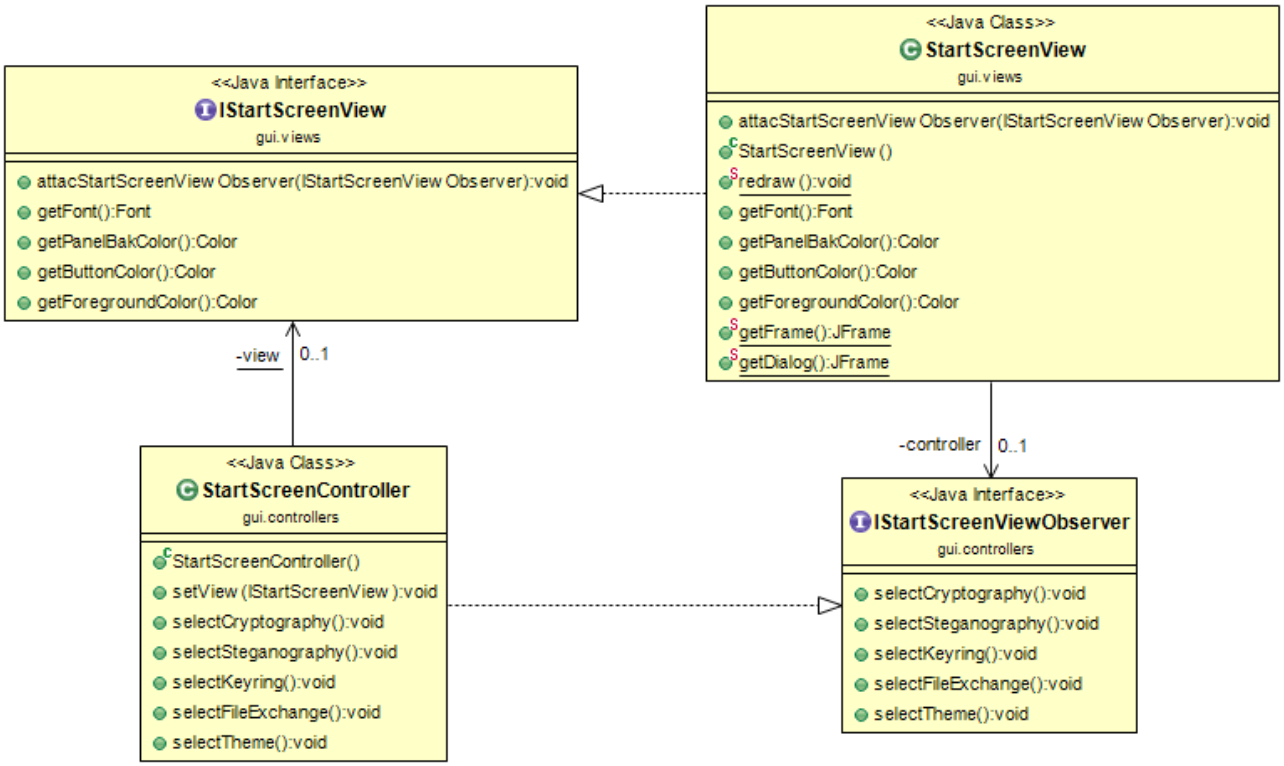
Diagramma UML delle classi relative alla parte dei controller dell'applicazione.



Descrizione degli aspetti principali:

- L'applicazione si avvia su una schermata principale (**StartScreenController**) dalla quale è possibile selezionare una delle funzioni del programma (**CryptographyController**, **SteganographyController**, **KeyringController**, **FileExchangeContoller**). Da ognuna di queste funzioni è possibile ritornare alla start screen senza necessità di chiuderla grazie al metodo *showStart()* dell'interfaccia **IGeneralViewObserver**, implementata da tutti i controller delle funzioni.
- Dalla schermata iniziale è possibile selezionare un tema visivo per l'applicazione tra uno di quelli proposti in **FileTypes**. La classe **ThemeChooser** si occupa di impostare le proprietà dei temi, mentre **GlobalSettings** ad ogni avvio del programma legge l'impostazione precedentemente salvata.
- La classe **OpenButtons** è una classe di utilità che fornisce un metodo di utilità utilizzato dai controller per richiedere all'utente di selezionare file e cartelle da aprire.

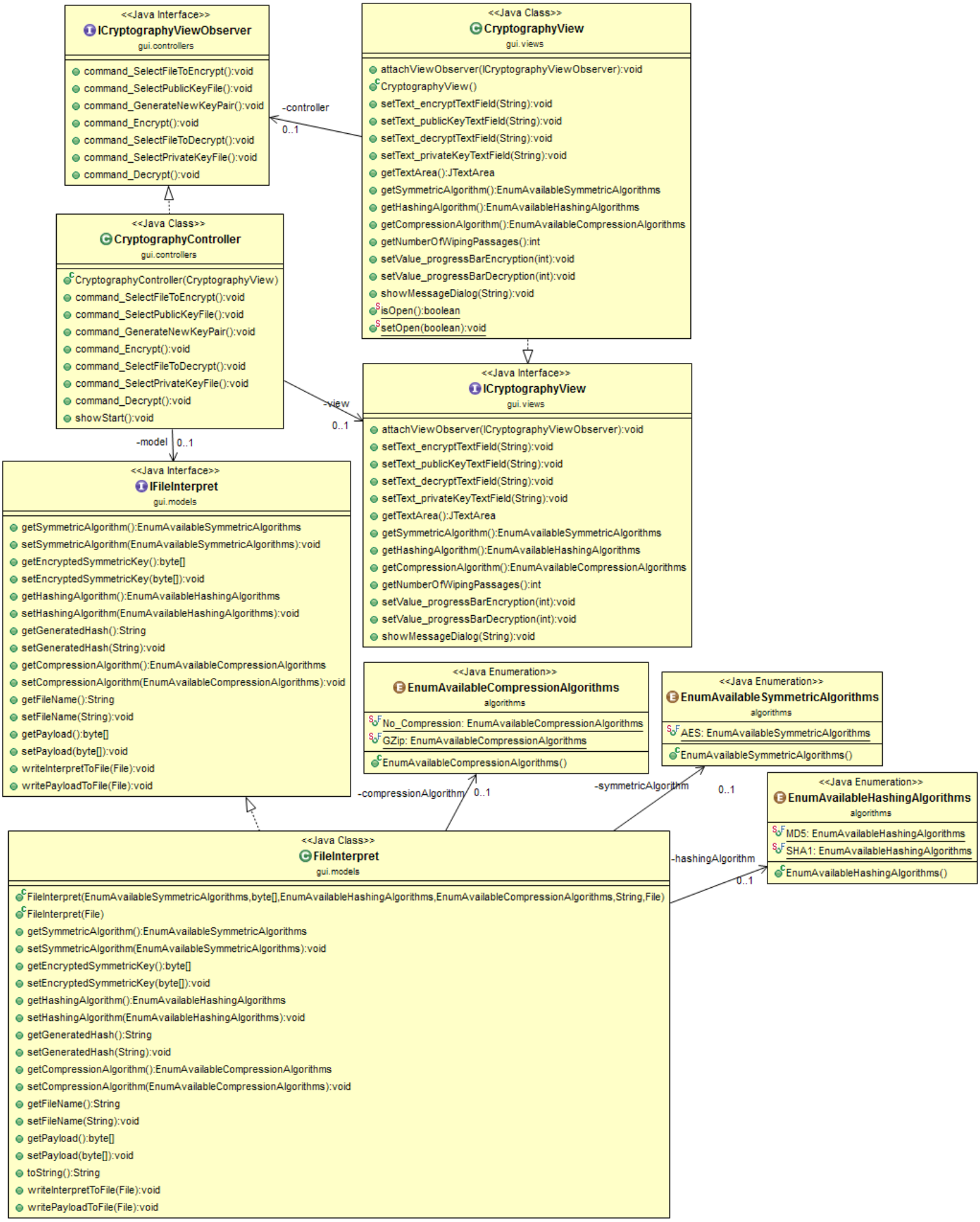
Diagramma UML delle classi relative alla start screen.



Descrizione degli aspetti principali:

- La start screen è la schermata di avvio dell'applicazione. Da qui è possibile eseguire contemporaneamente più GUI (ma non due dello stesso tipo).

