Meta-relazione per "Programmazione ad Oggetti"

Giovanni Di Santi

24 settembre 2020

Sommario

Questo documento è una relazione di meta livello, ossia una relazione che spiega come scrivere la relazione. Lo scopo di questo documento è quello di aiutare gli studenti a comprendere quali punti trattare nella loro relazione, ed in che modo farlo, evitando di perdere del tempo prezioso in prolisse discussioni di aspetti marginali tralasciando invece aspetti di maggior rilievo. Per ciascuna delle sezioni del documento sarà fornita una descrizione di ciò che ci si aspetta venga prodotto dal team di sviluppo, assieme ad un elenco (per forza di cose non esaustivo) di elementi che *non* dovrebbero essere inclusi.

Il modello della relazione segue il processo tradizionale di ingegneria del software fase per fase (in maniera ovviamente semplificata). La struttura della relazione non è indicativa ma *obbligatoria*. Gli studenti dovranno produrre un documento che abbia la medesima struttura, non saranno accettati progetti la cui relazione non risponda al requisito suddetto. Lo studente attento dovrebbe sforzarsi di seguire le tappe suggerite in questa relazione anche per l'effettivo sviluppo del progetto: oltre ad una considerevole semplificazione del processo di redazione di questo documento, infatti, il gruppo beneficerà di un processo di sviluppo più solido e collaudato, di tipo top-down.

La meta-relazione verrà fornita corredata di un template LaTeX per coloro che volessero cimentarsi nell'uso. L'uso di LaTeX è vantaggioso per chi ama l'approccio "what you mean is what you get", ossia voglia disaccoppiare il contenuto dall'effettivo rendering del documento, accollando al motore LaTeX l'onere di produrre un documento gradevole con la struttura ed il contenuto forniti. Chi non volesse installare l'ambiente di compilazione in locale può valutare l'utilizzo dell'applicazione web Overleaf. L'eventuale utilizzo di LaTeX non è fra i requisiti, non è parte del corso di Programmazione ad Oggetti, e non sarà ovviamente valutato. I docenti accetteranno qualunque relazione in formato standard Portable Document Format (pdf), indipendentemente dal software con cui tale documento sarà redatto.

Indice

Capitolo 1

Analisi

1.1 Requisiti

Il gruppo si pone come obiettivo quello di realizzare un keepass per salvare in modo sicuro le password. Al momento i keepass desktop più usati sono keepass2 e keepassxc.

Requisiti funzionali

- Gestione account con possibilità di suddividere gli account in gruppi.
- Funzione per generare password e nickname in maniera causale.
- Import ed export del database in XML e in un formato simile a KDBX.
- Controllo robustezza password al momento dell'inserimento
- Salvataggio dei dati in un database locale criptato con algoritmi a scelta
- Sezione Statistics che mostra le statistiche relative al proprio database (Es. il numero di account salvati)

Requisiti non funzionali

- bho

1.2 Analisi e modello del dominio

In questa sezione si descrive il modello del *dominio applicativo*, descrivendo le *entità* in gioco ed i rapporti fra loro. Si possono sollevare eventuali aspetti

particolarmente impegnativi, descrivendo perché lo sono, senza inserire idee circa possibili soluzioni, ovvero sull'organizzazione interna del software. Infatti, la fase di analisi va effettuata **prima** del progetto: né il progetto né il software esistono nel momento in cui si effettua l'analisi. La discussione di aspetti propri del software (ossia, della *soluzione* al problema e non del problema stesso) appartengono alla sfera della progettazione, e vanno discussi successivamente.

È obbligatorio fornire uno schema UML del dominio, che diventerà anche lo scheletro della parte "entity" del modello dell'applicazione, ovvero degli elementi costitutivi del modello (in ottica MVC - Model View Controller): se l'analisi è ben fatta, dovreste ottenere una gerarchia di concetti che rappresentano le entità che compongono il problema da risolvere. Un'analisi ben svolta **prima** di cimentarsi con lo sviluppo rappresenta un notevole aiuto per le fasi successive: è sufficiente descrivere a parole il dominio, quindi estrarre i sostantivi utilizzati, capire il loro ruolo all'interno del problema, le relazioni che intercorrono fra loro, e reificarli in interfacce.

Elementi positivi

- Viene descritto accuratamente il modello del dominio.
- Alcuni problemi, se non risolubili in assoluto o nel monte ore, vengono dichiarati come problemi che non saranno risolti o sarano risolti in futuro.
- Si modella il dominio in forma di UML, descrivendolo appropriatamente.

