Документация
Проект “Password manager”

Мария Дренчева, 1 курс, Специалност “Информатика”

June 2025

Линк към github на проекта: `https://github.com/MariaD-BG/Password_manager`

Увод и съдържание

Проектът е система за съхранение, криптиране, декриптиране, обновяване и изтриване на пароли.

Основните части от зададената задача са две: работа с шифриращите алгоритми и работа с файловете и командите за боравене с криптираните пароли. Макар да са свързани, ще анализираме двете поотделно в двете основни части на тази документация. В първата част са описани алгоритмите за шифриране, минимални промени спрямо подаденото условие в случаи, в които в него е имало неточности или двусмислия, и обяснение на направените избори при осъществяване на класовете, свързани с шифрите. Във втората част е описана избраната архитектура за приемане и извършване на командите, начинът на записване на паролите във файловете и тяхното съхранение в паметта.

По време на писане на кода сме се стремяли при възникване на изкючение заради неточно поведение на потребителя да запазим работата на програмата и да намерим решение, което не я прекратява. Повечето взети решения са описани тук.

В Appendix има списък на всички използвани класове и структури с кратко описание на всяка от тях.

Част I. Шифриращи алгоритми

Трите вида шифри, `CaesarCipher`, `TextCodeCipher` и `HillCipher` са обединени от общ базов клас `Cipher`. За него най-важни са криптиращата и декриптиращата функция с обща за всички шифри дефиниция, изглеждаща по следния начин в `Cipher`:

```
virtual EncryptedMessage encrypt(const std::string& text) = 0;
virtual std::string decrypt(const EncryptedMessage& msg) = 0;
```

Използваме и помощен клас `EncryptedMessage`. За него най-важните атрибути са `unsigned char* encrypted`, `size_t len`, `size_t len_original`. Тук в `encrypted` се съхранява криптираното съобщение, `len` е размерът **в байтове** на записаното криптирано съобщение, а `len_original` е просто броят символи в **оригиналното съобщение**. Изборът на тази архитектура се основава на факта, че различните шифри записват в различен вид криптираното съобщение: докато при `CaesarCipher` например това е поредица от символи, при `TextCodeCipher` и `HillCipher` това е поредица от числа. Затова изборът на `unsigned char*` за съхраняване на самото криптирано съобщение е обоснован – в случая, в който то е поредица от символи, те се записват директно в тази памет. В случаите, в които записаното е поредица от числа, можем резонно да допуснем, че всяко от тях се побира в 4 байта памет (за char е ясно, че не можем да надхвърлим тази памет, а за `TextCodeCipher` е напълно резонно да предположим, че дължината на азбуката е не повече от 2,147,483,647 символа). Тогава за всяко число от криптираното съобщение заделяме по 4 байта, започвайки от `encrypted`. Затова е необходимо и да знаем колко байта памет сме записали – в противен случай не бихме могли да знаем кога да спрем да четем. Дължината на изходното съобщение е необходима за проверка при `CaesarCipher` и `TextCodeCipher` и за коректно орязване на изкуствено добавените символи при `HillCipher`. Основната мотивация за употреба на такъв клас беше и да имаме кохерентност, в смисъл, че

```
cipher.decrypt(cipher.encrypt(text)) == text
```

Съвсем накратко ще обърнем внимание на всеки от трите шифъра и ще посочим специфики при тяхното имплементиране. Повече внимание ще обърнем само на `HillCipher` поради необходимите допълнителни класове за неговото реализиране.

I.I `CaesarCipher`

Това е най-стандартният от трите шифъра и няма много интересни неща в него.

I.II `TextCodeCipher`

Този шифър също е стандартен. В началото изчисляваме (веднъж!) по зададения текст за всеки от символите, с които работим (по условие подмножество на тези с ascii кодове между 32 и 126), дали се среща и ако да, къде за първи път. Това реализираме с `int` масив с 95 елемента (колкото са възможните символи), като разглеждаме биекция между тях и разрешените символи, която ни позволява директен достъп до кодиращото число по зададен символ. Поддържаме възможност за гъвкава промяна на фиксирания диапазон от позволени символи.

I.III `HillCipher`

Поради необходимостта от работа с матрици, дефинираме клас `Matrix`, който има и наследник `MatrixSquare`. В последния сме реализирали и методи за намиране на детерминанта, адюнгирана матрица и обратна матрица (за справки – курса по линейна алгебра :)).