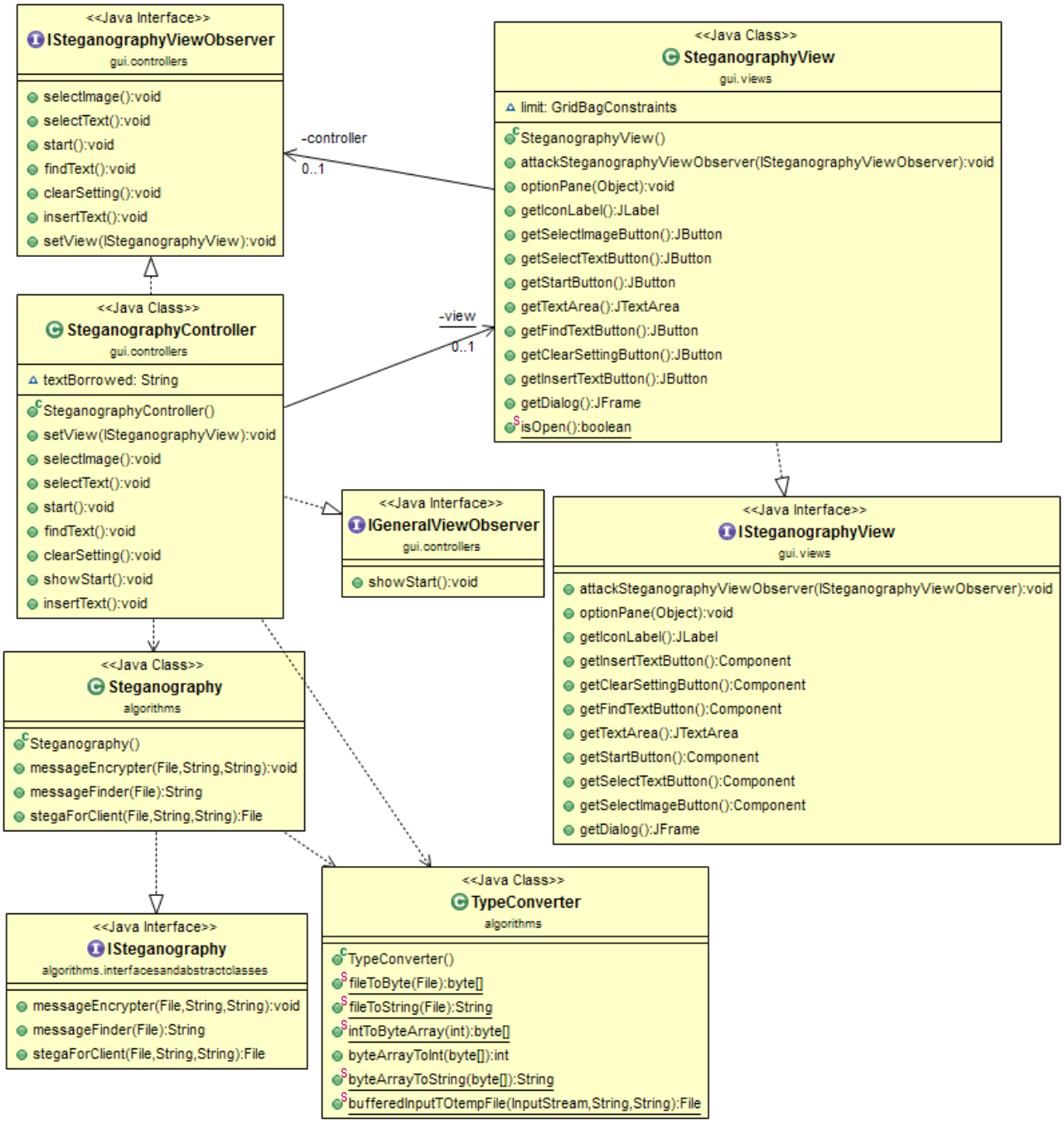
Diagramma UML delle classi relative alla funzione di crittografia.



Descrizione degli aspetti principali:

- Attraverso l'interfaccia grafica (**CryptographyView**) l'utente seleziona le impostazioni sia per crittografare, sia per decrittografare i file.
- Le impostazioni selezionabili dall'utente nelle ComboBox sono definite in specifiche Enum create per lo scopo (**EnumAvailableSymmetricAlgorithms**, **EnumAvailableHashingAlgorithms**, **EnumAvailableCompressionAlgorithms**), ad eccezione delle impostazioni relative al numero di sovrascritture per il wiping del file di origine, che sono incapsulate nella view.
- La classe **FileInterpret** costituisce un wrapper per i file cifrati dall'utente, aggiungendo ad essi alcuni metadati.

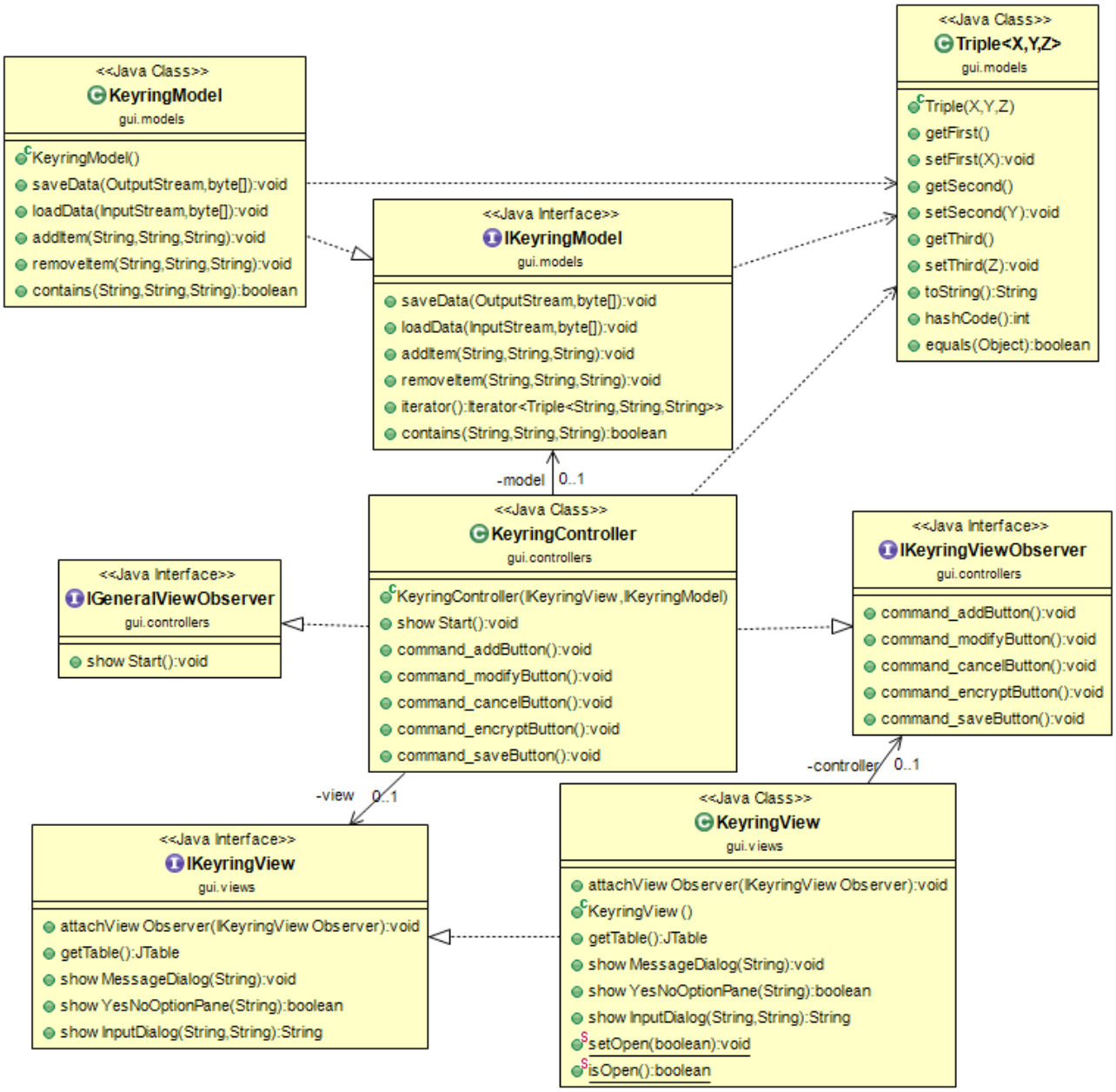
Diagramma UML delle classi relative alla funzione di steganografia.