Elementi negativi

- Manca una descrizione a parole del modello del dominio.
- Manca una descrizione UML delle entità del dominio e delle relazioni che intercorrono fra loro.
- Vengono elencate soluzioni ai problemi, invece della descrizione degli stessi.
- Vengono presentati elementi di design, o peggio aspetti implementativi.
- Viene mostrato uno schema UML che include elementi implementativi o non utili alla descrizione del dominio, ma volti alla soluzione (non

devono vedersi, ad esempio, campi o metodi privati, o cose che non siano equivalenti ad interfacce).

Esempio

GLaDOS dovrà essere in grado di accedere ad un'insieme di camere di test. Tale insieme di camere prende il nome di percorso. Ciascuna camera è composta di challenge successivi. GLaDOS è responsabile di associare a ciascun challenge un insieme di consigli (suggestions) destinati all'utente (subject), dipendenti da possibili eventi. GLaDOS dovrà poter comunicare coi locali cucina per approntare le torte. Le torte potranno essere dolci, oppure semplici promesse di dolci che verranno disattese.

La difficoltà primaria sarà quella di riuscire a correlare lo stato corrente dell'utente e gli eventi in modo tale da generare i corretti suggerimenti. Questo richiederà di mettere in campo appropriate strategie di intelligenza artificiale.

Data la complessità di elaborare consigli via AI senza intervento umano, la prima versione del software fornita prevederà una serie di consigli forniti dall'utente.

Il requisito non funzionale riguardante il consumo energetico richiederà studi specifici sulle performance di GLaDOS che non potranno essere effettuati all'interno del monte ore previsto: tale feature sarà oggetto di futuri lavori.

Capitolo 2

Design

2.1 Architettura

Per la realizzazione di KeePassJ abbiamo scelto di utilizzare il pattern architetturale Model-View-Controller (MVC).

TODO: Inserire schema UML MVC.

2.2 Design dettagliato

TODO: Spiegare a grandi linee

Giovanni Di Santi

Il mio compito principale del progetto è stato quello di gestire la parte crittografica e definire la struttura dell'header del database.

CryptoCipher

CryptoCipher è l'interfaccia che descrive i metodi necessari per effettuare l'encryption e la decryption di un array di `byte`.
Ogni implementazione disponibile di questa interfaccia è un AEAD Cipher (Authenticated Encryption with Associated Data).
Ho scelto questo schema di encryption per rendere il database resistente ad attacchi del tipo **CCA** (Chosen Ciphertext Attack), cifrando il contenuto del database e autenticando sia il contenuto che l'header (Associated Data).
Attualmente i cipher disponibili sono:

- ChaCha20-Poly1305.