Възникнаха две основни дилеми:

- Шифърът на Хил, при който се подава матрица с елементи от полето $\mathbb{Z}_{26}$, е замислен за работа с буквите A, B, ..., Z и никакви други символи.
- При кодиране с шифъра на Хил обикновено текстът се разделя на части, всяка с дължина равна на тази на броя редове в криптолюча, Когато обаче дължината на текста не се дели точно на този брой, последният блок остава непълен и не може да се криптира.

Базирайки се на знанията си от избираемия курс по криптография, взех следните две решения:

- Вместо матрици с елементи от $\mathbb{Z}_{26}$, използваме такива с елементи от $\mathbb{Z}_{95}$. Макар да обмислях възможността да криптирам само главните латински букви, придържайки се към $\mathbb{Z}_{26}$, и да запазя всички останали символи непроменени, това е крайно слаб алгоритъм за криптиране и сякаш разваля цялата красота на шифъра на Хил. Затова взех решение просто да го обобщим за по-голям брой символи.
- За да избегнем проблема последният блок да е непълен, добавяме изкуствено символи `'X'` накрая на оригиналния текст, криптираме го и накрая взимаме само първите `len_original` символа от декриптирания текст. Затова бе и от изключително значение в класа `EncryptedMessage` да запазим и оригиналния брой символи.

Част II Password Manager и йерархия на командите

Осъществяването на командите се извършва чрез два основни класа:
`PasswordManager` и `CommandProcessor`.

`PasswordManager` е Singleton – позволява се само една негова инстанция да съществува едновременно по време на изпълнение на програмата. Като атрибути, той има вектор от обекти на класа `PasswordFile` и поддържа индекс към текущ отворен файл. Поддържа методи, `create, open, save, load, update, delete`.

`CommandProcessor` служи за processing на въведените команди от стандартния вход и тяхната валидация. При командата `create` се валидира дали съответната конфигурация е валидна за дадения шифър. Допускаме следния синтаксис и аргументи за трите вида шифри:

- `CaesarCipher: create <filename> CAESAR <password> <integer shift>`
- `TextCodeCipher: create <filename> TEXTCODE <password> <config path>`
- `HillCipher: create <filename> HILL <password> <config path>`

За шифрите `TEXTCODE` и `HILL` като конфигурация се задава път към файл, който я съдържа. В случая на `TEXTCODE` проверяваме файлът да не е празен, а в `HILL` очакваме на първия ред да е размерът на квадратната матрица n, а след това да има n реда с по n цели числа, описващи тази матрица.

Поотделно ще коментираме с няколко думи всяка от командите, като посочим някои особености и имплементационни решения:

Команда `Create`

Командата `create <име на файл> <тип на шифъра> <парола> <конфигурация за шифъра>` създава файл, ако зададеният шифър е валиден с така подадените аргументи. При създаването на файл с командата `create` първите три реда описват шифрирането по следния пример:

```
CIPHER TEXTCODE
PASSWORD pass
ARGS  file path
```

Друга особеност е че независимо от името на файла, което се подава на командата `create`, се създава файл с разширение `.txt`. Причината е да се избегнат проблеми с файлове, породени от това, че потребителят иска да записва шифри във файл `my_file.png` или друго неподходящо разширение. Въпреки това са имплементирани методи за cast-ване на името към такова с разширение `.txt`, тоест не е необходимо при подаване на аргумента име на файла потребителят да добавя разширението `.txt`. Въпреки това, имената `my_file` и `my_file.txt` ще се третират като едно и също, като по този начин се избягва опасност от дублиране на имена и двусмислие кой файл ще бъде отворен. От друга страна, ако потребителят подаде име `file_name.png`, ще бъде създаден `file_name.png.txt`.

Команда `Open`

Командата `open <име на файл> <парола>` позволява отваряне на вече създаден файл с пароли и избор на текущ файл за работа. При изпълнение на командата се проверява дали съществува такъв файл и дали подадената парола съвпада с тази, записана във файла. При неуспешен опит (грешна парола или файлът не съществува) се извежда съобщение за грешка, а при успех — този файл се превръща в "текущо отворен"и всичките последващи операции (`save`, `load`, и др.) се отнасят именно за него.

Команда `Save`

Командата `save <website> <user> <password>` добавя нов запис в текущо отворения файл за въведения сайт (`website`), потребител (`user`) и парола (`password`). Шифрирането се извършва посредством избраната при създаването на файла криптосистема. При TextCode и Hill паролите се записват като поредица от цели числа (числа, разделени със запетая, например: "8|103,245,37,51,..."), като първото число, отделено със символ | е дължината на оригиналната парола. При Caesar шифрираната парола се записва като обикновен низ. Всеки ред във файла (след заглавната информация) има структура:

```
<website> <user> <encrypted-password>
```

където <encrypted-password> е резултатът от сериализацията на шифрования обект (`EncryptedMessage::serialize(bool)`).