Descrizione degli aspetti principali:

- Attraverso l'interfaccia grafica (**SteganographyView**) l'utente può selezionare una nuova immagine all'interno della quale nascondere dei dati, inserire tali dati in un'immagine caricata, salvare un'immagine alterata, leggere dati da un'immagine modificata precedentemente creata.
- La funzione di steganografia non prevede un modello, in quanto tutte le informazioni necessarie sono generate a runtime e non sono presenti impostazioni da salvare per una consultazione futura.

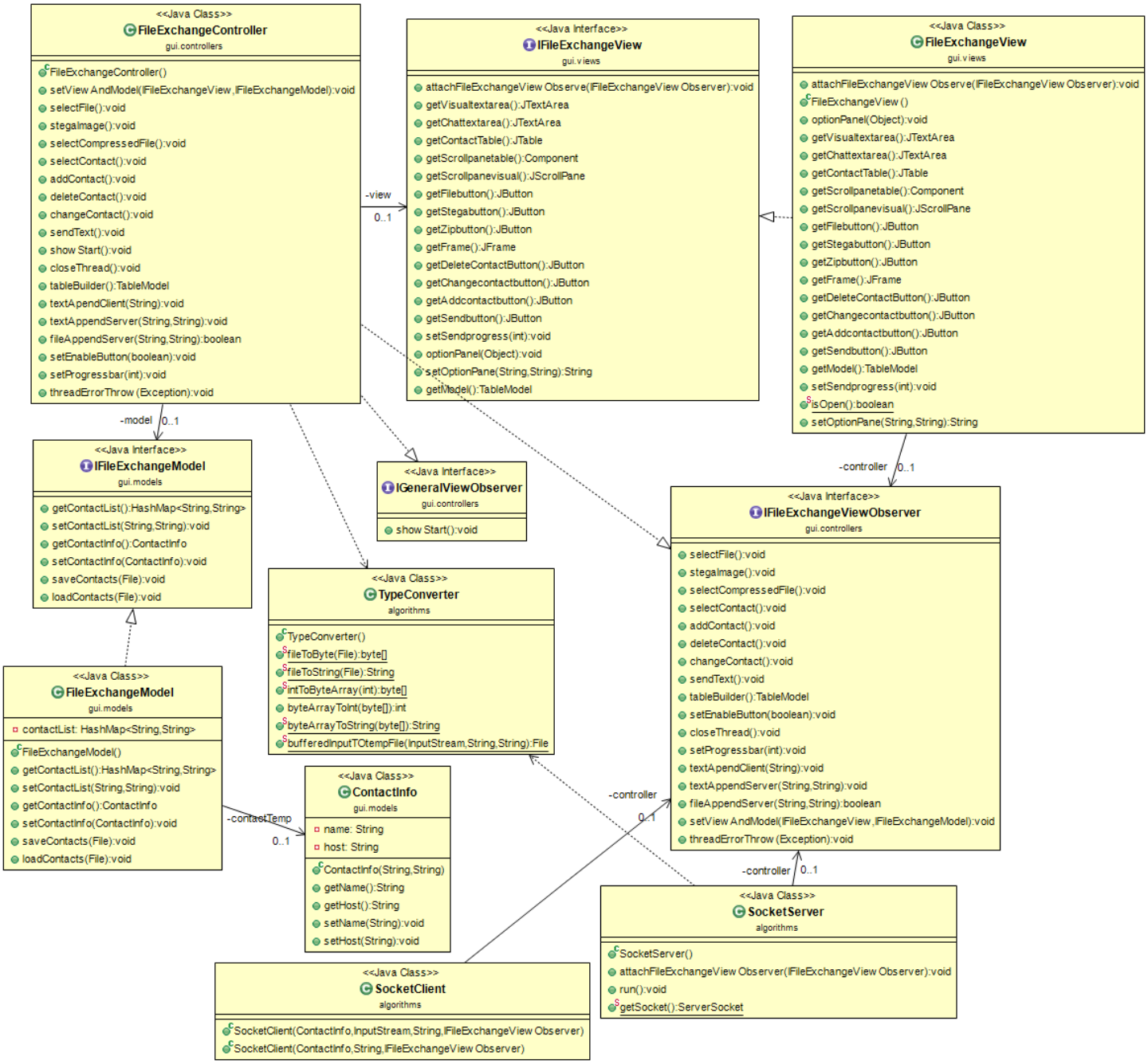
Diagramma UML delle classi relative alla funzione di password manager.



Descrizione degli aspetti principali:

- Attraverso l'interfaccia grafica (**KeyringView**) l'utente può aggiungere ad una tabella nuovi record contenenti il nome di un sito e relativi username e password. Può inoltre modificare o eliminare record precedentemente creati. Dalla stessa schermata è possibile caricare o generare una chiave di cifratura AES con la quale i dati saranno salvati su disco in forma cifrata.
- Il modello (**KeyringModel**) si occupa di caricare e salvare i database creati in forma crittografata, sotto forma di Set di **Triple**.

Diagramma UML delle classi relative alla funzione di scambio file sicuro.



Descrizione degli aspetti principali:

- Attraverso l'interfaccia grafica (**FileExchangeView**) l'utente può aggiungere un nuovo contatto, eliminarne uno esistente, o selezionarne uno tra quelli disponibili per avviare una sessione di chat.
- All'avvio della funzione di scambio file viene automaticamente avviato un server (**SocketServer**) che resta in ascolto di connessioni in arrivo da parte di un client (**SocketClient**).
- Una volta stabilita la connessione, client e server possono inviarsi a vicenda messaggi di testo o file. La sicurezza dei dati trasferiti è garantita dagli algoritmi di cifratura AES e RSA combinati, in maniera trasparente all'utente: RSA viene utilizzato per negoziare una chiave AES simmetrica per cifrare i dati veri e propri. Ogni trasferimento utilizza una chiave diversa.
- I contatti sono salvati e caricati dal modello (**FileExchangeModel**) sotto forma di Map di **ContactInfo**.

3 Organizzazione in package

Viene ora effettuata un'analisi dell'organizzazione in package dell'applicazione.

- **algorithms:** contiene le implementazioni dei vari algoritmi di cifratura simmetrica e asimmetrica, di hashing, di compressione, di wiping e di steganografia. Contiene inoltre la parte maggiormente a basso livello relativa alla funzione di scambio file attraverso la rete. Viene ora effettuata una breve descrizione dei sorgenti contenuti del package:
    - AES.java - Realizzazione dell'algoritmo di crittografia simmetrica AES. Estende la classe astratta e implementa l'interfaccia di crittografia simmetrica.
    - GZip.java – Realizzazione dell'algoritmo di compressione GZip. Implementa l'interfaccia di compressione.
    - Hashing.java – Realizzazione di alcuni algoritmi di hashing. Implementa la relativa interfaccia.
    - RSA.java - Realizzazione dell'algoritmo di crittografia a chiave pubblica RSA. Estende la classe astratta e implementa l'interfaccia di crittografia asimmetrica.
    - Wiping.java – Implementazione di un algoritmo di wiping basato su sovrascritture multiple. Implementa la reativa interfaccia.
    - Steganography.java – Implementazione di un algoritmo di steganografia LSB (Least Significant Bit).
    - SocketClient.java – Implementazione della parte client a basso livello della funzionalità di scambio file sicuro.
    - SocketServer.java - Implementazione della parte server a basso livello della funzionalità di scambio file sicuro.
    - EnumAsymmetricKeyTypes.java – Definizione delle tipologie di chiave in un algoritmo di cifratura a chiave simmetrica.
    - EnumAvailableCompressionAlgorithms.java – Definizione degli algoritmi di compressione attualmente disponibili nel programma.
    - EnumAvailableHashingAlgorithms.java – Definizione degli algoritmi di hashing attualmente disponibili nel programma.
    - EnumAvailableSymmetricAlgorithms.java – Definizione degli algoritmi di crittografia simmetrica attualmente disponibili nel programma.
    - ErrorMessages.java – Definizione di alcuni messaggi di errore utilizzati dal programma.
    - TypeConverter.java – Classe di utilità che fornisce metodi di conversione tra diversi tipi di dato e file.
- **algorithms.interfacesandabstractclasses:** contiene le interfacce e le classi astratte relative alle classi del package *algorithms*. Viene ora effettuata una breve descrizione dei sorgenti contenuti nel package:
    - ICompression.java – Interfaccia per gli algoritmi di compressione.
    - IHashing.java – Interfaccia per gli algoritmi di hashing
    - IAsymmetricCryptography.java – Interfaccia per gli algoritmi di crittografia asimmetrica.
    - ISymmetricCryptography.java – Interfaccia per gli algoritmi di crittografia simmetrica.
    - ISteganography.java – Interfaccia per la classe Steganography.
    - IWiping.java – Interfaccia per gli algorimi di wiping.
    - AbstractAsymmetricCryptography.java – Classe astratta che implementa metodi comuni agli algoritmi di crittografia asimmetrica.
    - AbstractSymmetricCryptography.java - Classe astratta che implementa metodi comuni agli algoritmi di crittografia simmetrica.

