ENGINYERIA TÈCNICA EN INFORMÀTICA DE SISTEMES

Curs: 2013/2014

Projecte Final de Carrera

Seguretat Informàtica

PasswordVault

Desenvolupament d'una aplicació segura per a la gestió de contrasenyes robustes

Girona, 13/06/2014

Autor: Carles Figueras Garcia

Tutora: Cristina Pérez Solà

Llicència

Aquesta obra s'ha desenvolupat en el marc d'un projecte de final de carrera en enginyeria tècnica en informàtica de sistemes per a la Universitat Oberta de Catalunya i està subjecta a una llicència de Reconeixement-NoComercial-SenseObraDerivada 3.0 Espanya de Creative Commons.

Agraïments:

A la Cristina i en Marc pel seu sacrifici i suport, sobretot en els darrers dos anys, i per tota la paciència que han tingut i els ànims que m'han traslladat.

A la Cristina Pérez Solà per la seva dedicació i esforç en la direcció d'aquest projecte i per les facilitats i consells que m'ha ofert en tot moment a l'hora de realitzar-lo.

Resum

Aquest projecte està emmarcat dins l'àrea de la seguretat informàtica i pretén desenvolupar una aplicació que permeti la gestió segura de les contrasenyes. Però no només s'ha de centrar en aquesta gestió, sinó que es pretén dotar a l'aplicació de la seguretat necessària, mitjançant algoritmes criptogràfics, per tal que aquestes dades no es puguin comprometre.

A més de garantir la seguretat de les dades també s'ha pensat en que s'ha d'oferir l'oportunitat que les contrasenyes que es gestionen siguin el suficientment robustes com per donar seguretat als serveis als quals donen accés. És per això que aquesta aplicació també ofereix, de forma estimada, el còmput de la robustesa de les contrasenyes gestionades i dóna l'opció d'auto generar contrasenyes aleatòries suficientment robustes.

S'ha implementat una interfície gràfica que permet una fàcil interacció de l'usuari per realitzar totes les operacions d'una forma intuïtiva sense que s'hagin de tenir especials coneixements sobre informàtica.

Índex de continguts

Capítol 1. Introducció 1
1.1 Problema a resoldre 1
1.2 Objectius a assolir 2
1.3 Metodologia de treball 2
1.3.1 Tasques a desenvolupar 2
1.4 Especificacions de les tecnologies a utilitzar 3
1.5 Planificació temporal 3
1.6 Estat de l'art 4
1.6.1 Aplicacions similars 4
1.6.1.1 KeePass 5
1.6.1.2 PasswordSafe 5
1.6.1.3 LastPass 5
1.6.2 Comparativa de funcionalitats 5
1.6.3 Eines criptogràfiques 6
1.6.3.1 Les funcions de hash 6
1.6.3.2 Secure Hash Algorithm 6
1.6.3.3 Seguretat de les funcions de hash 6
1.6.3.4 Bits de sal 7
1.6.3.5 Algoritmes de xifrat de bloc 7
1.6.3.6 Twofish Algorithm 9
1.7 Organització de la memòria 9
Capítol 2. Anàlisi i disseny 10
2.1 Requisits de l'aplicació 10
2.1.1 Operacions sobre els usuaris 11
2.1.1.1 Creació d'usuaris 11
2.1.1.2 Eliminació d'usuaris 11
2.1.2 Operacions sobre els comptes 11
2.1.2.1 Creació d'un compte 11
2.1.2.2 Modificació d'un compte 11
2.1.2.3 Eliminació d'un compte 11
2.2 Diagrama de classes 12
2.3 Disseny de la base de dades 13

2.3.1 Taula Users 13
2.3.2 Taula Accounts 13
Capítol 3. Protocols criptogràfics 15
3.1 Disseny del procés de creació d'usuaris 15
3.1.1 Algoritme criptogràfic del procés de creació d'usuaris 15
3.1.2 Diagrama del procés de creació d'usuaris 15
3.2 Disseny del procés d'autenticació 16
3.2.1 Algoritme criptogràfic del procés d'autenticació 17
3.2.2 Diagrama del procés d'autenticació 17
3.3 Disseny del procés de xifrat 18
3.3.1 Algoritme criptogràfic del procés de xifrat 18
3.3.2 Diagrama del procés de xifrat 19
3.4 Disseny del procés de desxifrat 19
3.4.1 Algoritme criptogràfic del procés de desxifrat 19
3.4.2 Diagrama del procés de desxifrat 20
3.5 Protecció contra col·lisions 21
3.6 Biblioteca criptogràfica utilitzada 21
Capítol 5. Eficiència 22
5.1 Entropia 23
Capítol 6. Conclusions 25
6.1 Millora del procés d'autenticació 25
6.2 Sincronització de les dades al núvol 26
Annex A – Manual d'usuari 27
Annex B – Bibliografia 33
Annex C – Glossari 34

Índex d'il·lustracions i taules

Figura 1 - Taula d'entregues 4
Figura 2 - Diagrama de Gantt 4
Figura 3 - Taula comparativa de funcionalitats 5
Figura 4 - Exemple de taules de permutacions 7
Figura 5 - Esquema general del mode CBC 8
Figura 6 - Diagrama de casos d'ús 10
Figura 7 - Diagrama de classes 12
Figura 8 - Taules de la base de dades 14
Figura 9 - Diagrama de seqüència del procés de creació d'un usuari 16
Figura 10 - Diagrama de seqüència del procés d'autenticació 17
Figura 11 - Diagrama de seqüència del procés de xifrat 19
Figura 12 - Diagrama de seqüència del procés de desxifrat 20
Figura 13–Implementació rudimentària de l'algoritme checkStrength() 22
Figura 14 - Comparativa robustesa amb la implementació rudimentària 23
Figura 15 - Algoritme del càlcul de l'entropia d'una contrasenya 24
Figura 16 - Comparativa de l'entropia de diferents contrasenyes 24
Figura 17 - Interfície de creació d'usuari 27
Figura 18 - Interfície del procés d'autenticació 28
Figura 19 - Interfície principal 28
Figura 20 - Detall del menú File 29
Figura 21 – Interfície d'eliminació d'usuari 29
Figura 22 - Detall dels botons 30
Figura 23 - Detall del menú Edit 30
Figura 24 - Creació d'un compte 30
Figura 25 - Modificació d'un compte 31
Figura 26 - Eliminació d'un compte 31
Figura 27–Altres funcions 32

Capítol 1. Introducció

Amb aquest capítol es vol explicar què és el que es pretén amb el desenvolupament d'aquesta aplicació i veure quins objectius es volen aconseguir. Per això es plantejarà una metodologia i una planificació a seguir per a la seva realització per, finalment, fer una petita ullada a les aplicacions de característiques similars que el mercat ofereix.

1.1 Problema a resoldre

Cada cop es realitzen més accions i es té més presència a la xarxa i això comporta que cada persona necessiti de més usuaris per poder interactuar amb tots els serveis amb els que treballa. Com que cada usuari anirà protegit per una contrasenya es produeix la necessitat que cada persona necessiti més contrasenyes.

Aquest fet per si sol no comporta cap problema en si mateix, però si es vol mantenir una certa seguretat dins la xarxa, s'haurien d'evitar dos dels principals problemes:

- Utilitzar sempre la mateixa contrasenya.
- Utilitzar contrasenyes febles o fàcils d'endevinar.

Si es volen evitar, s'hauran de crear contrasenyes suficientment robustes a atacs de força bruta, diferents per a cada servei i que no es puguin endevinar fàcilment a partir de dades conegudes. Aquestes tres característiques fan que aquestes contrasenyes també siguin difícils de recordar i sigui necessari tenir-les apuntades (amb el risc de comprometement que això comporta).

Per evitar aquesta amenaça potencial es vol crear aquesta eina per tal de poder tenir un arxiu dels diferents usuaris i contrasenyes que una persona ha de fer servir però que estiguin assegurats sota una contrasenya. D'aquesta manera, tota persona només haurà de recordar una sola contrasenya que li donarà accés a la resta.

Aquest fet, a part de ser un avantatge per no haver de recordar multitud de contrasenyes també constitueix el punt més feble ja que si es compromet aquesta contrasenya l'atacant tindria accés a tots els comptes. Així doncs, s'ha de tenir en compte que aquesta contrasenya ha de ser com més robusta millor.

1.2 Objectius a assolir

Amb aquest treball es pretén realitzar una aplicació que permeti protegir sota una contrasenya tota la informació de diversos noms d'usuari, la seva contrasenya i el lloc a on garanteixen l'accés i es crearà una GUI[1] per tal de facilitar l'aprenentatge i l'ús de l'aplicació per part dels usuaris. Per fer-ho es poden dividir en objectius parcials de dos tipus:

a) Objectius primaris:
   - Fer l'aplicació portable
   - Fer la interfície gràfica d'usuari
   - Estructura de dades
   - Xifrat de les dades
   - Auto-generació de contrasenyes robustes
   - Comprovació de la robustesa de les contrasenyes

b) Objectius secundaris:
   - Suport per a múltiples usuaris
   - Sincronisme de les dades

1.3 Metodologia de treball

S'utilitzarà el model en espiral com a mètode de desenvolupament de software realitzant aproximacions sobre l'especificació de les funcions per després realitzar el disseny, el desenvolupament i les proves. Un cop aquesta versió parcial de l'aplicació (prototip) sigui funcional tornar a realitzar una nova especificació ampliant les funcions i repetir la mateixa seqüència fins a completar la versió final.

1.3.1 Tasques a desenvolupar

Les tasques a desenvolupar són les següents:

- Especificar les funcions de l'aplicació final
- Especificar les versions intermèdies
- Especificar les tecnologies a utilitzar
- Per cada versió:
   - Dissenyar les estructures de dades
   - Dissenyar el diagrama de classes
   - Dissenyar els protocols criptogràfics
   - Implementar en Java
   - Provar el funcionament
- Realitzar les diapositives de suport a la presentació del treball
- Redactar el text de suport per la presentació
- Realitzar la presentació de l'aplicació final

[1]Graphic User Interface: Interfície gràfica per a la interacció de l'usuari amb l'aplicació

1.4 Especificacions de les tecnologies a utilitzar

La idea d'aquest treball és la de crear aquesta aplicació de forma que el seu ús i distribució sigui gratuït i no tingui cap cost per l'usuari final. És per això que per fer la implementació s'han escollit diverses tecnologies de codi lliure i/o gratuït.

Com a llenguatge de programació s'ha triat el Java i s'ha desenvolupat utilitzant l'entorn de desenvolupament integrat (IDE) *NetBeans*[2]. Aquest llenguatge de programació també dóna solució a un altre dels objectius establerts per aquest treball: la portabilitat de l'aplicació. El llenguatge de programació Java permetrà executar l'aplicació en qualsevol sistema informàtic que disposi de la màquina virtual de Java.

Per a l'emmagatzematge persistent de les dades es va pensar en un primer moment en el motor de bases de dades no relacional *MongoDB*[3]. Finalment, s'ha utilitzat com a solució d'emmagatzematge la base de dades *SQLite*[4] que permet portar en un sol fitxer la base de dades i accedir-hi des del codi amb la llibreria *sqlite4java*[5] sota llicència *Apache License 2.0*.