В случай, че потребителят се опита да използва тази команда за сайт и потребител, за които вече е известна парола в текущия файл, информираме потребителя за това и го питаме дали би искал все пак да променим паролата с новата.

Команда `Load`

`load <website> [<user>]` позволява зареждане (десериализация и декриптиране) на записана парола за даден сайт и потребител от текущия файл. Ако има запис за този сайт и потребител, се извежда декриптираната парола на екрана. Ако потребител не е подаден, се извеждат всички записани потребители и паролите им за този сайт. Десериализацията става съобразно вида на шифъра: за Caesar — директно като низ, а за останалите — чрез пресмятане на числовата последователност обратно до символи.

Команда `Update`

Командата `update <website> <user> <new-password>` променя паролата на даден сайт и потребител, ако такъв запис съществува. Ако новата парола съвпада с текущо записаната (след шифриране и сериализация), операцията се отказва с подходящо съобщение. По този начин се ограничава излишното презаписване на идентична информация.

Команда `Delete`

С `delete <website> [<user>]` може да се изтрият или конкретен запис (ако е подаден и потребител), или всички записи за дадения сайт (ако потребител не е указан). Изтриването води и до презапис на файловата система с обновено съдържание, където премахнатите записи вече не съществуват.

Команда `quit`/Команда `exit`

Приложението се затваря и програмата приключва.

Appendix: Списък на използваните класове

`class PasswordManager` Основният клас на приложението, отговорен за управлението на файловете с пароли, обработката на командите (създаване, отваряне, запис, зареждане, промяна и изтриване на пароли), както и за държането на текущо отворения файл. Използва вектор от обекти `PasswordFile` за съхранение на всички файлове с пароли, създадени по време на работата на програмата. Този клас е реализиран като сингълтън (single instance), което гарантира, че съществува само един негов екземпляр по време на изпълнението на програмата.

`struct PasswordFile` Представя един файл с пароли — съдържа информация за името, типа на използвания шифър, конфигурация, парола за достъп, както и списък от записи (`Entry`), представящи отделните пароли за сайтове и потребители. Съдържа указател към обект от тип `Cipher`, който реализира криптирането и декриптирането. Веднъж създаден, шифърът не може да бъде променян.

`struct Entry` Съдържа информация за сайта, името на потребителя и паролата му в един ред на файл с пароли.

`class CommandProcessor` Отговаря за извличането и изпълнението на команди, подадени от потребителя през командния ред. Интерпретира всяка команда като текстов низ, парсва параметрите ѝ и делегира изпълнението ѝ към съответните методи на обекта `PasswordManager`. Класът енкапсулира основния цикъл за въвеждане и обработка на команди и е свързващото звено между потребителския интерфейс и основната логика за управление на паролите. Това осигурява гъвкаво и лесно разширяване с нови команди.

`class EncryptedMessage` Класът представя резултата от криптиране — шифрованото съобщение във вид на масив от байтове, заедно с информация за дължината на оригиналния и шифрования текст. Осигурява методи за сериализация и десериализация на съобщенията с различни формати, подходящи както за шифри, работещи над низове, така и за такива, които използват числови масиви. Управлява собствеността на паметта върху шифрования буфер и не позволява копиране, което осигурява безопасност при използване и освобождаване на паметта при движение (move semantics).

`class Cipher` Абстрактен базов клас, дефиниращ общия интерфейс за всички криптиращи алгоритми в системата. Декларира виртуални методи за криптиране (`encrypt`) и декриптиране (`decrypt`), които се ре-

ализират във всеки конкретен шифър. Осигурява и статични методи за валидация и създаване (**static factory pattern**) на шифри по подаден тип и конфигурация, използвайки централно място за селекция и инстанциране на конкретния шифър във време на изпълнение. Класът съдържа и помощни константи, определящи допустимия диапазон от ASCII символи. Използването на фабричния шаблон улеснява добавянето на нови шифри и централизира управлението на типовете шифри.