- **gui.controllers:** contiene i controller e le relative interfacce dell'applicazione. Viene ora effettuata una breve descrizione dei sorgenti contenuti nel package:
    - CryptographyController.java – Implementazione del controller per la GUI di crittografia.
    - ICryptographyViewObserver.java – Interfaccia implementata dal controller di crittografia.
    - FileExchangeController.java - Implementazione del controller per la GUI di scambio file sicuro.
    - IFileExchangeViewObserver - Interfaccia implementata dal controller di scambio file sicuro.
    - KeyringController.java - Implementazione del controller per la GUI del password manager.
    - IKeyringViewObserver.java - Interfaccia implementata dal controller del password manager.
    - StartScreenController.java - - Implementazione del controller per la GUI della start screen.
    - IStartScreenViewObserver.java - Interfaccia implementata dal controller della start screen.
    - SteganographyController.java - Implementazione del controller per la GUI di steganografia.
    - ISteganographyViewObserver.java - Interfaccia implementata dal controller di steganografia.
    - IGeneralViewObserver.java – Interfaccia implementata da tutti i controller delle gui secondarie. Consente di tornare alla start screen.
- **gui.models:** contiene il codice relativo al modello dell'applicazione. Viene ora effettuata una breve descrizione dei sorgenti contenuti nel package:
    - ContactInfo.java – Rappresenta la struttura di un contatto (coppia host-alias) per la GUI di scambio file.
    - FileExchangeModel.java – Memorizza l'insieme dei contatti per la GUI di scambio file.
    - FileInterpret.java – Costituisce un wrapper per i file crittografati tramite l'apposita GUI.
    - IFileInterpret.java – Interfaccia di FileInterpret.
    - CorruptedDataException.java – Eccezione che si verifica quando il checksum calcolato su alcuni dati non corrisponde a quello calcolato precedentemente.
    - GlobalSettings.java – Memorizza su disco il file delle impostazioni globali (relative a tutte le view).
    - ThemeChooser.java – Gestisce i temi dell'applicazione.
    - Triple.java – Classe che memorizza triple di valori generici. È utilizzata da KeyringModel.
    - KeyringModel.java – Memorizza l'insieme dei record inseriti dall'utente nel password manager.
    - IKeyringModel.java – Interfaccia di KeyringModel.
- **gui.views:** contiene il codice relativo alle view dell'applicazione. Viene ora effettuata una breve descrizione dei sorgenti contenuti nel package:
    - OpenButtons.java – Classe che incapsula un JFileChooser per selezionare diversi tipi di file.
    - AbstractGuiMethodSetter.java – Classe astratta contenente metodi per velocizzare la stesura di codice nelle view e funzioni per il ridimensionamento delle immagini.
    - CryptographyView.java – Realizzazione della view relativa alla funzione di crittografia/decrittografia.
    - ICryptographyView.java – Interfaccia relativa a CryptographyView.
    - FileExchangeView.java – Realizzazione della view relativa alla funzione di scambio file sicuro.
    - IFileExchangeView.java - Interfaccia relativa a FileExchangeView.
    - KeyringView.java - Realizzazione della view relativa alla funzione di password manager.
    - IKeyringView.java – Interfaccia relativa a KeyringView.
    - StartScreenView.java - Realizzazione della view relativa alla start screen.
    - IStartScreenView.java - Interfaccia relativa a StartScreenView.
- **test:** contiene alcune classi utilizzate durante la fase di test delle varie funzionalità e il main.

4 Suddivisione del lavoro

Nella realizzazione del progetto il lavoro è stato diviso tra i componenti del gruppo nel seguente modo:

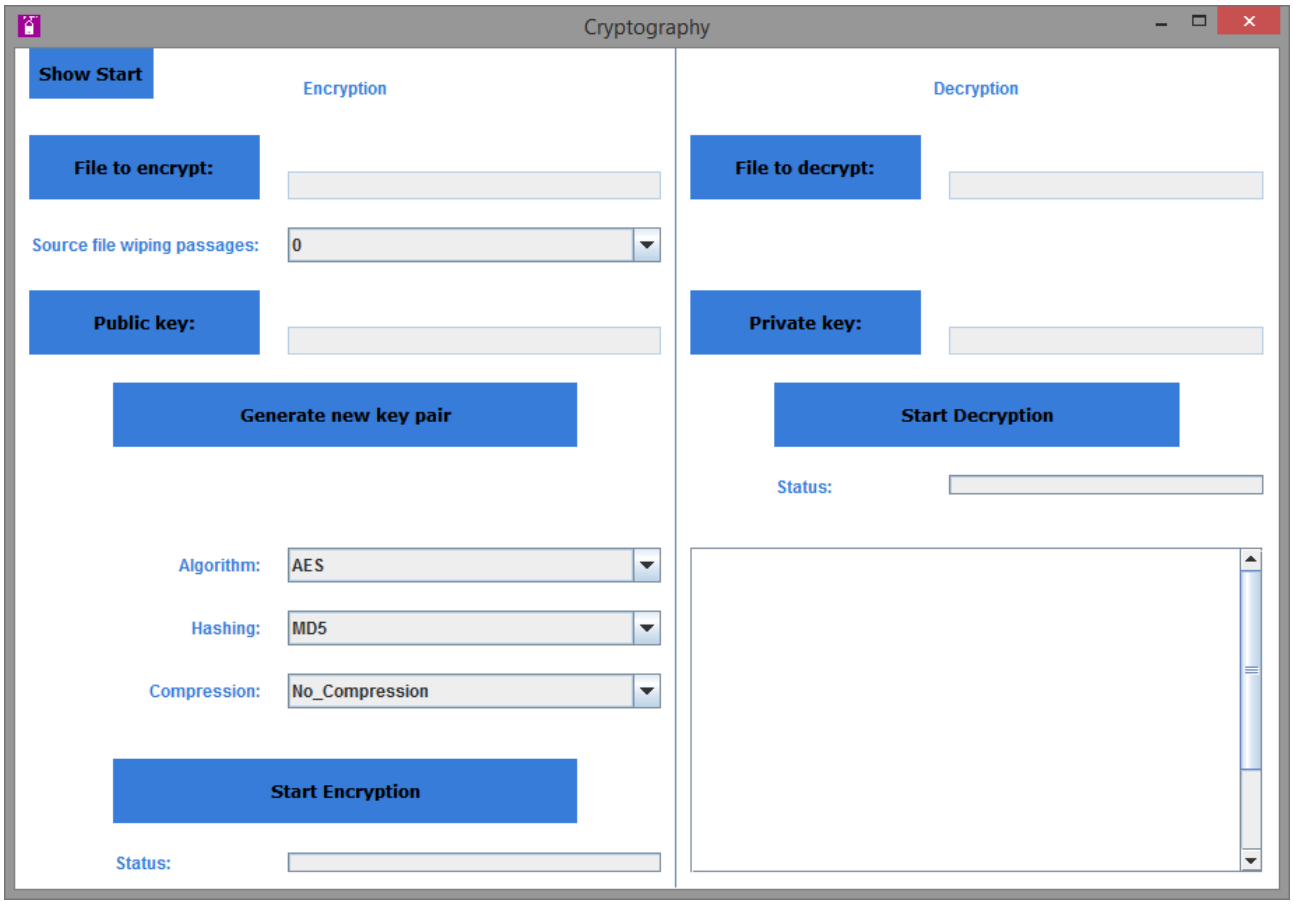
- Eugenio Severi si è occupato dello sviluppo degli algoritmi di crittografia simmetrica e asimmetrica e delle relative classi astratte e interfacce, degli algoritmi di compressione e wiping, della GUI di crittografia e di quella del password manager.
- Filippo Vimini si è occupato dello sviluppo degli algoritmi di hashing, di steganografia; ha sviluppato le classi per le comunicazioni client-server per la funzione di scambio file sicuro. Ha realizzato inoltre le GUI relative alle funzioni di steganografia e scambio file.