Altres eines utilitzades en el desenvolupament d'aquesta aplicació són *Gantt Project*[6], per a la realització de diagrames temporals, i *StarUML*[7] per al disseny de classes i diagrames.

1.5 Planificació temporal

Per tal de realitzar una planificació temporal del projecte tindré en compte una previsió de cinc prototips previs a l'aplicació final:

- Prototip 1: Crear l'aplicació que emmagatzema les dades sense xifrar.
- Prototip 2: Introduir el xifratge de les dades.
- Prototip 3: Introduir una auto-generació robusta de contrasenyes i la comprovació de la robustesa de les introduïdes pels usuaris.
- Prototip 4: Introduir suport per a múltiples usuaris.
- Prototip 5: Introduir el sincronisme de les dades.

[2]NetBeans - https://netbeans.org/
[3]MongoDB - https://www.mongodb.org/
[4]SQLite - https://sqlite.org/
[5] Sqlite4java - https://code.google.com/p/sqlite4java/
[6] Gantt Project - http://www.ganttproject.biz/
[7]StarUML - http://staruml.sourceforge.net/en/

Basant-me en aquest prototips es podria realitzar una planificació de les entregues com la següent:

PAC 2	PAC 3	PAC 4	Lliurament
- Estructura de les dades - Prototip 1	- Prototip 2 - Prototip 3	- Prototip 4 (objectiu opcional) - Prototip 5 (objectiu opcional) - **Aplicació final** - Memòria del projecte - Diapositives i text de la presentació	- Presentació del projecte

Figura 1 - Taula d'entregues

D'una manera més gràfica es pot observar aquesta planificació mitjançant un diagrama de Gantt:

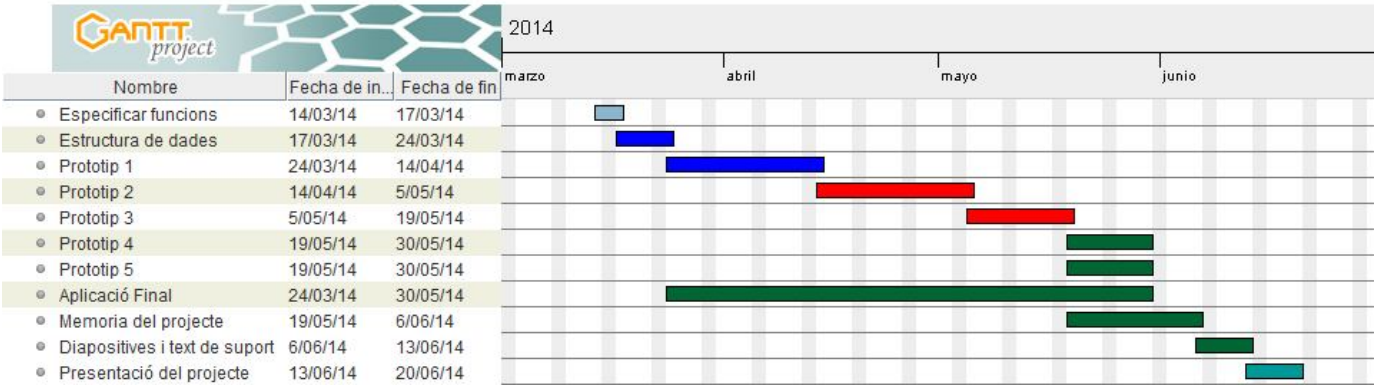

Figura 2 - Diagrama de Gantt

1.6 Estat de l'art

Amb aquest apartat es vol fer una aproximació a l'estat actual de les diferents opcions que es poden trobar al mercat que cerquen uns objectius similars i de les diferents eines criptogràfiques que s'han utilitzat en el desenvolupament d'aquest projecte per tal de garantir la privacitat de les dades.

1.6.1 Aplicacions similars

Actualment existeixen multitud d'aplicacions que permeten protegir sota una única contrasenya les diferents contrasenyes que es fan servir. De la recerca que s'ha realitzat s'ha pogut comprovar que n'hi ha que són gratuïtes, altres de pagament o bé que requereixen una quota mensual. Les més destacables són:

1.6.1.1 KeePass

És una aplicació de codi lliure, totalment gratuïta, portable i que utilitza els algoritmes AES i *Twofish* per tal de xifrar la base de dades i la funció *hash* SHA-256 per a emmagatzemar la contrasenya mestra. Entre altres característiques d'aquesta aplicació també es pot trobar un generador de contrasenyes o un sistema per copiar a la memòria la contrasenya durant un temps determinat.

1.6.1.2 PasswordSafe

Una altra aplicació de codi lliure i totalment gratuïta. Aquesta aplicació és més senzilla i no és portable però permet gestionar diversos arxius de base de dades amb la mateixa instal·lació de l'aplicació. També permet portar la base de dades a altres equips sempre i quan hi hagi la mateixa versió instal·lada en tots els equips.

1.6.1.3 LastPass

LastPass és segurament una de les aplicacions d'aquest tipus més completes on s'ofereix una solució propietària en què la descàrrega del programa és gratuïta amb característiques bàsiques i una versió *premium* per uns $24 a l'any per cada usuari. Entre les característiques bàsiques ofereix un sincronisme de les contrasenyes que gestiona entre els diferents navegadors utilitzats, l'opció de fer *login* amb un sol clic o bé l'emplenament automàtic de formularis. Amb la versió Premium s'ofereix l'accés a les dades des de diferents dispositius mòbils, sincronisme amb el núvol o l'opció de *Single Sign-On* a diverses aplicacions al núvol.

1.6.2 Comparativa de funcionalitats

Com hem vist hi ha moltes aplicacions que faciliten la gestió de les contrasenyes tot i que cada aplicació té característiques diferents i ofereix serveis diversos. Ara veurem una taula on s'estudien aquestes característiques:

	Keepass	**PasswordSafe**	**LastPass**	**PasswordVault**
Algoritme de xifrat	AES/Twofish	Twofish	AES	Twofish
Auto generador	Si	Si	Si	Si
Portable	Si	No	No	Si
Sincronisme al núvol	No	No	Si	No
Llicència	Gratuïta	Gratuïta	24 USD/any	Gratuïta

Figura 3 - Taula comparativa de funcionalitats

1.6.3 Eines criptogràfiques

Per tal de garantir la seguretat de l'aplicació, tant del seu ús i accés, com de les dades que s'han d'emmagatzemar s'han utilitzat una sèrie d'eines i algoritmes criptogràfics. Ara veurem una descripció de cadascun dels utilitzats i de l'ús que se li ha donat.

1.6.3.1 Les funcions de hash

Les funcions de *hash* són un tipus de funcions fonamentals en la criptografia moderna que el que fan és assignar a una cadena de qualsevol llargada alguna cadena d'una determinada llargada anomenada *valor de hash*. Per tal que a aquestes funcions se'ls pugui donar un ús criptogràfic és aconsellable que compleixin dues propietats. Si suposem que la nostra funció *hash* és h(x), ha de ser computacionalment inviable:

a) trobar dues cadenes amb un valor de hash comú ($h(x) = h(y)$)
b) trobar l'entrada que origina un determinat valor *hash*, tal que $h(x) = y$

1.6.3.2 Secure Hash Algorithm

Una de les característiques que s'han tingut en compte a l'hora d'escollir la funció de *hash* és la resistència a les col·lisions.

L'essència d'una funció *hash* és que per qualsevol missatge de mida *n* genera un "resum" d'una mida fixe, per exemple 512 bits en el cas de la funció SHA-512. L'altra característica bàsica és que una petita modificació a l'entrada produeixi una sortida absolutament diferent.

És evident que al tenir moltes més possibilitats a l'entrada de la funció que sortides possibles ens trobarem que més d'una entrada produirà la mateixa sortida i obtindrem una col·lisió. Les col·lisions són un punt feble d'aquest tipus de funcions ja que com més alta sigui la probabilitat de trobar-ne una, més fàcil serà que un atacant tingui èxit.

1.6.3.3 Seguretat de les funcions de hash

La seguretat d'una funció *hash* ve donada per la complexitat que tingui el més eficient dels atacs coneguts. En el cas de les funcions *hash* és l'anomenat atac de l'aniversari[8] que aconsegueix trobar col·lisions en $2^{n/2}$ intents[9]. Aquest mètode el que aconsegueix és trobar dues entrades qualsevols amb la mateixa sortida.

Tot i això, si un atacant aconseguís comprometre les nostres dades i aconseguir el resultat de la funció de *hash* de la nostra contrasenya, hauria de calcular alguna pre-imatge, és a dir, alguna entrada que tingués exactament el mateix resultat de la funció *hash*. En aquest cas serien necessaris 2^n intents[10].

---

[8] "Handbook of applied cryptography", pàg. 369
[9] "Handbook of applied cryptography", pàg. 335 – taula 9.2
[10] "Handbook of applied cryptography", pàg. 335 – taula 9.2

1.6.3.4 Bits de sal

Una altra de les tècniques que es poden utilitzar per intentar recuperar el missatge que ha generat un determinat *hash* és utilitzar les anomenades *rainbow tables*[11]. Són unes taules pre-calculades dels diferents valors de sortida d'una funció *hash* amb la seva corresponent entrada. Això fa que un eventual atacant no hagi de calcular la funció per cada entrada que vulgui comprovar sinó que simplement hagi de comparar els diferents valors de la taula amb el *hash* a corrompre i, per tant, necessiti menys temps de computació ja que és molt menys costosa l'operació de comparació que la de generació del *hash*.

Per fer inviable la utilització d'aquestes taules s'ha implementat la utilització de bits de sal a l'hora de codificar la contrasenya. És per això que s'ha dissenyat la classe *Password*. Aquesta classe és l'encarregada de generar una cadena pseudo-aleatòria criptogràficament forta i de la llargada indicada. Per tal de generar la llavor inicial s'ha utilitzat la llibreria de Java SecureRandom[12] que produeix una sortida aleatòria no determinista.

1.6.3.5 Algoritmes de xifrat de bloc

Una de les característiques que fan que un criptosistema sigui segur és que la informació que proporciona el text xifrat en relació al text en clar sigui la mínima possible. En concret, s'ha de vetllar perquè les propietats estadístiques del text en clar no es traslladin al text xifrat.

Per la implementació d'aquesta aplicació s'ha utilitzat un criptosistema de xifratge de bloc. Aquest tipus de criptosistemes es basen en què divideixen el text a xifrar en blocs de dades d'una llargada determinada. Per cada bloc s'aplicaran una sèrie de permutacions basades en una taula inicial σ, on després d'una sèrie d'iteracions es transposa mitjançant una taula final σ^{-1}.

Taula de permutacions σ							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Taula de permutacions σ^{-1}							
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Figura 4 - Exemple de taules de permutacions

[11]http://en.wikipedia.org/wiki/Rainbow_table
[12]http://docs.oracle.com/javase/7/docs/api/java/security/SecureRandom.html

En cada iteració de l'algorisme es realitzen una sèrie de substitucions i transposicions mitjançant unes claus intermèdies calculades a partir de la clau donada per xifrar. S'ha de destacar que l'algoritme escollit està basat en el mode CBC[13] que consisteix en l'encadenament dels blocs durant el xifratge i, d'aquesta manera, aconseguir una dependència entre els blocs per tal d'evitar que es puguin perdre o modificar blocs sense que es detecti.