`class CaesarCipher` Клас, реализиращ шифъра на Цезар — всеки символ от оригиналния текст се измества с фиксиран брой позиции в рамките на зададен диапазон от ASCII символи. Класът предоставя механизми за промяна на стъпката на изместване, методи за криптиране и декриптиране, както и статични функции за валидация и създаване на шифър по зададена конфигурация. CaesarCipher наследява абстрактния клас `Cipher` и участва в централизирания фабричен метод за създаване на шифри. Поддържа както положителни, така и отрицателни стойности за изместване, като гарантира коректно трансформиране на всеки символ в допустимия диапазон.

`class TextCodeCipher` Клас, реализиращ шифър от типа `Text code`, при който всяка буква или символ се кодира с позицията на първото си появяване в зададен референтен текст (азбука). Позволява гъвкаво дефиниране на използваната азбука — тя може да бъде заредена от външен файл или подадена директно като низ. При криптиране всеки символ от въведената парола се заменя с числото, отговарящо на неговата позиция в азбуката. Класът съдържа методи за валидиране и създаване по конфигурация, както и за сериализация и десериализация на шифрованите съобщения. Наследява интерфейса на базовия клас `Cipher` и се интегрира в общия фабричен механизъм на приложението.

`class HillCipher` Клас, реализиращ шифъра на Хил — шифър, който използва обратима матрица на цели числа по модул 95 за защита на паролите. При криптиране всеки блок от символи се представя като вектор, умножава се по ключовата матрица и резултатът се сериализира като поредица от числа. Класът валидира и зарежда ключовата матрица от външен файл, като гарантира нейната обратимост, което е задължително за декриптиране. Осигурява методи за криптиране, декриптиране, валидация и създаване от конфигуриращ файл.

`class Matrix` Обобщен клас за представяне на двумерна матрица от цели числа с произволен брой редове и колони. Реализира базови операции като умножение на матрици, достъп до елемент по индекс и проверка за валидност на размерите. Поддържа конструктори за създаване на матрици от вектор от вектори, от едномерен вектор (ред или колона),

или с фиксирани размери и стойности. Всички стойности се съхраняват и обработват по модул 95, съгласно използваните криптографски алгоритми в приложението. Тази стойност обаче може да се промени с промяна на статична член-данна на класа, отговарящ за шифъра.

`class MatrixSquare` Унаследява класа `Matrix` и представя квадратна матрица (еднакъв брой редове и колони), върху която са реализирани допълнителни линейно-алгебрични операции. Поддържа изчисление на детерминанта и обратна матрица по модул 95, изваждане на минор и създаване на блокова диагонална матрица (block diagonal). Класът се използва основно за реализиране на криптиране и декриптиране по метода на Хил (Hill cipher), където се изисква работа с обратими квадратни матрици.

`class Utility` Помощен статичен (utility) клас, съдържащ набор от общи функции за работа с низове, буфери, модулярна аритметика и асоциирани преобразувания, използвани в целия проект. Класът предоставя средства за директно писане и четене на цели числа от масиви байтове, функции за работа по модул 95 (използван в криптографските алгоритми), намиране на модулен обратен елемент, обръщане към ASCII кодове, премахване на излишни разширения на файлове и премахване на водещи и завършващи интервали в низове. Всички функции са статични, което не позволява създаване на инстанция на класа и гарантира гъвкавост и повторна употреба на кода в практически всички части на приложението.

Възможни подобрения

- **Подобрения в текущия код:** В момента криптирането при `HillCipher` работи с $O(n^2)$ памет, а може спокойно да мине с $O(n)$. Тривиално е да се оправи, но късно се усетих :) Също така в момента в имената на фунцкиите в класовете `Matrix` и `MatrixSquare` присъства константата 95, която е hard-code-ната в условието, но кодът е написан така, че да може да се променя. Това е козметична промяна, но би било по-добре да няма такива константи в имената на функциите.
- **Поддръжка на повече шифри:** Интегриране на нови криптографски методи (например Playfair, personal favourite) може да бъде реализирано лесно благодарение на фабричната структура на кода.
- **Подобрена работа със записи:** Поддръжка на бележки/категории към паролите, възможност за търсене и филтриране по сайт или потребител, автоматично изтриване на стари или неползвани записи и др.
- **Синхронизация и експортиране:** Добавяне на функционалност за експортиране/импорт на пароли към/от друг файл.

- **По-гъвкава обработка на грешки:** По-детайлни съобщения за грешки, логване на неуспешни опити за достъп до файл, ограничаване на брой неуспешни опити за въвеждане на парола и др.
- **По-добра работа с файлове:** Добавяне на възможност в началото потребителят да подаде път към папка, в която да се съхраняват всички новосъздадени файлове.