Alcune parti del progetto sono state sviluppate in sinergia:

- Realizzazione dello stile generale delle view (comune in tutta l'applicazione).
- Realizzazione della funzionalità di salvataggio e caricamento dei temi visivi.
- Pulizia finale del codice.

5.1 Progettazione di dettaglio: parte di Eugenio Severi

Funzione di crittografia



L'idea è quella di fornire all'utente un modo sicuro per cifrare i propri dati senza dover ricordare numerose password complesse. Dal momento che qualunque password scelta dall'utente difficilmente sarà realmente casuale e composta da tutti i 256 caratteri ASCII, la generazione di chiavi è interamente gestita dall'applicazione. L'unico aspetto legato alle chiavi che l'utente è tenuto a conoscere è la coppia di chiavi **RSA**. Infatti per ogni file viene generata una diversa chiave di cifratura simmetrica con l'algoritmo selezionato dall'utente (istanziato dinamicamente tramite reflection tra uno di quelli che rispondono all'interfaccia **ISymmetricCryptography**). Il modello (**FileInterpret**) si occupa di incapsulare nel file di output la chiave simmetrica utilizzata per cifrare il file in forma cifrata grazie ad RSA. Di conseguenza ogni file sarà cifrato con una chiave simmetrica diversa, in maniera trasparente all'utente. Questa scelta rende superflua la generazione di un "initialization vector" per gli algoritmi simmetrici. Un eventuale aggressore dovrebbe necessariamente risalire alla chiave privata RSA per poter decifrare i dati. Inoltre l'utente può trasmettere senza problemi la propria chiave pubblica a terzi, i quali potranno generare file cifrati che solo l'utente potrà decifrare. A discrezione dell'utente il file cifrato può essere anche compresso con uno degli algoritmi che rispondono all'interfaccia **ICompression** al fine di risparmiare spazio su disco. Le informazioni sull'algoritmo di compressione sono anch'esse memorizzate nel modello. L'utente può inoltre selezionare un algoritmo di hashing da allegare al file cifrato, in modo da verificarne l'integrità nel momento in cui viene decifrato. L'applicazione memorizza nel file di output anche il nome del file originale, in modo da poterlo ripristinare con facilità qualora il file cifrato venisse rinominato. Infine l'utente può anche eliminare il file di origine con un algoritmo di wiping (classe **Wiping**) scegliendo il numero di sovrascritture da effettuare. È rilevante specificare che questo tipo di algoritmo può non risultare efficace se eseguito su file system con journaling, in quanto le politiche di allocazione dei settori del disco potrebbero far sì che il file originario non sia sovrascritto, o lo sia solo parzialmente. Per avere la certezza dell'eliminazione su questi tipi di file system sarebbe necessario eseguire il wiping anche dello spazio libero su disco. Questa opzione non è stata implementata nel programma in quanto ritenuta inutilmente onerosa nella maggior parte dei casi.

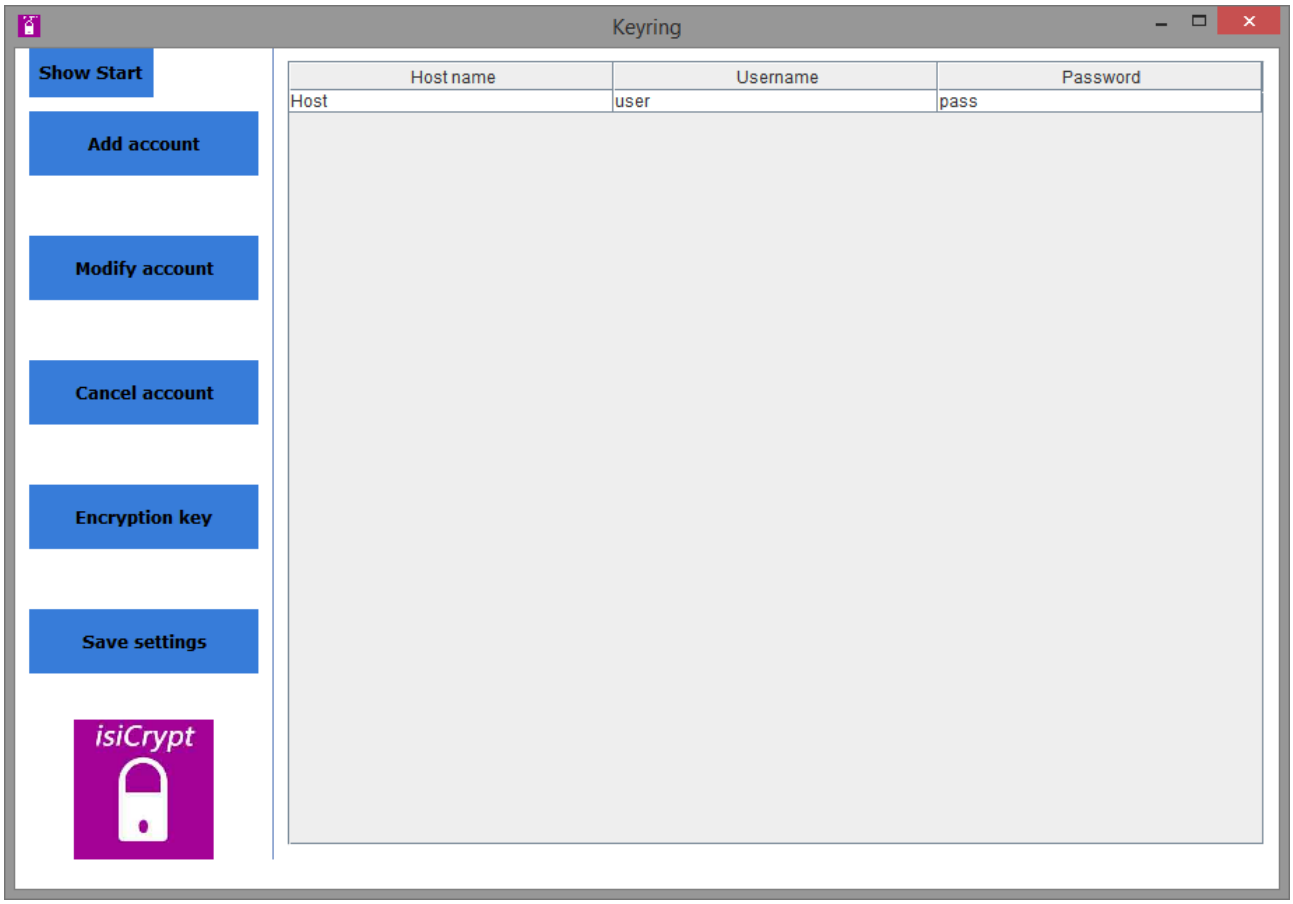
La parte sinistra dell'interfaccia (**CryptographyView**) si occupa delle funzioni di crittografia, mentre dalla parte destra l'utente può selezionare un file da decrittografare e una chiave privata **RSA** per decriptare la chiave simmetrica. L'integrità del file viene verificata durante l'operazione di decrittografia. Nell'interfaccia grafica sono presenti anche due progress bar e una text area di log che informano l'utente sullo stato del processo.

Una nota sull'implementazione di **AES**: inizialmente era prevista la possibilità di selezionare chiavi di lunghezza diversa (128, 192, 256 bit), in accordo con le specifiche dell'algoritmo. Tuttavia le impostazioni predefinite del JDK non consentono chiavi più lunghe di 128 bit a causa di alcune leggi sull'esportazione statunitensi. Questo limite è aggirabile installando una policy disponibile sul sito di Oracle (Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy). Al momento la lunghezza delle chiavi nell'applicazione è limitata a 128 bit, un valore considerabile relativamente sicuro, non essendo ancora noti metodi efficaci per violarlo.