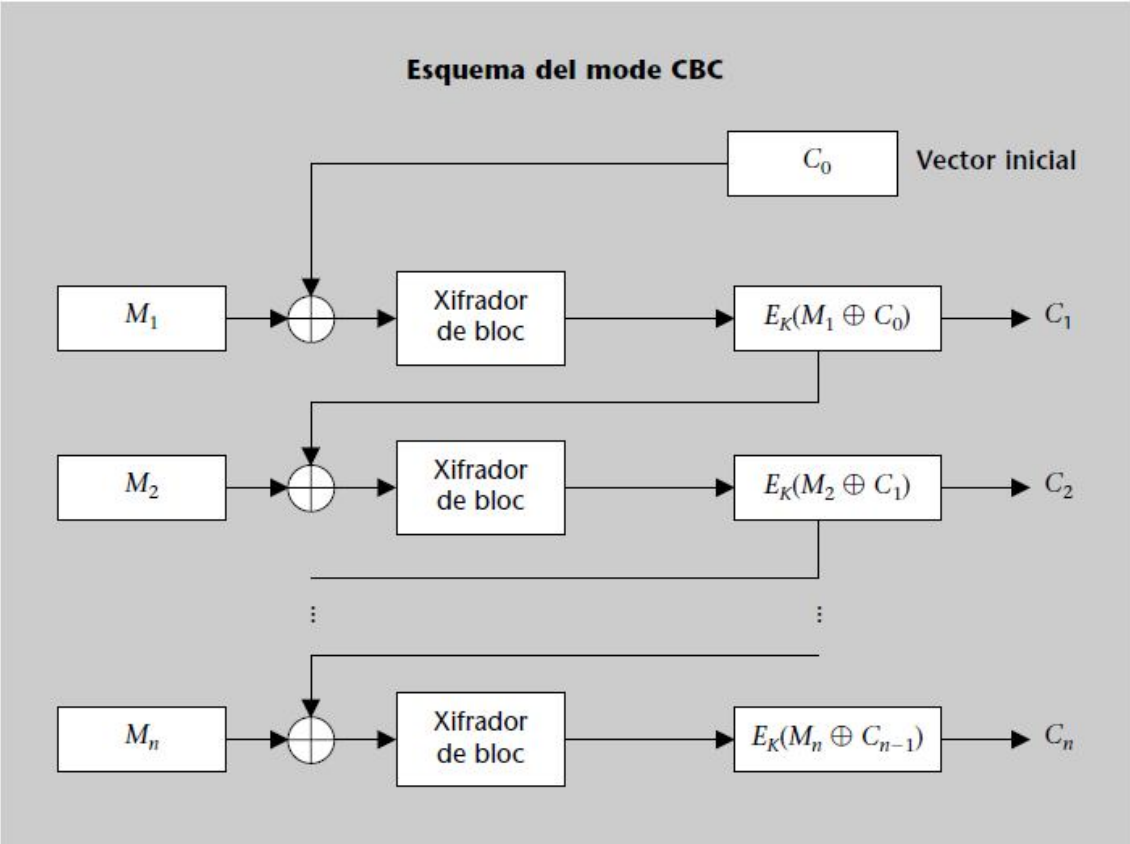

Figura 5 - Esquema general del mode CBC

[13]Cipher Block Chaining: Mode de xifratge on els blocs s'encadenen per generar una dependència amb el bloc anterior

1.6.3.6 Twofish Algorithm

Fins ara hem vist de quina forma l'aplicació garanteix que no es pugui fer un ús de l'aplicació per usuaris no autoritzats. Tot i això, pot ser possible que algun usuari malintencionat pugui arribar a corrompre la base de dades i arribar a accedir a les dades. Per tal de protegir les dades emmagatzemades i impossibilitar una lectura correcta s'ha implementat el xifratge de les dades emmagatzemades.

L'algoritme escollit per realitzar el xifratge de les dades és l'anomenat *Twofish* dissenyat per Bruce Schneier. Aquest és un algoritme de xifrat per blocs amb clau simètrica, és a dir, que divideix el missatge que es vol xifrar en blocs d'una mida determinada i que s'utilitza la mateixa clau tan per xifrar com per desxifrar.

Fins a dia d'avui, el millor criptoanàlisis publicat està basat en una variant de la tècnica del criptoanàlisis diferencial que analitza quines diferències existeixen entre dos textos. En aquesta variant només es consideren diferències en alguns bits i no en tot el bloc sencer. Utilitzant aquesta tècnica, l'any 2000 es va publicar un criptoanàlisis de les 16 rondes que formen l'algoritme que aconseguia trobar diferencials amb una probabilitat de $2^{-57.3}$per cada bloc[14].

1.7 Organització de la memòria

Amb aquest document s'ha volgut plasmar la realització d'aquest projecte començant pels motius existents per desenvolupar aquesta aplicació i les seves especificacions. Tot seguit podrem veure una descripció de les diferents eines i algoritmes criptogràfics que s'han utilitzat així com exemples d'aplicacions similars que existeixen al mercat.

En un segon capítol es farà un anàlisi dels requisits de l'aplicació i del seu disseny per passar a parlar dels protocols criptogràfics dissenyats en un tercer capítol.

Finalment trobarem dos capítols més, un que es centra a parlar de l'eficiència computacional d'alguns dels mètodes més importants i un darrer capítol amb les conclusions i les línies de treball que es poden seguir a partir de l'aplicació resultant.

[14]https://www.schneier.com/twofish-analysis-shiho.pdf

Capítol 2. Anàlisi i disseny

Dins d'aquest apartat es tractaran tots els aspectes sobre l'anàlisi i el disseny de l'aplicació i les raons que han motivat aquestes decisions.

2.1 Requisits de l'aplicació

Amb aquest apartat veurem quins són els requeriments d'ús d'aquesta aplicació i de quina forma s'han implementat. Com s'ha parlat a la introducció el nucli d'aquesta aplicació és l'emmagatzematge de les dades d'accés dels diferents comptes que té un usuari. Per tant, les funcionalitats bàsiques de l'aplicació es poden dividir en dos grups:

- Operacions sobre els usuaris
- Operacions sobre els comptes

Aquestes funcionalitats les podem veure esquematitzades en el següent diagrama de casos d'ús de l'aplicació:

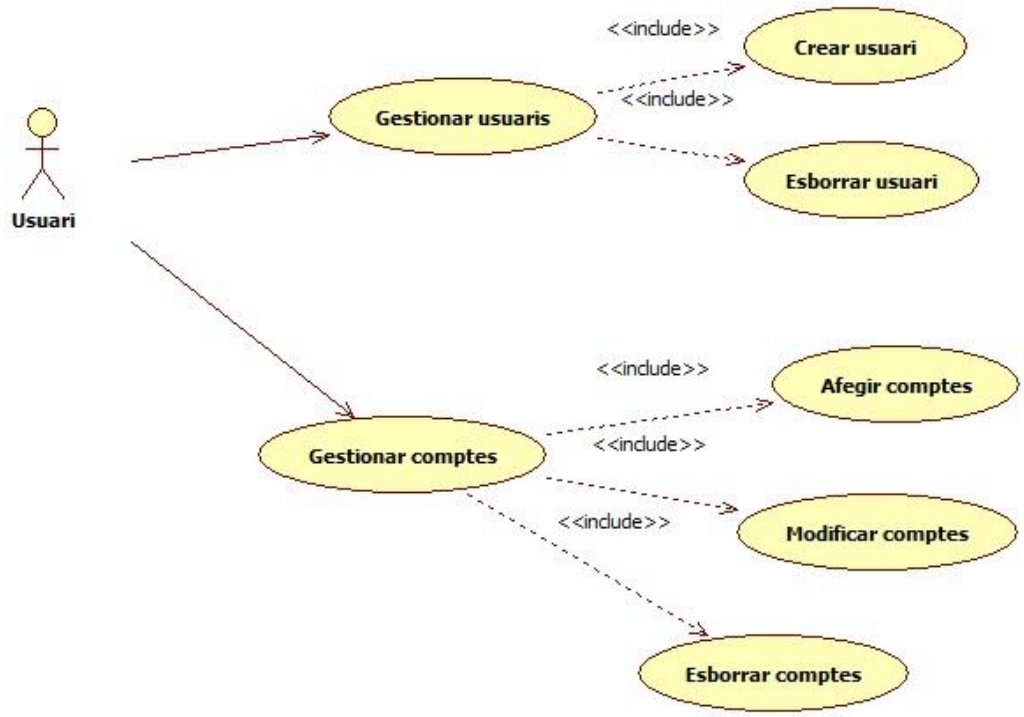

Figura 6 - Diagrama de casos d'ús

2.1.1 Operacions sobre els usuaris

L'aplicació ha de permetre una gestió correcta dels usuaris per tal de poder crear-ne de nous o esborrar un usuari existent.

2.1.1.1 Creació d'usuaris

Aquest procés és el primer amb el que s'enfrontarà l'usuari que tingui un primer contacte amb aquesta aplicació. Amb aquest pas es garanteix la creació d'una contrasenya que permetrà a l'usuari xifrar o desxifrar les seves dades i accedir a l'aplicació en els usos posteriors.

2.1.1.2 Eliminació d'usuaris

Aquest pas no es refereix només a deixar sense efecte l'usuari i que no pugui accedir més a l'aplicació sinó que també implica l'esborrat de totes les dades relacionades amb aquest usuari.

2.1.2 Operacions sobre els comptes

Cada usuari de l'aplicació anirà emmagatzemant les dades dels diferents comptes als que ha d'accedir. Per tant, l'aplicació ha de permetre la creació de nous comptes on s'emmagatzemin les dades de l'usuari i ha de garantir al propietari d'un compte el fet de poder actualitzar les dades i, fins i tot, eliminar-les.

2.1.2.1 Creació d'un compte

És el procés on l'usuari pot introduir les dades del nou compte per tal que siguin emmagatzemades a la base de dades xifrades amb la seva clau personal.

2.1.2.2 Modificació d'un compte

Amb aquest pas es pretén donar a l'usuari la possibilitat de poder canviar la contrasenya d'algun dels seus comptes i que aquest modifiqui les dades emmagatzemades a l'aplicació i es desin xifrades amb la seva pròpia clau.

2.1.2.3 Eliminació d'un compte

Amb aquest últim procés s'habilita la possibilitat que l'usuari pugui esborrar les dades de qualsevol compte de la seva propietat i que ja no vulgui mantenir.