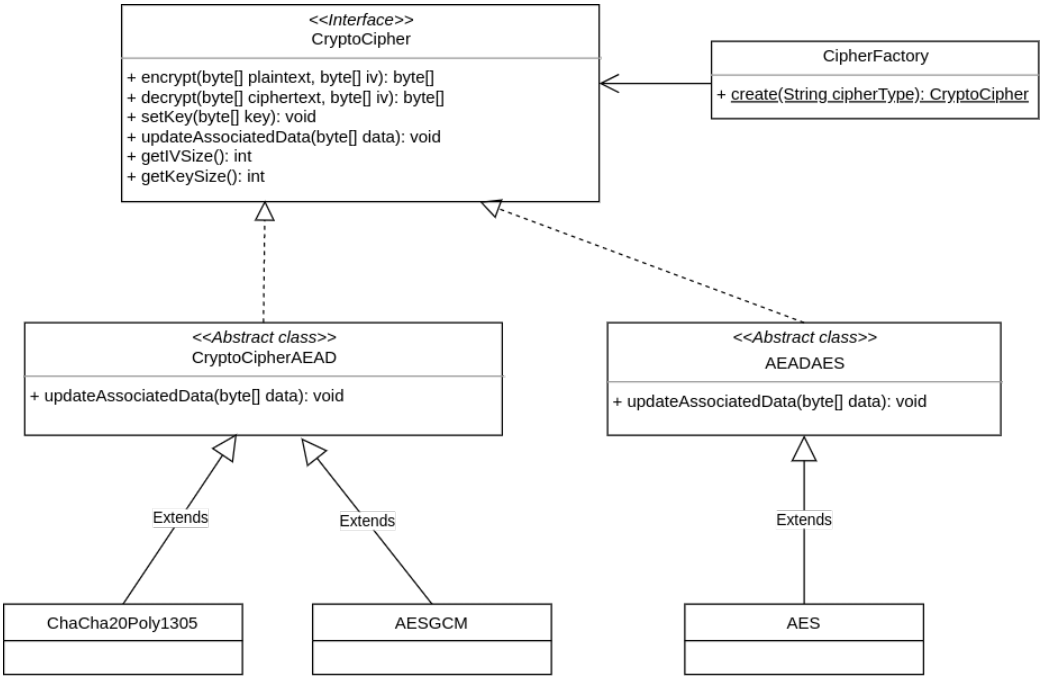

Figura 2.1: rappresentazione UML del pattern factory per creare un CryptoCipher

- AES-GCM.
- AES-256-CBC-HMAC-SHA-512.

Esistono due `abstract class` diverse per implementare un **CryptoCipher**, poiché la costruzione di **AES** che sarebbe `AES-256-CBC-HMAC-SHA-512` è manuale, mentre **ChaCha20-Poly1305** e **AES-GCM** sono implementate direttamente in openjdk11. La classe astratta **AEADAES**, permette di essere estesa per costruire altri cryptosystem come `AES-192-CBC-HMAC-SHA-384`, tuttavia ho deciso di estendere solo lo schema più sicuro.
Nonostante i dati da cifrare e decifrare sono nella pratica degli {Input,Output}Stream, non ho usato le classi CipherOutputStream e CipherInputStream per:

- Rendere più semplice il suo utilizzo.
- Facilitare il testing delle varie implementazioni.

KDF

KDF (Key Derivation Function) è l'interfaccia che descrive i metodi necessari e opzionali per generare una chiave simmetrica per cifrare/decifrare il

database.
Gli algoritmi disponibili sono:

- Argon2.
- Scrypt.
- PBKDF2.

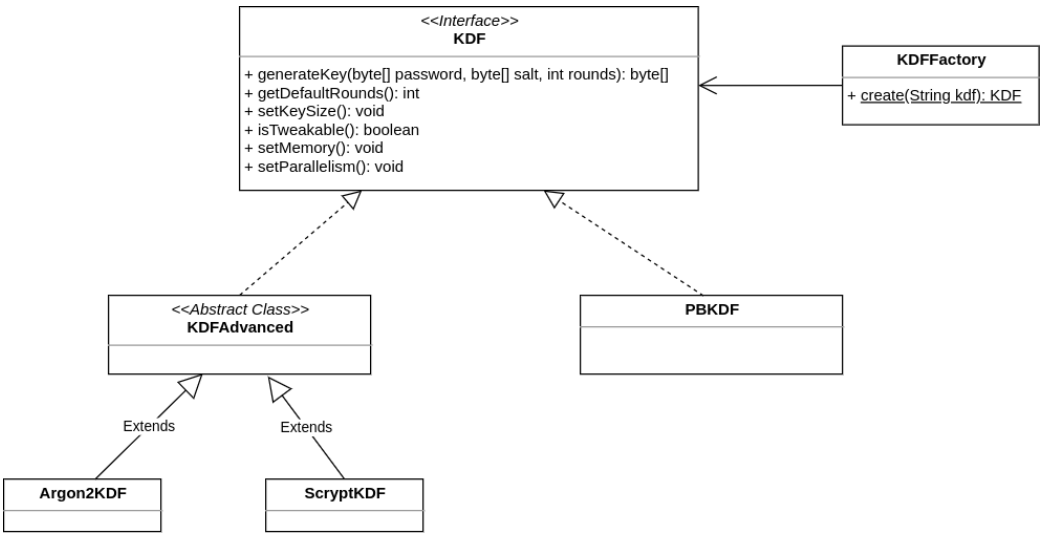

Figura 2.2: rappresentazione UML del pattern factory per creare un KDF

Argon2 e **Scrypt** estendono **KDFAdvanced** poiché i loro algoritmi permettono di definire parametri extra come il paralellismo e la memoria usata dal KDF. **PBKDF2** è un vecchio metodo per generare una chiave dalla password e l'unico parametro configurabile è il numero di round che usa internamente, per questo ho settato il campo `tweakable` a falso.

Per capire perché il pattern `Factory` è usato sia per creare **KDF** e **CryptoCipher** bisogna prima analizzare il parsing dell'header e la relativa encryption/decryption del database.