Segue una descrizione dettagliata delle classi e interfacce usate, secondo il pattern MVC.

- In **CryptographyView**, che estende da **AbstractGuiMethodSetter**, è realizzata l'interfaccia grafica tramite GridBagLayout come da screenshot nella pagina precedente. Implementa l'interfaccia **ICryptographyView**, attraverso la quale il controller comanda la GUI.
- **FileInterpret** costituisce il modello. Consiste in una classe che funge da wrapper per i file selezionati dall'utente. Nella modalità di crittografia consente di incapsulare il file cifrato in uno più grande contenente metadati sullo stesso (algoritmi utilizzati e relative chiavi, checksum), mentre in modalità di decrittografia consente di leggere un file precedentemente memorizzato in questo formato e di estrarre informazioni da esso. Implementa l'interfaccia **IFileInterpret**, attraverso la quale il controller invia comandi al modello, e *Serializable*, per poter scrivere su disco il file wrapper. Lancia l'eccezione **CorruptedDataException**, creata ad hoc per l'applicazione, qualora l'hash di un file non coincida con quello calcolato precedentemente.
- **CryptographyController** si occupa di inviare comandi alla view e al model, oltre ad implementare le funzioni di crittografia proprie di questa parte del programma. Implementa l'interfaccia **IGeneralViewObserver**, che fornisce un solo metodo che consente di ritornare alla start screen del programma; implementa l'interfaccia **ICryptographyViewObserver**, di cui ognuno dei metodi corrisponde alla pressione di un pulsante sulla view. Il controller utilizza la classe **OpenButtons** per presentare all'utente una finestra di dialogo di selezione di file e cartelle. Il controller utilizza alcune classi del package *algorithms*, oltre ad interfacce del package *algorithms.interfacesandabstractclasses*. In particolare si serve di:
    - un algoritmo di cifratura simmetrica che risponde all'interfaccia **ISymmetricCryptography**;
    - un algoritmo di cifratura asimmetrica (**RSA**), che risponde all'interfaccia **IAsymmetricCryptography**;
    - un algoritmo realizzato con pattern Singleton di compressione che risponde all'interfaccia **ICompression**;
    - un algoritmo di cancellazione sicura dei file realizzato con pattern Singleton (**Wiping**), che risponde all'interfaccia **IWiping**.
- Una descrizione dettagliata della struttura dei sorgenti utilizzati in questa porzione del programma relativi ai package *algorithms* e *algorithms.interfacesandabstractclasses* è presente in corrispondenza dei primi due diagrammi UML della relazione. Tutte le classi e interfacce degli algoritmi utilizzati in questa porzione di programma sono state realizzate autonomamente da questo componente del gruppo nell'ambito del progetto sfruttando le implementazioni già presenti in Java, ad eccezione dell'algoritmo di wiping, che è stato implementato da zero prendendo ispirazione da alcuni esempi online, e di hashing, che costituisce parte del lavoro dell'altro componente del gruppo.

Funzione di password manager



Appena l'interfaccia (**KeyringView**) viene aperta viene chiesto all'utente di selezionare una chiave di cifratura **AES** per leggere un database eventualmente presente grazie al modello dell'applicazione (**KeyringModel**). Se l'utente non individua una chiave, l'utente potrà procedere alla creazione di un nuovo database, ma non potrà salvarne il contenuto finché non avrà generato una chiave per questo scopo. Sempre attraverso il modello i dati possono essere salvati su file nella cartella dell'applicazione, situata nella home dell'utente, per una consultazione futura.

Al momento per proteggere il database viene utilizzata una chiave di cifratura memorizzata su file, ma in futuro si prevede di aggiungere la possibilità di utilizzare una password a scelta dell'utente, dalla quale sarà derivata la chiave di cifratura vera e propria.

Segue una descrizione dettagliata delle classi e interfacce usate, secondo il pattern MVC.

- In **KeyringView**, che estende da **AbstractGuiMethodSetter**, è realizzata l'interfaccia grafica tramite GridBagLayout come da screenshot qui sopra. Implementa l'interfaccia **IKeyringView**, attraverso la quale il controller comanda la GUI.
- **KeyringModel** costituisce il modello, che si occupa di salvare il database creato dall'utente sotto forma di HashSet di **Triple** (classe generica che memorizza gruppi di tre valori) composte da tre stringhe rappresentanti il nome dell'host, un username e una password. La scelta di un Set come struttura dati è motivata dall'assunzione secondo cui è ragionevole pensare che all'interno del database non esistano record duplicati, sebbene ad uno stesso host possano essere associati più account. Il modello implementa l'interfaccia **IKeyringModel**, attraverso la quale il controller invia comandi al modello. Sia **KeyringModel** che **Triple** implementano *Serializable*, al fine di salvare su disco il database.

- In **Triple** sono stati ridefiniti i metodi *hashCode()* e *equals()*, al fine dell'individuazione di duplicati durante l'inserimento nel Set.
- **KeyringController** si occupa di inviare comandi alla view e al model, oltre a sfruttare le funzioni di crittografia della classe **AES** del package *algorithms*. Implementa l'interfaccia **IGeneralViewObserver**, che fornisce un solo metodo che consente di ritornare alla start screen del programma; implementa l'interfaccia **IKeyringViewObserver**, di cui ognuno dei metodi corrisponde alla pressione di un pulsante sulla view. La classe **AES** viene utilizzata per fornire le funzionalità di crittografia. Il controller utilizza inoltre la classe **OpenButtons** per presentare all'utente una finestra di dialogo di selezione di file e cartelle.

Salvataggio file impostazioni globali

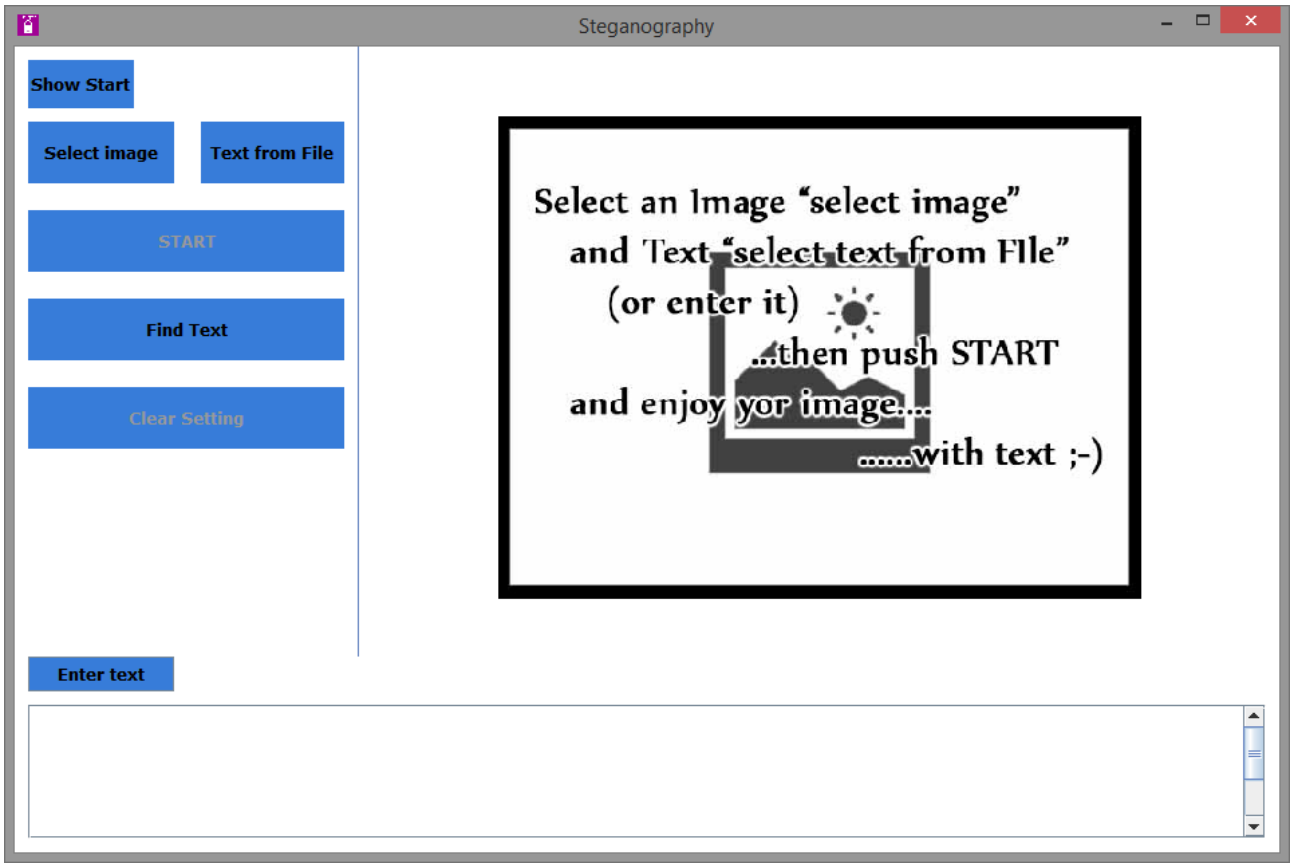
- La classe **GlobalSettings** realizza questa funzione, implementando *Serializable*, salvando se stessa su file all'interno della directory home dell'utente.