2.2 Diagrama de classes

Aquesta aplicació s'ha dissenyat seguint els principis de la programació orientada a objectes i he decidit implementar-la utilitzant cinc classes:

- PasswordVaultGUI: representa l'aplicació i la seva interfície gràfica d'usuari.
- ConnectorDB: representa la connexió amb la base de dades.
- User: representa a l'usuari que està utilitzant l'aplicació.
- Account: representa cadascun dels comptes que l'usuari emmagatzema.
- Password: genera cadenes aleatòries

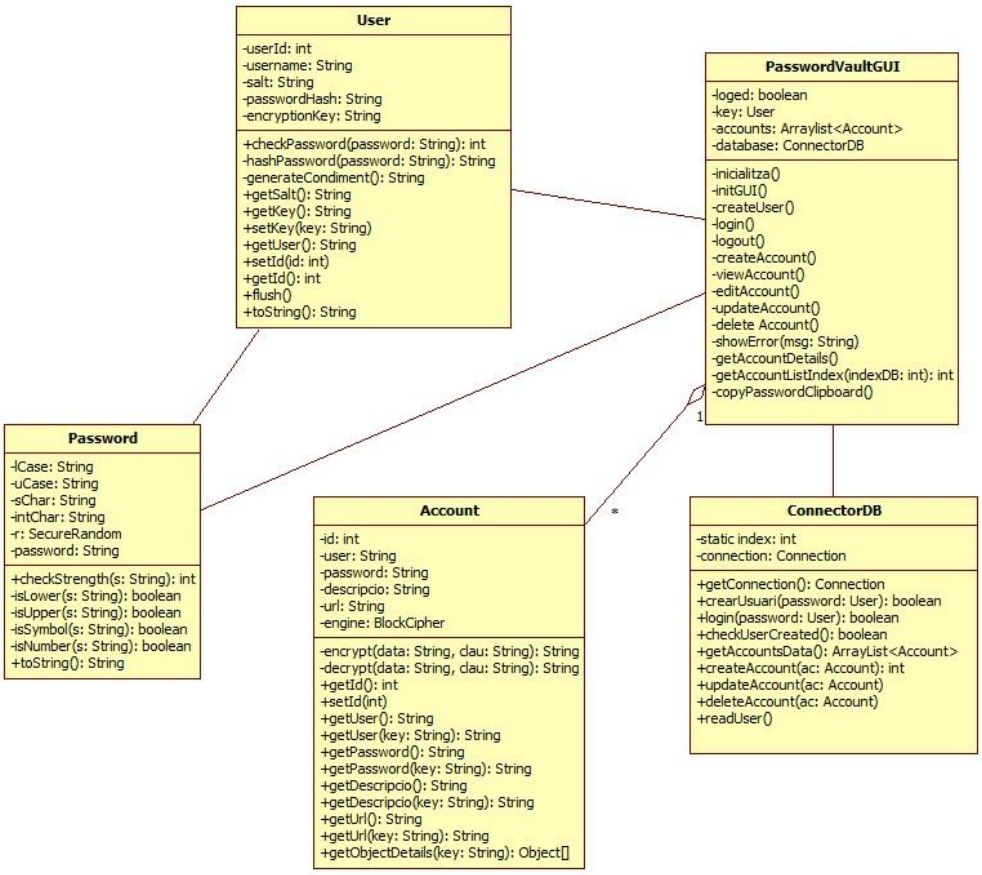

Figura 7 - Diagrama de classes

Aquesta separació de les funcions s'ha fet per tal de facilitar la reutilització del codi i l'actualització de l'aplicació. Cada classe s'ha implementat de manera que encapsuli els diversos atributs i mètodes que l'han de formar perquè la implementació d'una classe no afecti a les altres classes. Així, d'aquesta forma, es podrien realitzar canvis en la base de dades i només afectaria a la classe ConnectorDB o es podria donar suport a alguna altra funció de *hash* i només s'hauria d'actualitzar la classe User.

2.3 Disseny de la base de dades

Una de les característiques de les aplicacions d'aquesta mena és que les dades que s'emmagatzemen estiguin disponibles de forma persistent per poder-les consultar, modificar, afegir o esborrar amb posterioritat. Per fer això possible s'ha utilitzat el motor de base de dades SQLite tal com s'ha explicat al capítol 1. En aquest capítol es parlarà de les taules que s'han dissenyat i de la seva funcionalitat.

2.3.1 Taula Users

Aquesta taula contindrà les dades que identifiquen als diferents usuaris autoritzats i la seva contrasenya d'accés a l'aplicació. Aquesta contrasenya estarà xifrada amb l'algoritme SHA-512 i en aquesta taula també contindrà els bits de sal utilitzats. La taula està composada pels camps següents:

- user_id: un camp auto numèric que s'utilitzarà com a clau índex i com a identificador de l'usuari
- user: un camp de text on s'emmagatzemarà el nom de l'usuari, que serà únic
- password: un camp de text on s'emmagatzemarà la contrasenya codificada amb l'algoritme SHA-512
- salt: un camp de text on s'emmagatzemaran els bits de sal utilitzats al codificar la contrasenya amb l'algoritme SHA-512

2.3.2 Taula Accounts

És la taula on s'emmagatzemaran les dades relatives al comptes que l'usuari vagi creant. Com veurem en el capítol següent les dades que s'emmagatzemaran en aquesta taula estaran guardades de forma segura. Aquesta taula està composada pels camps següents:

- account_id: un camp auto numèric que s'utilitzarà com a clau índex i com a identificador del compte
- user: un camp de text on s'emmagatzemarà l'usuari relacionat amb aquest compte
- password: un camp de text on s'emmagatzemarà la contrasenya assignada
- descripcio: un camp de text on s'emmagatzemarà una descripció sobre el compte
- url: un camp de text on s'emmagatzemarà l'adreça a on aquest compte dóna accés
- owner: un camp de text on s'emmagatzemarà l'usuari propietari d'aquest registre. Actuarà com a clau forana basat en el camp *user_id* de la taula *Users*

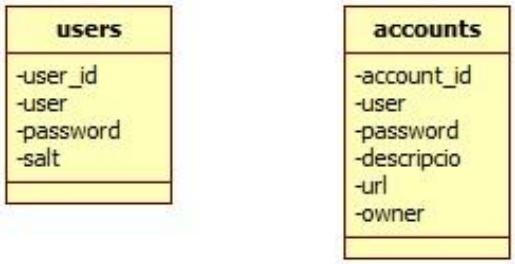

Figura 8 - Taules de la base de dades

Capítol 3. Protocols criptogràfics

Aquest tercer capítol es centra en els diferents protocols de seguretat que s'han implementat per tal que l'aplicació sigui segura. Aquesta protecció vindrà donada tant pel protocol establert en el procés d'autenticació com per l'ús d'algoritmes i tècniques criptogràfiques que dificulten la suplantació.

Aquesta voluntat de realitzar una aplicació segura porta implícit la utilització d'aquestes tècniques i algoritmes criptogràfics mitjançant algun protocol que garanteixi la seva integritat i funcionalitat. En aquest capítol es parlarà dels diferents processos clau on la seguretat és un punt important i de quin ha estat el protocol dissenyat per aconseguir-ho.

3.1 Disseny del procés de creació d'usuaris

Aquest procés és el que té lloc quan es crea un nou usuari i serà el primer procés que tindrà lloc en la primera execució. També és el responsable de garantir que no es puguin crear dos usuaris amb el mateix nom i del correcte emmagatzematge de la clau d'accés que s'utilitzarà també per xifrar les dades.

3.1.1 Algoritme criptogràfic del procés de creació d'usuaris

El procés de creació d'usuaris s'ha dissenyat tenint en compte que el procés d'autenticació es realitzarà mitjançant una contrasenya. Per aquest motiu en el disseny d'aquest procés s'ha tingut en compte l'emmagatzematge d'aquesta contrasenya d'accés.

En el moment de crear un usuari nou primer de tot es comprovarà que aquest no existeixi i llavors es procedirà a la creació del nou usuari i a l'emmagatzematge de la contrasenya. Tal i com s'ha parlat anteriorment aquesta contrasenya estarà emmagatzemada en forma de *hash*. Per tal d'evitar els atacs d'aniversari[15] el que farem és generar una cadena aleatòria que actuarà com els bits de sal que utilitzarem per calcular la funció *hash* de la contrasenya.

Aquests bits de sal estaran emmagatzemats a la base de dades per tal de poder-los fer servir més endavant en futurs processos d'autenticació.

3.1.2 Diagrama del procés de creació d'usuaris

Com s'ha dit a la introducció d'aquest treball un dels pilars de la seguretat sobre la informació que aquesta aplicació contindrà és la contrasenya que protegeix el seu accés. Per això, no només és necessari que aquesta contrasenya sigui robusta i difícil de predir sinó que el seu emmagatzematge ha de ser suficientment segur perquè un atacant no sigui capaç d'esbrinar-la.

[15] "Handbook of applied cryptography", pàg. 369

Per fer això possible s'ha implementat un protocol per l'emmagatzematge segur d'aquesta contrasenya a la base de dades tal com es pot veure a la Figura 9.

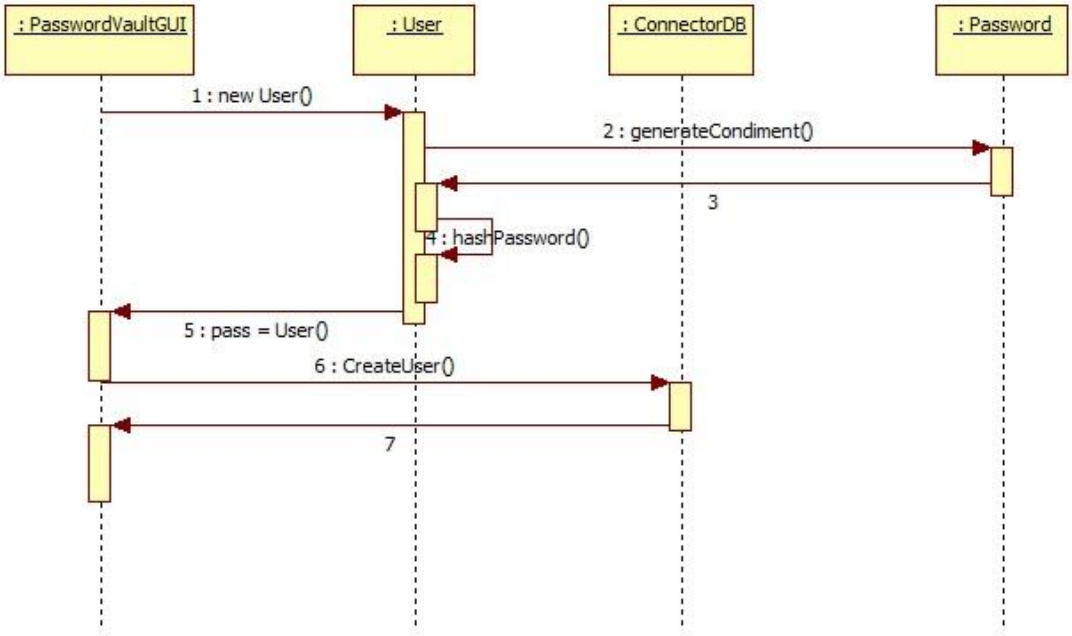

Figura 9 - Diagrama de seqüència del procés de creació d'un usuari

Seguint aquest protocol de creació d'usuari veiem que el constructor realitza una crida a una funció anomenada *generateCondiment()* que mereix una especial atenció ja que és un dels punts claus en la seguretat sobre la contrasenya. També es pot observar que és el mètode cridat just abans de generar la funció resum d'aquesta contrasenya. Això és així perquè el que s'aconsegueix amb aquest mètode és generar una cadena aleatòria que serà la que formarà els bits de sal que acompanyaran a la contrasenya en el moment de generar el *hash*. Un cop generats els bits de sal es procedeix a generar la funció resum de la contrasenya i els bits de sal mitjançant el mètode *hashPassword()*.