KDB

Per progettare questa parte non ho usato le interfacce perché:

- Ho solo una implementazione disponibile.
- Sono più flessibile quando devo cambiare la signature di un metodo, senza dover usare un IDE o un LSP per il refactoring.

- Principio YAGNI e KISS.

KDBHeader è la classe che si occupa di:

- Parsare l'header (Lettura).
- Configurare i vari parametri (Scrittura).

KDB è la classe che tramite il **KDBHeader** si occupa di cifrare/decifrare dati arbitrari. Il pattern factory per **CryptoCipher** e **KDF** è utile quando in **KDB** si effettua l'operazione `encrypt` e `decrypt`. I metodi richiedono a **KDBHeader** il valore (String) del `Cipher` e del `KDF` che viene passato come parametro di {Cipher,KDF}Factory.create() per generare l'oggetto richiesto. I due metodi pubblici principali di **KDB** sono:

- `write`: che esegue l'encryption dell'array di byte in input e lo scrive sul file.
- `read`: che legge il file e lo decrypta. Questo metodo lancia un `IOException` se il file non esiste o un `AEADBadTagException` quando il file è corrotto. Il file può risultare corrotto o perché la password è sbagliata o perché è stato effettivamente manipolato. Non lancio tipi diversi di eccezioni (es. `BadPaddingException`) in base a vari tipi di errore, per evitare vari tipi di attacchi (praticamente difficili, ma teoricamente possibili) come il padding oracle.

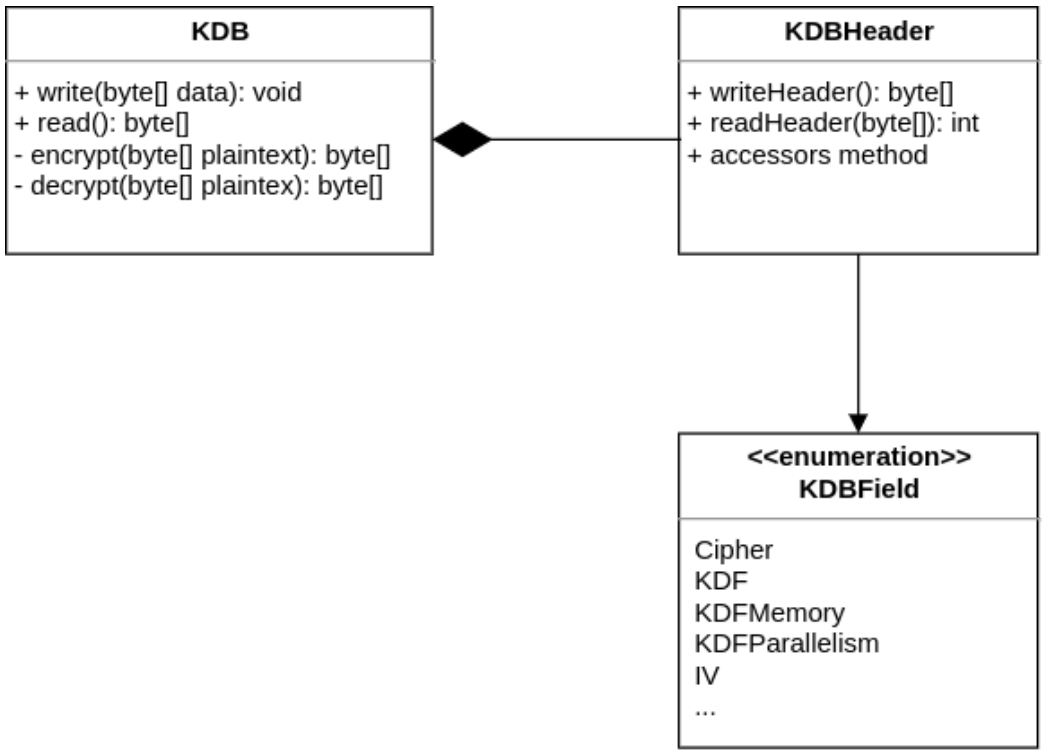

Figura 2.3: rappresentazione UML di KDB

Capitolo 3

Sviluppo

3.1 Testing automatizzato

Durante lo sviluppo del nostro progetto abbiamo utilizzato `Junit` per testare il corretto funzionamento delle varie classi. **Funzionalità testate automaticamente**:

- *CryptoCipher*: viene testato il corretto comportamento dei vari cipher utilizzando lo specifico `Factory`.
- *KDF*: vengono testate le varie KDF utilizzando lo specifico `Factory`.
- *Crypto Util*: test di varie utility per crittografia, come il PKCS#7 padding e SHA256.
- *KDBHeader*: viene testato il corretto funzionamento del **KDBHeader**, settando vari parametri e controllando il corretto funzionamento dei getter.
- *KDB*: test sulla scrittura e lettura di diverse combinazioni di `cipher` e `KDF`.