5.2 Progettazione di dettaglio: parte di Filippo Vimini

Descrizione classi generali

- **ThemeChooser**. Classe che gestisce i temi grafici dell'intera applicazione. La sua funzione è impostare un tema scelto dall'utente tra quelli disponibili per personalizzare l'aspetto dell'applicazione. L'ultimo tema selezionato viene salvato su disco dalla classe **GlobalSettings** in modo da ripristinarlo all'avvio successivo.
- **OpenButtons**. Fornisce funzioni per selezionare da file system file generici, specifici (come JPG e TXT) e cartelle a scelta dell'utente.
- **Hashing**. La classe implementa le funzioni per il calcolo degli algoritmi di hashing, in particolare può essere effettuato il controllo dell'integrità di un file tramite algoritmo MD5 e SHA1. È realizzata con il pattern Singleton. Implementa l'interfaccia **IHashing**.
- **TypeConverter**. Raccoglie l'insieme dei metodi che convertono tipi di dato in più punti dell'applicazione. In particolare permette la conversione: File a byte[]; File a String; int a byte[]; byte[] a int; byte[] a String; BufferedInputStream a File temporaneo.
- **AbstactGuiMethodSetter**. La classe è una raccolta di metodi specializzati nel compito di impostare parametri per disegnare la GUI in modo più ordinato all'interno di un GridBagLayout, oltre al disegno dei pulsanti e il ridimensionamento delle immagini. Tutte le View dell'applicazione estendono da questa classe.

Funzione di steganografia



La parte di steganografia si occupa di nascondere un testo dentro un'immagine. Per prima cosa l'utente deve selezionare un'immagine da filesystem con estensione accettate sono JPG. Una copia dell'immagine viene ridimensionata e visualizzata come icona nella GUI, l'altra resta in attesa che l'utente selezioni un testo o lo scriva nel box situato nella parte inferiore. Quando sia l'immagine che il testo sono selezionati l'utente può iniziare la steganografia: la copia dell'immagine selezionata viene trasformata in un raster per poter operare con l'array di byte che la compone; anche il testo e la sua lunghezza totale vengono convertiti in array di byte. Il passo successivo consiste nel nascondere l'array di byte che compone il testo e l'array di byte che rappresenta la lunghezza del testo dentro il raster dell'immagine. La chiave per fare ciò è intrinseca nella struttura del byte, infatti i bit che lo compongono hanno un rango: quello a sinistra è il più significativo, mentre quello a destra è il meno significativo. Così l'algoritmo, scorre i byte dell'array dell'immagine sostituendone il bit meno significativo con quello della lunghezza del testo (massimo un integer, 32 bit), poi con quello del testo stesso. Il testo verrà cosi nascosto dentro l'immagine senza che questa ne subisca cambiamenti notevoli e rendendo impossibile il ritrovamento del testo senza sapere l'algoritmo che lo ha generato. L'immagine verrà automaticamente salvata in un percorso deciso dall'utente in formato PNG perché questo formato non comprime l'immagine rischiando quindi di eliminare o rovinare il testo.

L'applicazione permette all'utente anche di ritrovare il testo nascosto, una volta selezionata l'immagine, l'algoritmo la trasforma in un raster e controlla, tutti i bit meno significativi dell'array per la lunghezza di un integer, così ottiene la lunghezza totale del testo. Continua così il controllo per tutta la lunghezza del testo e lo converte in una stringa. Se non è presente testo, generalmente i bit presi in analisi per la lunghezza del testo superano il valore di un intero (Integer) e viene segnalato un messaggio di errore. Può anche non accadere ma in questo caso viene visualizzato o un testo nullo o un testo senza significato.

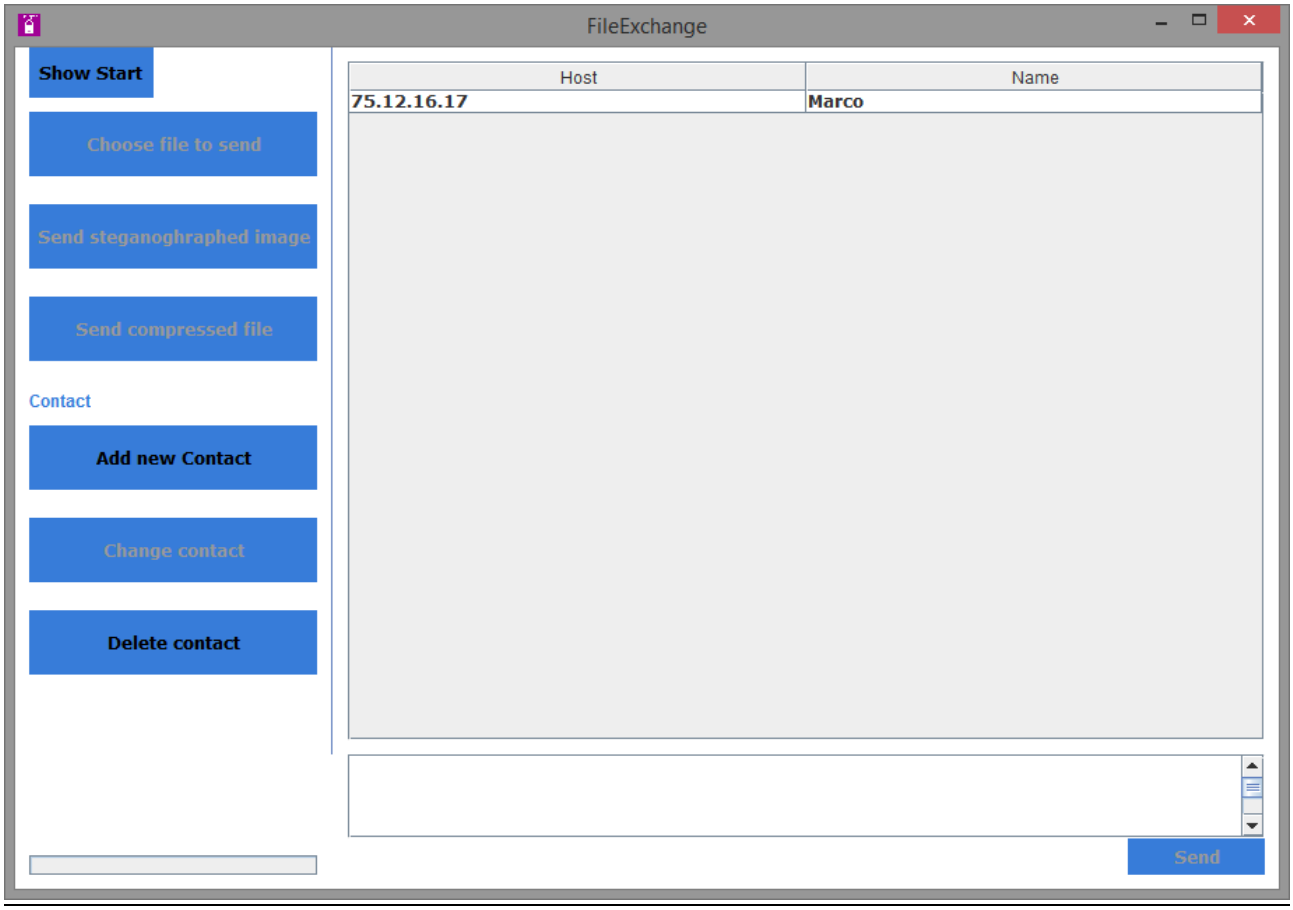
Per la realizzazione dell'algoritmo e parte della formattazione del codice sono state seguite le linee guida presenti al seguente link: http://www.dreamincode.net/forums/topic/27950-steganography/

È noto un bug grafico grafico minore dell'anteprima dell'immagine quando si inseriscono immagini con orientamento portrait.