3.2 Disseny del procés d'autenticació

Aquest procés té lloc cada vegada que un usuari vol accedir a l'aplicació i serà l'encarregat de verificar si la contrasenya escrita per l'usuari és correcta i es pot garantir l'accés a l'aplicació.

3.2.1 Algoritme criptogràfic del procés d'autenticació

El procés d'autenticació de l'usuari s'ha implementat mitjançant l'accés per contrasenya. Per evitar que aquesta contrasenya pugui ser descoberta per usuaris malintencionats està emmagatzemada a la base de dades utilitzant una funció de resum, en concret, utilitzant l'algoritme *Secure Hash Algorithm* en la seva segona versió (SHA-512).

Per poder autenticar l'usuari i permetre l'accés a l'aplicació es segueix el protocol descrit en la Figura 10 i que es basa en els passos següents:

- Recuperar els bits de sal utilitzats en la creació
- Calcular la funció resum de la contrasenya a validar.
- Sol·licitar a la base de dades la funció resum correcta.
- Comparar els dos resultats de la funció resum.

En el cas que la funció resum sobre la contrasenya a validar ens doni el mateix resultat que l'emmagatzemat voldrà dir que la contrasenya és correcta i es pot garantir l'accés.

3.2.2 Diagrama del procés d'autenticació

Així com el procés de creació d'un usuari és un dels processos més importants per tal de garantir la integritat de les dades, el procés d'autenticació no és menys important. Per això s'ha dissenyat conjuntament amb el de creació d'usuaris per tal d'evitar la suplantació de l'usuari.

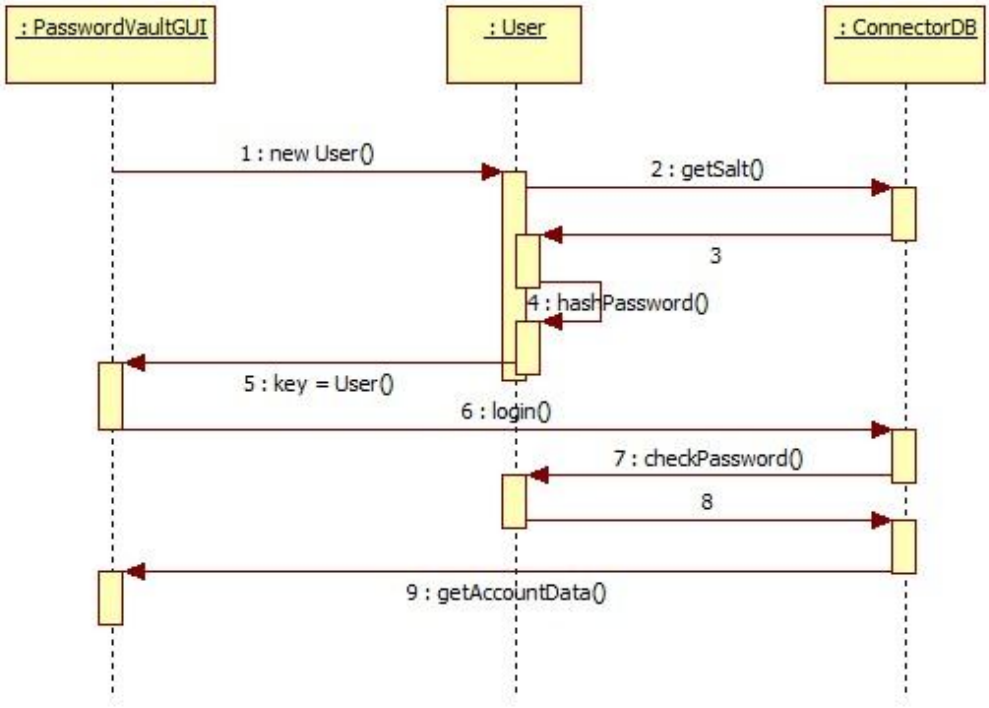

Figura 10 - Diagrama de seqüència del procés d'autenticació

El protocol d'autenticació de l'usuari és lleugerament diferent al de creació ja que en aquest cas no és necessari generar els bits de sal sinó que es fa necessari recuperar els utilitzats en el moment de la creació. És per aquest motiu que al generar un nou objecte de la classe *user* (*new User()*) es crida a un constructor diferent que en el cas anterior. Abans de generar el *hash* de la contrasenya introduïda es realitzarà una crida al mètode *getSalt()* per tal de recuperar de la base de dades els bits de sal que es van utilitzar en la creació de l'usuari, poder calcular el *hash* correctament i poder validar si el procés d'autenticació es realitza amb èxit o no.

Aquesta validació es realitzarà a la classe *User* mitjançant la informació de la funció resum de la contrasenya correcta emmagatzemades a la base de dades i el nou *hash* obtingut amb la contrasenya a validar i els bits de sal corresponents.

3.3 Disseny del procés de xifrat

Aquest procés és un dels punts cabdals per tal de garantir la integritat de les dades emmagatzemades d'un usuari. Aquest procés serà l'encarregat del correcte xifrat de les dades mitjançant la contrasenya de l'usuari i de garantir que aquestes dades es mantinguin xifrades en tot moment.

3.3.1 Algoritme criptogràfic del procés de xifrat

Tal i com s'ha parlat en el capítol introductori s'ha de tenir en compte que en el cas que algun atacant arribés a tenir un accés il·lícit a les dades, aquestes estiguin correctament protegides per tal d'impossibilitar la seva lectura i el seu ús. També s'ha de tenir en compte que per la naturalesa de l'aplicació els usuaris tindran emmagatzemades dades sensibles com són les seves claus d'accés a diferents serveis.

En base a aquestes dues premisses s'ha dissenyat l'aplicació per tal que les dades sensibles estiguin emmagatzemades a la base de dades xifrades utilitzant l'algoritme *Twofish*. Per garantir la integritat de les dades s'ha dissenyat aquesta part de la lògica de l'aplicació de manera que les dades només siguin desxifrades en el moment en que s'hagin de presentar a l'usuari.

Així doncs, l'aplicació només treballarà amb les dades desxifrades de les descripcions dels comptes ja que són necessàries per poder presentar el llistat amb tots els comptes que l'usuari disposa. La resta de dades que componen un compte (com són la contrasenya, el nom d'usuari o l'adreça del lloc a on donen accés) només seran desxifrades en el moment en que aquell compte en concret sigui consultat.

3.3.2 Diagrama del procés de xifrat

Un dels punts forts en la seguretat d'aquesta aplicació és el fet que les dades estiguin xifrades per impedir que qualsevol persona no autoritzada hi tingui accés. Per aconseguir aquest fet s'ha dissenyat el protocol que es pot veure a la següent figura on es garanteix aquest fet.

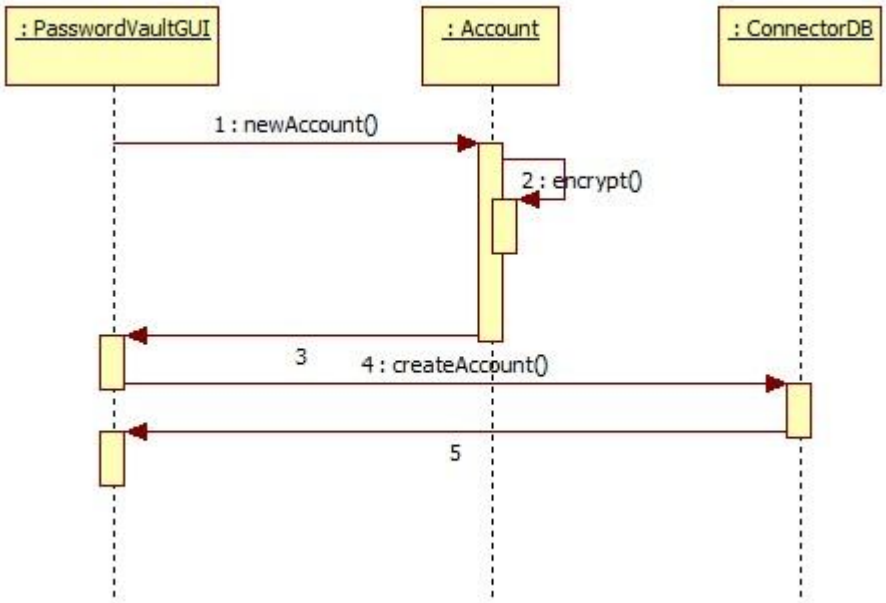

Figura 11 - Diagrama de seqüència del procés de xifrat

Com es pot comprovar en aquest diagrama el primer que es fa al generar un compte nou és xifrar les dades, és a dir, és el mateix mètode constructor de l'objecte l'encarregat del xifratge de les dades utilitzant la contrasenya de l'usuari. Un cop xifrades s'envien cap a la base de dades i es desaran de forma segura.

3.4 Disseny del procés de desxifrat

Aquest procés serà l'encarregat de permetre que un usuari legítim pugui accedir a les seves dades i poder-les visualitzar desxifrades correctament. S'ha dissenyat de tal forma que les dades estiguin en tot moment xifrades i que només es desxifrin en el moment en què sigui necessari per poder-les mostrar a l'usuari.

3.4.1 Algoritme criptogràfic del procés de desxifrat

En aquest cas el que es fa és recuperar les dades xifrades que estan emmagatzemades a la base de dades. Per tal de protegir al màxim aquestes dades es mantindran xifrades fins al moment en que es vulgui consultar les dades d'un compte en concret. En aquest moment serà quan es procedirà al desxifrat de les dades d'aquest compte per poder-les mostrar a l'usuari.

3.4.2 Diagrama del procés de desxifrat

L'altre punt cabdal en la seguretat de les dades és el fet de permetre l'accés a elles per l'usuari legítim i que les pugui llegir sense problemes. Per permetre aquest accés autoritzat s'ha implementat aquest protocol que, sense perjudicar a la seguretat proporcionada pel xifrat, permet que l'usuari legítim hi pugui accedir.