I test vengono eseguiti anche in remoto tramite l'apposita *bitbucket pipelines* che esegue la build del progetto e lancia i vari test.

3.2 Metodologia di lavoro

TODO.

3.3 Note di sviluppo

Giovanni Di Santi

- *javax.crypto*: usata per lavorare con Cipher, KDF, e MessageDigest.
- *Stream*: usata per manipolare l'header in modo efficace ed elegante.
- *Libreria Google Guava*: lavorare con i byte array in java non è comodo. Questa libreria ha varie classi per semplificare il lavoroi tra cui Bytes.
- *ByteBuffer*: per convertire in little endian i bytes, i Data{Input,Output}Stream in java lavorano solo in big-endian.
- *Libreria Argon2 e Scrypt*: Questi due KDF non erano disponibili dentro `javax.crypto`.
- *Libreria Apache Commons*: per convertire byte array in formato esadecimale.

Capitolo 4

Commenti finali

In quest'ultimo capitolo si tirano le somme del lavoro svolto e si delineano eventuali sviluppi futuri.

Nessuna delle informazioni incluse in questo capitolo verrà utilizzata per formulare la valutazione finale, a meno che non sia assente o manchino delle sezioni obbligatorie. Al fine di evitare pregiudizi involontari, l'intero capitolo verrà letto dai docenti solo dopo aver formulato la valutazione.

4.1 Autovalutazione e lavori futuri

È richiesta una sezione per ciascun membro del gruppo, obbligatoriamente. Ciascuno dovrà autovalutare il proprio lavoro, elencando i punti di forza e di debolezza in quanto prodotto. Si dovrà anche cercare di descrivere *in modo quanto più obiettivo possibile* il proprio ruolo all'interno del gruppo. Si ricorda, a tal proposito, che ciascuno studente è responsabile solo della propria sezione: non è un problema se ci sono opinioni contrastanti, a patto che rispecchino effettivamente l'opinione di chi le scrive. Nel caso in cui si pensasse di portare avanti il progetto, ad esempio perché effettivamente impiegato, o perché sufficientemente ben riuscito da poter esser usato come dimostrazione di esser capaci progettisti, si descriva brevemente verso che direzione portarlo.

4.2 Difficoltà incontrate e commenti per i docenti

Questa sezione, **opzionale**, può essere utilizzata per segnalare ai docenti eventuali problemi o difficoltà incontrate nel corso o nello svolgimento del

progetto, può essere vista come una seconda possibilità di valutare il corso (dopo quella offerta dalle rilevazioni della didattica) avendo anche conoscenza delle modalità e delle difficoltà collegate all'esame, cosa impossibile da fare usando le valutazioni in aula per ovvie ragioni. È possibile che alcuni dei commenti forniti vengano utilizzati per migliorare il corso in futuro: sebbene non andrà a vostro beneficio, potreste fare un favore ai vostri futuri colleghi. Ovviamente *il contenuto della sezione non impatterà il voto finale.*

Appendice A

Guida utente

Capitolo in cui si spiega come utilizzare il software. Nel caso in cui il suo uso sia del tutto banale, tale capitolo può essere omesso. A tal riguardo, si fa presente agli studenti che i docenti non hanno mai utilizzato il software prima, per cui aspetti che sembrano del tutto banali a chi ha sviluppato l'applicazione possono non esserlo per chi la usa per la prima volta. Se, ad esempio, per cominciare una partita con un videogioco è necessario premere la barra spaziatrice, o il tasto "P", è necessario che gli studenti lo segnalino.

Elementi positivi

- Si istruisce in modo semplice l'utente sull'uso dell'applicazione, eventualmente facendo uso di schermate e descrizioni.

Elementi negativi

- Si descrivono in modo eccessivamente minuzioso tutte le caratteristiche, anche minori, del software in oggetto.
- Manca una descrizione che consenta ad un utente qualunque di utilizzare almeno le funzionalità primarie dell'applicativo.