

**Софийски университет „Св. Кл. Охридски”**

Факултет по математика и информатика

специалност : „Защита на информацията в компютърните системи и мрежи“

**Дисциплина: Моделиране на защитени взаимодействия в компютърните системи**

**Курсова работа на тема “Описание на система за съхранение, проверка и генериране на сигурни пароли”**

*Автор:*

Иван Ивов Чучулски, фак. номер: 4MI3400043

*Ръководител:*

проф. д.т.н. Веселин Целков

летен семестър, 2021/2022г.

Съдържание

[**1.** **Въведение** 3](#_Toc107081520)

[**2.** **Описание на системата** 3](#_Toc107081521)

[**3.** **Описание на основните алгоритми в системата** 6](#_Toc107081522)

[**3.1** **Основни функционални възможности на потребителите** 6](#_Toc107081523)

[**3.2** **Процедура по добавяне на данни за автентикация** 7](#_Toc107081524)

[**3.3** **Процедура по генериране данни за автентикация** 11](#_Toc107081525)

[**3.4** **Процедура по извличане на данни за автентикация** 14](#_Toc107081526)

[**3.5** **Процедура по премахване на данни за автентикация** 17](#_Toc107081527)

# **Въведение**

Целта на проекта е да бъдат формално описани взаимодействията в система за съхранение, проверка и генериране на сигурни пароли. С помощта на граф-схеми, е-мрежи ще бъдат представени основните функционални характеристики на системата и ще бъдат проверени условията за пълнота и непротиворечивост на алгоритмите.

Системата представлява клиент-сървър приложение, което позволява на потребителите да записват пароли за използвани от тях уеб сайтове или други услуги. Приложението дава възможност за генериране на сигурна парола за конкретна услуга, ако потребителите не желаят да измислят такава.

Клиентската част на системата ще поддържа два варианта – десктоп приложение с графичен потребителски интерфейс, което може да се изпълнява на всички основни операционни системи за персонални компютри, а вторият вариант на системата е с конзолен интерфейс, който изисква от потребителя въвеждане на текстови команди.

# **Описание на системата**

При съхранението на чувствителна информация като пароли за достъп е необходимо да се предприемат мерки за защита от неоторизиран достъп, модификация или кражба. За целта ще използваме концепциите за криптографска хеш функция и симетрично криптиране на данните.

Паролите, които потребителите подават при регистрация в системата ще преобразуваме с помощта на криптографска хеш функция и ще пазим в базата от данни резултата от пресмятането, който е поредица от байтовете на техния хеш. При опит за влизане в системата се пресмята хеша на въведената от потребителя парола и за успешен вход е необходимо той да съвпада с този в базата данни. Допълнителна сигурност ще ни донесе използването на т.нар. salt – това е поредица от байтове, която произволно генерираме при регистрация, долепяме към подадената парола и така пресмятаме хеша. По този начин се предпазваме от варианта, в който паролите за достъп на потребителите съвпадат и резултата от хеш функцията ще е еднакъв, а също така и на практика обезсмисляме много от brute-force атаките и атаките с rainbow таблици. Ще използваме 16-байтов salt и криптографската хеш функцията SHA-512.



Фигура 1 Хеширане на парола със salt

Тъй като приложението цели да предостави възможността на потребителите да могат да записват пароли за използвани от тях услуги е нужно записаната информация да бъде записана достатъчно сигурно, но да може да възстановена в явен вид. Това се постига с използването на симетрична криптографска функция, която използва един частен ключ, който трябва да остане таен за криптиране и декриптиране на информацията. За генерирането на този частен ключ ще изискваме от потребителя т.нар. “master” парола, която той трябва да въведе при регистрация. При нужда от добавяне или генериране на парола е необходимо потребителят да въведе своята “master” парола отново, като това добавя още едно ниво на сигурност и автентикация след влизането в системата.