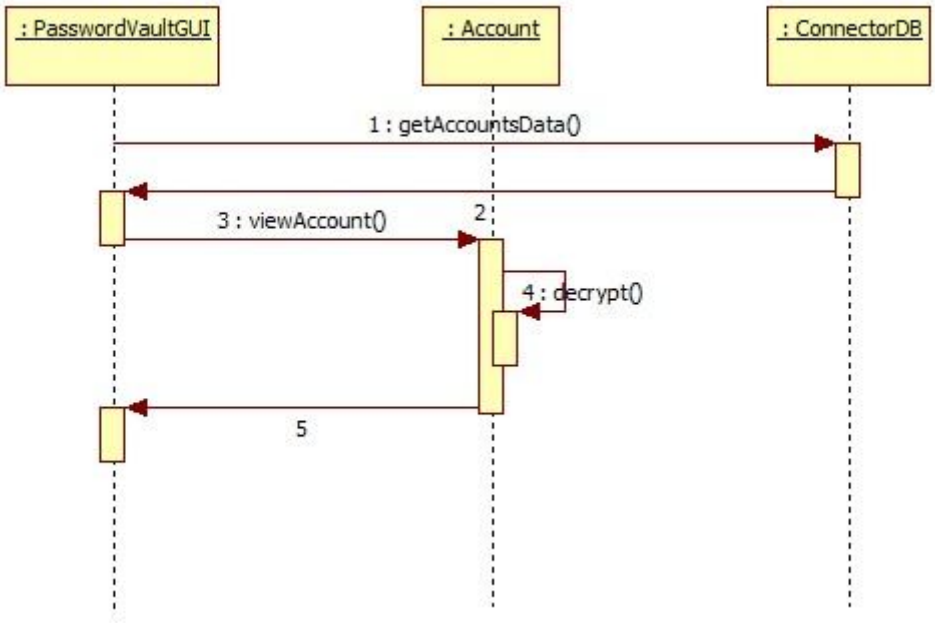

Figura 12 - Diagrama de seqüència del procés de desxifrat

En aquest cas el que es pretén és que les dades es mantinguin xifrades fins al moment que l'usuari hi vulgui accedir. És per aquest motiu que les dades es recuperen xifrades de la base de dades mitjançant el mètode *getAccountsData()* i fins el moment en que es vol consultar un compte no es procedeix al desxifrat d'aquestes dades en concret. Serà just en el moment que mitjançant el mètode *viewAccount()* el mateix objecte *Account* procedirà a desxifrar les dades utilitzant la contrasenya de l'usuari i mostrar-les en clar.

3.5 Protecció contra col·lisions

Al capítol 1 s'ha parlat de la possibilitat que tenen les funcions resum de produir col·lisions, és a dir, que un atacant pogués trobar alguna seqüència diferent de la contrasenya escollida que tingués el mateix valor de la funció *hash*. En aquest cas, tot i que poc probable[16], l'aplicació donaria per vàlida aquesta contrasenya no vàlida ja que tindria el mateix valor de la funció *hash* que la contrasenya correcta.

Per garantir la integritat de les dades en aquests casos s'ha implementat el protocol de xifrat de forma que la clau utilitzada per l'algoritme sigui la contrasenya escollida per l'usuari. D'aquesta forma, en cas que un atacant aconseguís trobar una col·lisió, veuria garantit el seu accés a l'aplicació però no podria llegir les dades emmagatzemades perquè el procés de desxifrat no es realitzaria correctament.

3.6 Biblioteca criptogràfica utilitzada

Per implementar els algoritmes criptogràfics s'ha utilitzat una biblioteca de tercers desenvolupada per *The Legion of the Bouncy Castle*[17] que implementa diferents algoritmes per al llenguatge de programació Java.

He triat aquesta biblioteca ja que està publicada sota llicència GPL, és de codi lliure i així segueix la filosofia de creació de *PasswordVault* per tal que sigui una aplicació gratuïta.

[16] "Handbook of applied cryptography", pàg. 335 – taula 9.2
[17] The Legion of the Bouncy Castle - https://www.bouncycastle.org/

Capítol 5. Eficiència

Un dels aspectes importants en la implementació dels algoritmes és l'anomenada eficiència computacional o eficiència. Aquesta eficiència es refereix a la quantitat de temps que un ordinador necessita per realitzar una tasca amb un algoritme determinat.

En aquest anàlisi teòric dels algoritmes, l'eficiència és una característica que representa la complexitat de l'algoritme estudiat amb una notació asimptòtica, és a dir, com el límit superior on es situaria una execució en el pitjor dels casos.

Dins l'aplicació *PasswordVault* s'ha implementat un mètode que realitza una estimació de la robustesa de la contrasenya mentre l'usuari l'escriu. Per tant, és un mètode força utilitzat que s'executa cada vegada que es detecta que l'usuari ha realitzat algun canvi a la contrasenya. Per aquest motiu, si aquest mètode tingués una complexitat molt elevada podria fer que l'ús de l'aplicació arribés a incomodar l'usuari per la lentitud i provocar una sensació de mal funcionament.

Veurem ara aquest algoritme i l'estudi de la seva complexitat ja que al ser un mètode molt executat s'ha d'implementar el més eficient possible.

```
if (s.length() > 9) {
    strength = 20;
}

while ((!lower || !upper || !symbol || !number) && (i < s.length())) {
     if (isLower(s.substring(i, i+1)) && !lower) {
         lower = true;
         strength += 20;
     }
     if (isUpper(s.substring(i, i+1)) && !upper) {
         upper = true;
         strength += 20;
     }
     if (isSymbol(s.substring(i, i+1)) && !symbol) {
         symbol = true;
         strength += 20;
     }
     if (isNumber(s.substring(i, i+1)) && !number) {
         number = true;
         strength += 20;
     }
     i++;
}
```

Figura 13–Implementació rudimentària de l'algoritme checkStrength()

Com es pot comprovar amb aquesta porció del codi, el mètode per comprovar la robustesa d'una contrasenya va avançant caràcter a caràcter fins a comprovar totes les característiques. El seu cas pitjor seria que no es comprovés fins a l'últim caràcter i, per tant, hauria fet *n* iteracions. És a dir, és un algoritme de complexitat O(n).

Tot i que la complexitat resultant d'aquest algoritme és acceptable també s'ha de considerar si el resultat obtingut és el desitjable. Per això s'ha realitzat una taula comparativa de diferents sortides per poder fer aquest anàlisi:

Contrasenya a avaluar	**Robustesa (0-100)**
tauleta	20
tauletetes	40
Tauletetes	60
Tau1etetes	80
Tau1etete$	100
1aB*	80
pxoedkcnftqalzwitrgsmkyhvj	40

Figura 14 - Comparativa robustesa amb la implementació rudimentària

Com es pot comprovar amb la taula anterior aquesta implementació té un inconvenient molt important al basar-se amb els caràcters per separat: pot arribar a assignar una robustesa acceptable a una contrasenya molt feble (*1aB**) i una robustesa més feble a una contrasenya realment més robusta (*pxoedkcnftqalzwitrgsmkyhvj*).

És per aquest motiu que per realitzar aquestes estimacions s'utilitza el concepte d'entropia que es veurà tot seguit.

5.1 Entropia

Dins del camp de la teoria de la informació amb el concepte d'entropia es mesura la incertesa d'una variable aleatòria. Aquesta definició està basada a l'entropia de Shannon que mesura la quantitat esperada d'informació que un missatge conté.

Si suposem que X és una variable aleatòria que pot prendre un nombre finit de valors (en el cas que ens ocupa seria el conjunt de contrasenyes possibles), tenim que l'entropia és una mesura de l'aproximació de la mitjana de bits necessaris per poder codificar els elements d'X.[18]

Per fer el càlcul d'aquesta entropia utilitzarem la fórmula següent:

$$H = L * \left(\frac{\log N}{\log 2}\right)$$

On L és la llargada de la contrasenya i N és el nombre estimat de possibles opcions per a cada caràcter.

[18] "Handbook of applied cryptography", pàg. 56

Ara veurem l'algoritme checkStrength() implementat amb aquestes noves consideracions i la seva complexitat:

```
while ((!lower || !upper || !symbol || !number) && (i < s.length())) {
      if (isLower(s.substring(i, i+1)) && !lower) {
          lower = true;
          averageSymbols += 26;
      }
      if (isUpper(s.substring(i, i+1)) && !upper) {
          upper = true;
          averageSymbols += 26;
      }
      if (isSymbol(s.substring(i, i+1)) && !symbol) {
          symbol = true;
          averageSymbols += 8;
      }
      if (isNumber(s.substring(i, i+1)) && !number) {
          number = true;
          averageSymbols += 10;
      }
      i++;
  }

double strength = s.length() * (Math.log(averageSymbols)/Math.log(2));
```

Figura 15 - Algoritme del càlcul de l'entropia d'una contrasenya

En aquest cas el que es fa és calcular l'entropia (H) que fa essencial el càlcul d'N, és a dir, el nombre estimat de possibles opcions per a cada caràcter. Per aquest motiu el que realitza aquest algoritme és anar avançant caràcter a caràcter fins a comprovar totes les característiques i anar actualitzant el nombre de possibles opcions per a cada caràcter. El seu cas pitjor seria que no es comprovés fins a l'últim caràcter i, per tant, hauria fet *n* iteracions. És a dir, és un algoritme de complexitat O(n).

A la taula següent s'exposa l'estudi de les diferents contrasenyes i la seva entropia en comparació amb l'execució de la implementació més rudimentària:

Contrasenya a avaluar	**Entropia (en bits) H=L*(log N / log 2)**	**Robustesa (0-100)[19]**
tauleta	33	20
tauletetes	47	40
Tauletetes	57	60
Tau1etetes	60	80
Tau1etete$	61	100
1aB*	25	80
pxoedkcnftqalzwitrgsmkyhvj	122	40

Figura 16 - Comparativa de l'entropia de diferents contrasenyes

Com es pot comprovar el càlcul de l'entropia d'una variable és molt més acurat i no provoca les disfuncions que la implementació més rudimentària generava. Com es veu a la taula de la Figura 16 es pot assignar un càlcul de la robustesa més real ja que assigna una entropia baixa a una contrasenya feble (*1aB**) i una entropia més alta a una contrasenya més robusta (*pxoedkcnftqalzwitrgsmkyhvj*).

[19] Amb la implementació rudimentària de l'algoritme *checkStrength()*

Capítol 6. Conclusions

Si analitzem els objectius proposats a l'inici d'aquest treball podem veure que es van dividir en dos grups: objectius primaris i objectius secundaris.

De la realització d'aquest projecte de final de carrera es pot concloure que s'ha aconseguit un nivell alt en l'assoliment dels objectius proposats inicialment. Pel que fa als objectius primaris s'han aconseguit implementar tots i cadascun d'ells proporcionant una funcionalitat total a l'aplicació final.

En el cas dels objectius secundaris s'ha aconseguit realitzar un dels dos objectius proposats, el que fa referència al suport multiusuari, és a dir, que l'aplicació pugui ser utilitzada per varis usuaris.

Pel que fa al segon dels objectius secundaris (sincronització al núvol), tot i no poder-se implementar, s'ha afegit un apartat on es proposa un escenari i un possible disseny de la solució que es podria utilitzar per tal d'implementar-lo.

A continuació es proposen dues línies de treball sobre aquest projecte que es podrien seguir per tal de continuar amb el seu desenvolupament i afegir noves funcionalitats.

6.1 Millora del procés d'autenticació

Una línia de treball que es podria seguir recau sobre el procés d'autenticació. Actualment es realitza mitjançant l'accés amb contrasenya que, tot i utilitzar els bits de sal per tal d'evitar l'ús de *Rainbow tables*, té l'inconvenient que ha d'estar emmagatzemada d'alguna forma dins la base de dades.