Segue una descrizione dettagliata delle classi e interfacce usate, secondo il pattern MVC. Nessuna classe si occupa della parte "model" poiché in questa funzione non vengono gestiti dati.

- **Steganography**: Questa classe si occupa degli algoritmi per nascondere il testo dentro l'immagine, utilizzando operatori bitwise. Per la sua gestione è stata creata l'interfaccia **ISteganography**.
- **SteganographyView**: La classe si occupa di gestire la parte grafica (GUI), tramite GridBagLayout. È acceduta tramite l'interfaccia **ISteganographyView** e estende da **AbstactGuiMethodSetter**.
- **SteganographyController**: La classe funge da controller, e associa gli elementi grafici della View alle funzioni realizzate nella classe Steganography. È acceduta tramite la sua interfaccia **ISteganographyViewObserver**. Utilizza **OpenButtons** per prelevare file immagine e testo da disco.

Funzione di scambio file sicuro



L'applicazione è basata su server (**SocketServer**), che si occupa di ricevere informazioni, e client (**SocketClient**), di inviarle. Quando l'utente entra nello scambio file, viene avviato il server su un thread separato che monitora, tramite protocollo TCP, i dati entranti sulla porta di default (19999). Viene visualizzata la tabella con tutti i contatti salvati precedentemente e viene data la possibilità di aggiungerne o eliminarne altri. Il contatto viene selezionato con un doppio click sulla riga corrispondete e vengono salvate le informazioni dell'ip del contatto da usare come host, dal quale il client invierà i messaggi. Inoltre vengono abilitate, oltre alla chat, le opzioni per scambiare file, immagini con testo nascosto e file

compressi. La tabella viene sostituita con un'area di testo dove vengono visualizzati i messaggi inviati e ricevuti.

Lo scambio di dati fra server e client è messo in sicurezza dalla crittografia tramite AES. Per far ciò server e client si scambiano chiave pubblica tramite algoritmo Diffie-Hellman.

Non essendo controllata da server online, la connessione deve avvenire direttamente fra le parti e quindi per funzionare dietro NAT richiede la configurazione del portfowarding nel router per reindirizzare le connessioni allo specifico host.

Purtroppo non essendoci un controllo sui contatti online viene attivato lo scambio file anche con server non connessi. Ne risulta che quando si invia qualcosa l'applicazione si blocca finché non viene ricevuto un messaggio di errore dal client.

Per poter eseguire uno scambio deve essere visualizzata l'area di testo, ovvero devono essere selezionati i reciproci contatti dalla tabella

Segue una descrizione dettagliata delle classi e interfacce usate, secondo il pattern MVC.

- **SocketClient**: La classe si occupa delle funzioni di client tramite socket su protocollo TCP. Gestisce inoltre lo scambio di chiavi seguendo l'algoritmo Diffie-Hellman grazie alla classe **RSA**, in modo da rendere possibile la criptazione dei dati. SocketClient invia dati cifrati grazie alle funzioni disponibili nella classe **AES**.
- **SocketServer**. La classe si occupa delle funzioni di server tramite socket su protocollo TCP. Resta in ascolto sulla porta assegnata in attesa di dati; gestisce l'algoritmo Diffie-Hellman per lo scambio di chiavi con il client grazie alla classe **RSA**, in modo tale da rendere possibile la decriptazione, tramite classe **AES**, dei dati.
- **FileExchangeView**. La classe si occupa di gestire la parte grafica (GUI), tramite GridBagLayout. Viene gestita tramite interfaccia **IFileExchangeView**.
- **FileExchangeController**. Classe che realizza la parte del controller. Il suo scopo è collegare la parte grafica agli algoritmi impiegati e ai dati del modello. È gestito tramite interfaccia **IFileExchangeViewObserver**. Utilizza **OpenButtons** per richiedere all'utente i percorsi in cui salvare i file scambiati.
- **FileExchangeModel**. Gestisce la lista di contatti di tipo **ContactInfo** tramite HashMap di coppie di stringhe. Si occupa di salvare su disco i contatti inseriti dall'utente, oltre a ricaricarli all'avvio. È gestita tramite interfaccia **IFileExchangeModel**.
- **ContactInfo**. È utilizzata da **FileExchangeModel** per rappresentare un contatto selezionato dall'utente durante la sessione di scambio file.

Schermata di avvio

- **StartScreenView**. Questa classe ha la funzione di rappresentare graficamente le opzioni della finestra di apertura del programma, corrispondenti alle funzionalità dell'applicazione. Gestisce l'abilitazione dei pulsanti permettendo l'esecuzione di una sola istanza per funzione, andando così a limitare i problemi di concorrenza e migliorando la semplicità d'uso. Viene acceduta tramite la sua interfaccia **IStartScreenView**.
- **StartScreenController**. Gestisce il collegamento fra la parte View e quella Model della finestra iniziale dell'applicazione. Implementa l'interfaccia **IStartScreenViewObserver**.

Librerie utilizzate

Sono state utilizzate funzioni della libreria Apache per controllare String e I/O, mentre è stata usata la JUnique per rendere l'applicazione instanziabile una sola volta evitando cosi problemi di concorrenza.

6 Testing

Il testing dell'applicazione è stato eseguito in più fasi, procedendo parallelamente allo sviluppo del codice. Inizialmente sono stati testati i sorgenti del package "algorithms" individualmente attraverso classi di test in cui venivano creati oggetti di ogni tipo richiesto dagli algoritmi e ne verificavano il funzionamento testandone i metodi uno a uno.

Successivamente, col procedere della realizzazione delle interfacce grafiche, gli algoritmi precedentemente testati sono stati combinati tra loro nelle classi "controller" al fine di fornire le funzionalità di alto livello offerte all'utente. Ad esempio In più punti del programma viene utilizzato l'algoritmo di cifratura asimmetrica RSA al fine di proteggere le chiavi generate da un algoritmo di cifratura simmetrico (AES).

Contestualmente allo sviluppo dei controller è avvenuta l'implementazione delle viste e dei modelli. Una volta ultimati, sono stati collegati al controller per testarne le funzionalità.

Una volta terminata la fase di sviluppo, l'applicazione è stata sottoposta ad una serie di test sia su sistema operativo Microsoft Windows 8.1, sia su Ubuntu 14.04 con JDK 8. È stato riscontrato un bug grafico che si presenta talvolta su alcune configurazioni Mac OS X. Non è stato possibile studiare a fondo tale bug a causa della non disponibilità di un computer con quel sistema operativo.

7 Note finali

Durante la realizzazione del progetto ci si è resi conto che il tempo necessario per implementare le varie funzionalità era diverso da quanto preventivato in fase di progettazione. In particolare la creazione delle view è stata più complessa del previsto, sia a causa della scelta di rendere l'interfaccia grafica il più ridimensionabile e precisa possibile, sia a causa di alcuni bug sopraggiunti in seguito che hanno dilatato considerevolmente la fase di test finale. In particolare abbiamo riscontrato bug su sistemi operativi e versioni di java differenti da quelli utilizzate per lo sviluppo.

Nonostante questi problemi la divisione del lavoro tra i componenti del gruppo è rimasta equilibrata, in quanto le maggiori difficoltà riscontrate da uno dei due si manifestavano anche sull'altro. Di conseguenza non abbiamo apportato cambiamenti significativi alla suddivisione del lavoro in corso d'opera.

L'applicazione consegnata ai fini del progetto è completa nelle sue funzionalità. Tuttavia verrà portata avanti anche in seguito al fine di migliorare ulteriormente l'esperienza utente e introdurre nuove funzionalità, tra cui una nuova interfaccia grafica per la gestione e la memorizzazione di più chiavi di crittografia, l'integrazione con uno o più servizi di cloud storage, un'interfaccia command line per consentire l'esecuzione di task in modalità batch, l'implementazione di altri algoritmi (anche non già presenti nelle classi di Java), un metodo per riconoscere quali contatti sono on-line all'apertura dello scambio file e altro ancora.