Aquest procés es podria implementar mitjançant la generació d'un parell de claus (una de pública i una de privada) per cada usuari. D'aquesta forma la clau pública és la que estaria emmagatzemada a la base de dades i la clau privada només estaria en possessió de l'usuari. Així, per tal de realitzar el procés d'autenticació l'usuari podria enviar un repte xifrat amb la clau privada de forma que l'aplicació pogués comprovar que qui vol accedir és l'usuari legítim utilitzant la clau pública d'aquest usuari. Un cop autenticat es podria utilitzar la clau privada de l'usuari per xifrar les dades.

D'aquesta forma, en cas que un atac exitós aconseguís obtenir la informació de la base de dades, només obtindria la clau pública dels usuaris que no li permetria ni autenticar-se ni desxifrar les dades emmagatzemades.

6.2 Sincronització de les dades al núvol

Una altra línia de treball possible és la d'oferir la possibilitat de tenir les dades emmagatzemades al núvol i permetre la sincronització. La millor forma seria tindre les dades replicades en local i al núvol per, d'aquesta manera, poder consultar alguna dada encara que no es tingui accés a internet. Llavors, en els moments en què si que hi hagués connexió a internet, poder fer la sincronització entre la còpia local i el núvol. Per fer-ho, una opció seria afegir dos camps de text a la taula *Accounts* que haurien d'emmagatzemar la data i la hora de la darrera modificació, d'aquesta forma es podria saber quina és la versió més nova de les dades i actualitzar la més vella.

El procés seria simplement recuperar del núvol les dades d'aquell usuari, comparar data i hora dels registres existents i actualitzar el més vell. En el cas dels registres creats de nou s'haurien d'afegir a la taula on no existissin. El fet de saber si el registre és nou vindria donat perquè en una de les repliques[20] no existeix i en la que existeix no hi ha data ni hora de sincronització ja que encara no s'ha pogut realitzar mai.

En el cas de la gestió dels registres que s'han eliminat seria necessari incloure a la taula *Accounts* un camp de tipus booleà que servís per indicar que s'ha eliminat però no ha pogut ser sincronitzat i un cop sincronitzat es pot procedir a la seva eliminació.

Una possible implementació seria utilitzar l'API[21] de *Google Drive* per tal que es pogués utilitzar aquest servei d'emmagatzematge per anar guardant la versió més recent de la base de dades al núvol i així poder-la recuperar des de qualsevol lloc.

[20] O bé a la còpia local o al núvol.

[21] Application Programming Interface: Mètodes que proporcionen algunes aplicacions perquè els desenvolupadors puguin interaccionar amb elles.

Annex A – Manual d'usuari

Instal·lació

Un dels objectius d'aquesta aplicació era que fos portable per facilitar el seu ús en diferents dispositius. Per això no és necessària cap instal·lació i l'únic requisit és dur els fitxers que componen l'aplicació per tal de poder-los utilitzar en els diferents dispositius.

Inici de l'aplicació

El primer cop que s'inicia l'aplicació ens demanarà que creem un usuari. Amb la contrasenya d'aquest usuari serà amb la que es protegirà l'accés no desitjat a les dades. Un cop creat l'usuari en donarà accés directament.

En les execucions següents per defecte ens demanarà que ens autentiquem per poder accedir a l'aplicació tot i que també es permet crear més usuaris.

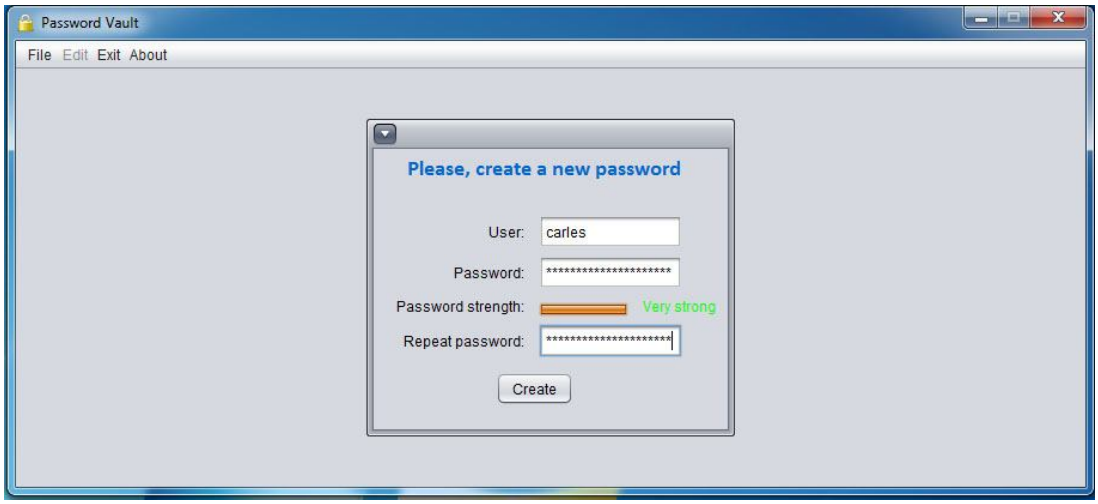

Figura 17 - Interfície de creació d'usuari

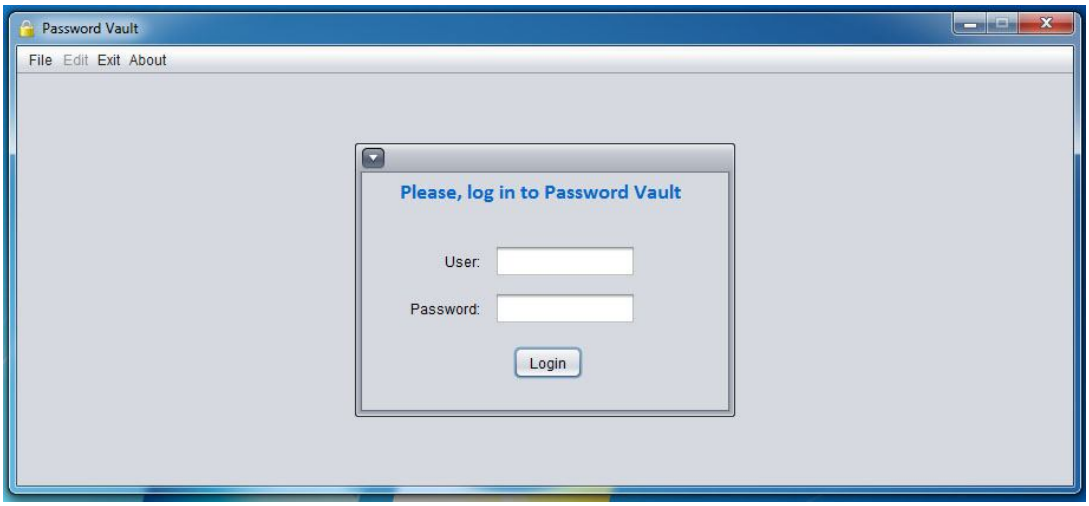

Figura 18 - Interfície del procés d'autenticació

Interfície principal

Aquesta és la pantalla principal de l'aplicació i la que ens permet realitzar totes les accions. Està dividida en dues parts:

1. Llista amb la descripció dels diferents comptes que té l'usuari
2. Detalls del compte seleccionat.

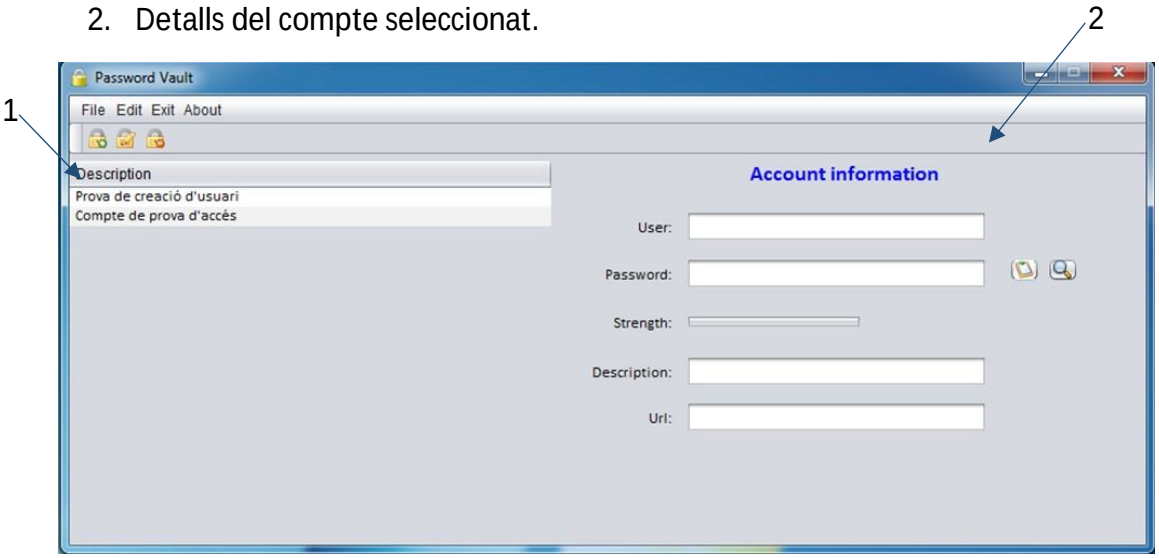

Figura 19 - Interfície principal

Operacions amb usuaris

Les operacions bàsiques que es poden realitzar amb els usuaris són les de crear un usuari nou i eliminar un usuari. Aquestes operacions són accessibles des del menú *File* que també ens permetrà desconnectar-nos per poder canviar d'usuari.

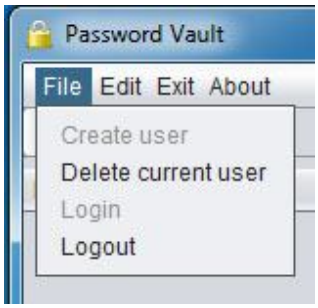

Figura 20 - Detall del menú File

Crear Usuari

La opció de crear un usuari només està activa quan no estem autenticats a l'aplicació. Aquesta operació serà la mateixa que té lloc al iniciar l'aplicació per primer cop, ens demanarà un nom d'usuari, una contrasenya i en el cas que l'usuari no existeixi el crearà i ens donarà accés.

Eliminar Usuari

Aquesta operació només està activa quan estem autenticats a l'aplicació. El que permet és eliminar l'usuari i totes les dades associades, per tant, eliminarà tots els comptes que tingui vinculats amb l'usuari actiu.

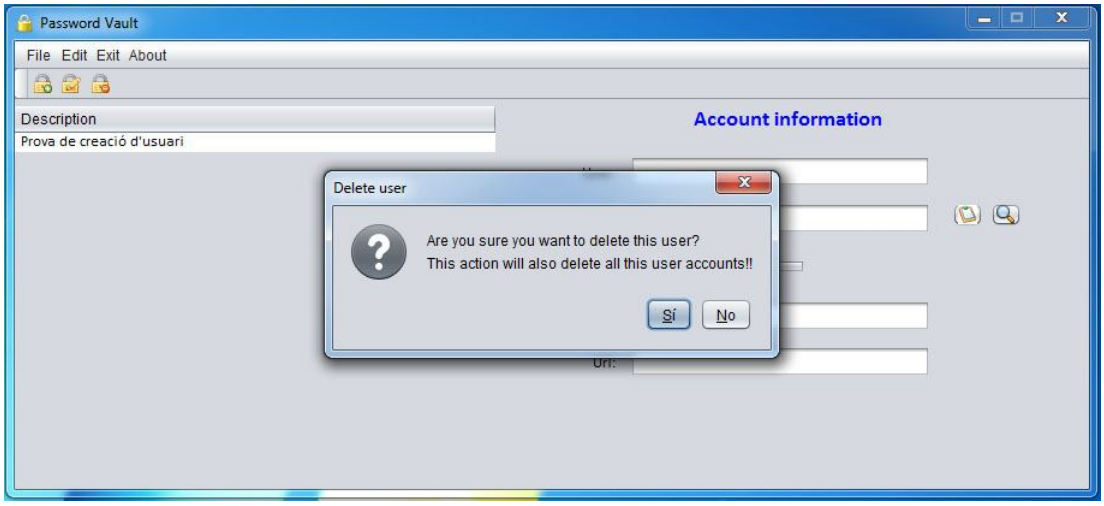

Figura 21 – Interfície d'eliminació d'usuari

Operacions amb comptes

Les tres operacions principals que es poden fer són les de crear, modificar i eliminar els diferents comptes. Aquestes accions són accessibles tant des dels botons principals com des del menú *Edit*.

Figura 22 - Detall dels botons

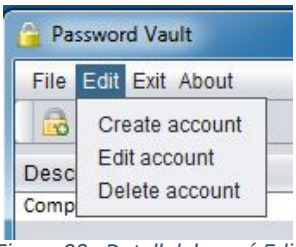

Figura 23 - Detall del menú Edit

L'altra operació que permet l'aplicació és la de consultar les dades dels comptes que té creats un usuari. Simplement fent un clic a la descripció del compte que es vol consultar es podran veure la resta de dades a la part dels detalls.

Operació Crear

En realitzar l'operació de crear un nou compte, l'aplicació ens ofereix la part dels detalls del compte en disposició per poder introduir les dades del compte que volem crear. Un cop fet podrem clicar el botó de nou per emmagatzemar les noves dades a la base de dades.

També se'ns ofereix l'opció de cancel·lar la creació del compte.

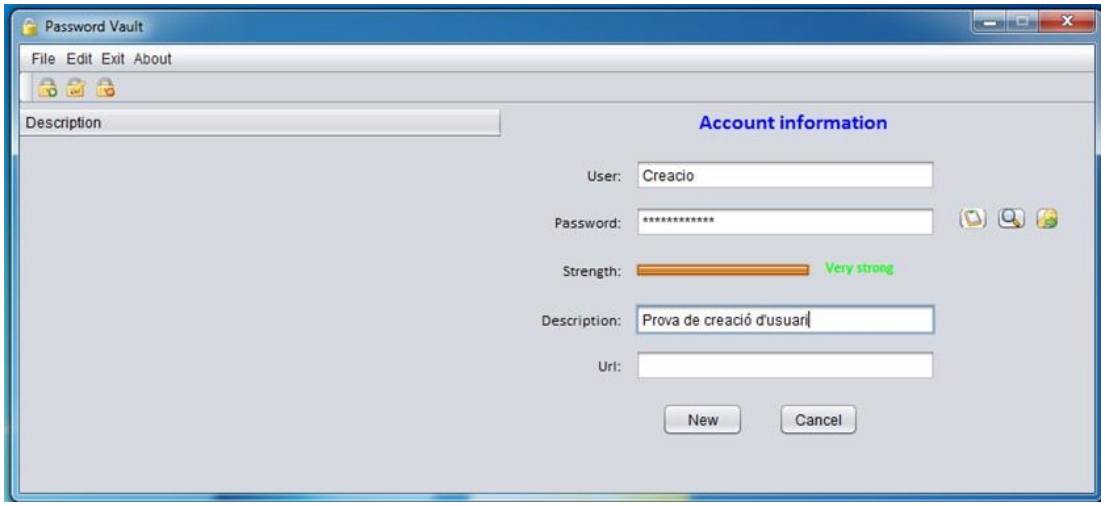

Figura 24 - Creació d'un compte

Operació Modificar

Aquesta operació realitza un comportament similar a la de consultar. En aquest cas, però, permetrà modificar les dades del compte consultat i guardar les modificacions al clicar el botó d'actualitzar.

De la mateixa forma que l'operació anterior també és possible cancel·lar la modificació d'un registre.

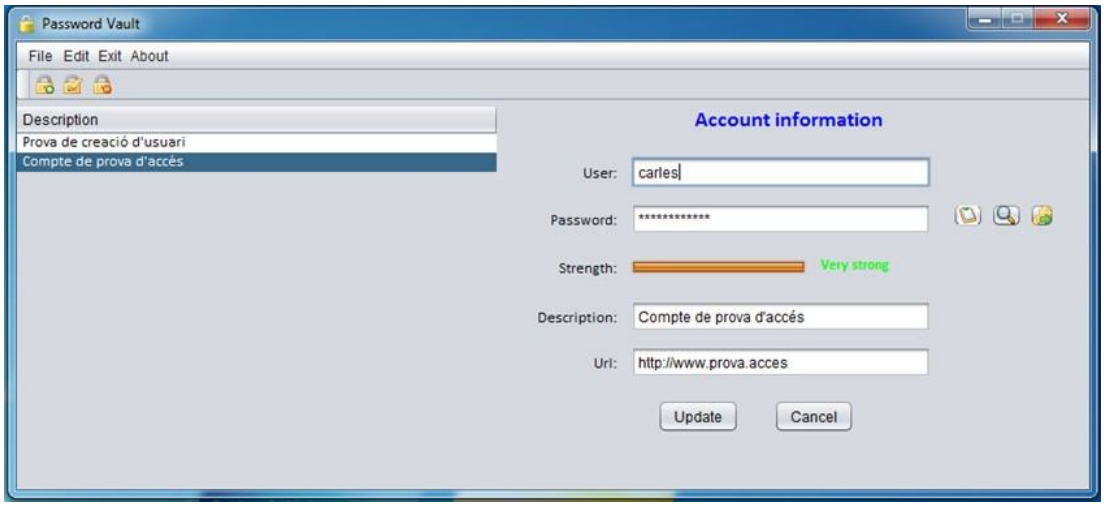

Figura 25 - Modificació d'un compte

Operació Eliminar

Aquesta operació serveix per eliminar el compte que es tingui seleccionat. Com que és una operació que no es pot desfer l'aplicació sol·licitarà una confirmació.

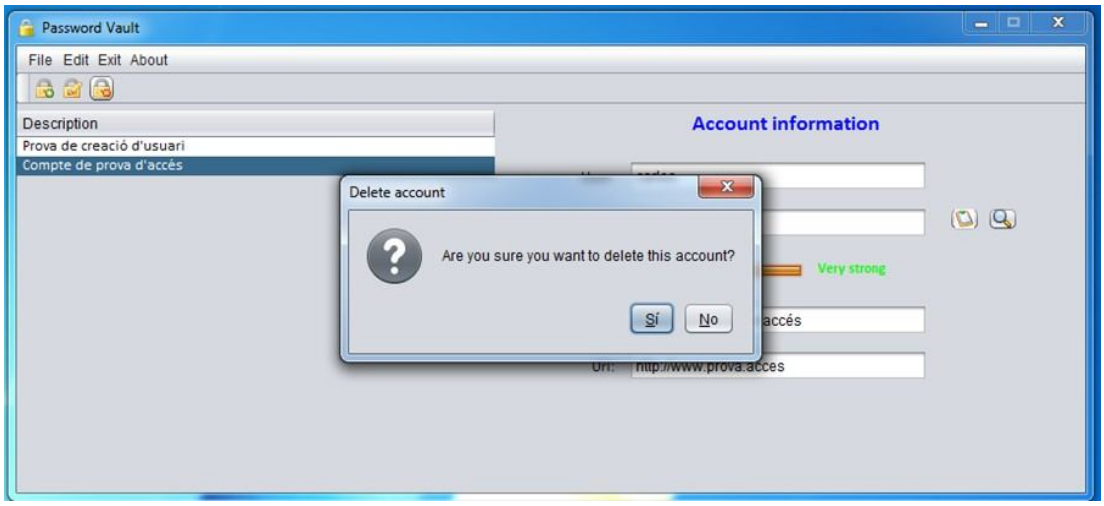

Figura 26 - Eliminació d'un compte

Altres funcions

Per tal de facilitar el seu ús, també ofereix la possibilitat de poder veure la contrasenya corresponent a cada compte (en lloc de veure-ho representat amb asteriscos), la de copiar aquesta contrasenya directament al porta papers i l'auto-generació d'una contrasenya robusta.

L'accés a aquestes funcions es realitza mitjançant els botons que hi ha al costat de la contrasenya a la zona dels detalls del compte.

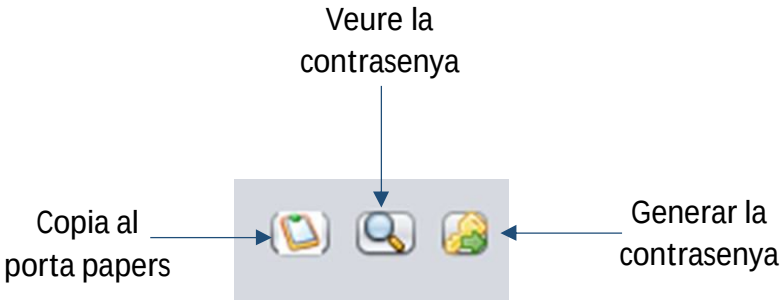

Figura 27–Altres funcions

Annex B – Bibliografia

Java Platform, Standard Edition 7 API Specification - http://docs.oracle.com/javase/7/docs/api/overview-summary.html

TheLegion of theBouncyCastle - https://www.bouncycastle.org/

SQLiteDocumentation - http://www.sqlite.org/docs.html

Menezes, van Oorschot&Vanstone. 1997. Handbook of appliedcryptography. Editorial CRC Press.

Josep Domingo Ferrer, Jordi Herrera Joancomartí i Helena Rifà Pous. 2006. Criptografia. Universitat Oberta de Catalunya.

Bates, Bert& Sierra, Kathy. 2005. HeadFirst Java. Editorial O'Reilly.

McLaughlin, Pollice&West. 2007. HeadFirstObject-OrientedAnalysisandDesign. Editorial O'Reilly.

Annex C – Glossari

Bits de sal: bits aleatoris que s'encadenen al text del que es vol calcular una funció de resum per tal d'evitar l'èxit d'atacs mitjançant taules pre-calculades.

Cipher Block Chaining (CBC): Mode de xifratge on els blocs s'encadenen per generar una dependència amb el bloc anterior

Compte: representació de les dades necessàries que descriuen els serveis als quals l'usuari.

Entropia: mesura que representa la incertesa de valors que pot prendre una cadena aleatòria.

Funció resum: funcions fonamentals en la criptografia moderna que el que fan és assignar a una cadena de qualsevol llargada alguna cadena d'una determinada llargada.

Hash: veure funció resum.

Rainbow tables: taules que contenen parelles de paraules i el seu valor hash pre-calculat.