Фигура 2. Симетрично криптиране

За образуването на частния ключ ще използваме алгоритъма “PBKDF2”, който дава възможност да добавим и случайно генерирана последователност от байтове(salt на ключа), тъй като отново съществува възможността “master” паролите на някои потребителите да съвпадат, както и правим по-трудни атака за отгатване на ключа, знаейки само “master” паролата. За самото криптиране използваме алгоритъма AES в режим CBC, като той изисква освен текста за криптиране да бъде подадена и случайно генерирана поредица от байтове, наречена инициализационен вектор(IV), която служи за подравняване на данните и увеличава силата на криптирането. В базата данни записваме криптирания текст, байтовете на salt-та за ключа и байтовете на инициализационен вектор.

При декриптиране потребителят въвежда своята “master” парола, която заедно с байтовете на salt-та за ключа формират тайния ключ. После ключа и инициализационен вектор участват в декриптирането на получаването на паролата в явен вид.

# **Описание на основните алгоритми в системата**

## **Основни функционални възможности на потребителите**

Потребителите в системата могат да изпълняват следните основни действия, като на всеки от тях ще асоциираме примитив, чрез който ще ги идентифицираме в диаграмите.

* Вписване(регистрация) в системата, примитив “Register”
* Заявяване на достъп до системата(Логин), примитив “Login”
* Изпращане на отговор от системата, примитив “SendResponse”
* Анализ и визуализация на отговора, примитив “ViewResponse”
* Отразяване на потребителското действие, примитив “LogUserAction”
* Проверка на наличните функционалности на приложението(Помощ), примитив “Help”
* Въвеждане на данни за автентикация в услуга, примитив “EnterCredentialDetails”
* Въвеждане на “master” парола: примитив “MasterPassword”
* Заявка за добавяне на парола, примитив “AddPassword”
* Въвеждане на данни за автентикация в услуга и генерация на парола, примитив “GeneratePassword”
* Проверка сигурността на данни за автентикация, примитив “CheckPassword”
* Извличане на парола, примитив RetrievePassword
* Премахване на парола, примитив RemovePassword
* Изход от системата, примитив “Quit”

## **Процедура по добавяне на данни за автентикация**

Алгоритъмът за добавяне на парола е последователен. Процедурата позволява на потребител да запише парола за уеб услуга или приложение, което използва. Това действие е достъпно само за потребители, които успешно са преминали процеса по автентикация, избрали са да извършат това действие и имат валидна сесия. Сесията на всеки потребител се стартира след като той успешно влезе в системата и се прекратява, ако потребителят не инициира никаква дейност в продължителност на 10 минути.

Първо ще моделираме стъпките на алгоритъма с помощта на краен автомат. Основните състояния на заявка по добавяне на парола са:

* s1: постъпва нова заявка
* s2: заявката е за добавяне на данни за автентикация
* s3: заявката е невалидна
* s4: подготовка за криптиране на въведената парола
* s5: криптирането на въведената парола е успешно
* s6: успешно добавяне на новите данни в базата данни
* s7: заявката е обработена

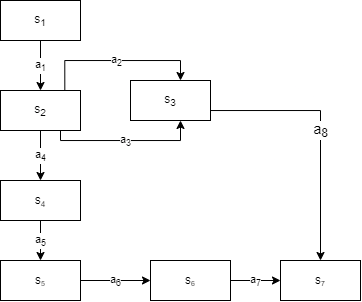
Основните събития, извършващи преходи и промяна състоянието на заявка по добавяне на парола са:

* a1: определяне на заявката
* a2: сесията на потребителя е изтекла
* a3: параметрите на заявката са невалидни
* a4: извличане на записаната master парола с подадената от потребителя
* a5: извършване на операциите по криптиране на въведената парола
* a6: добавяне на новите данни за автентикация в базата данни
* a7: записване на потребителското действие в системния дневник и връщане на отговор
* a8: съобщение за грешка

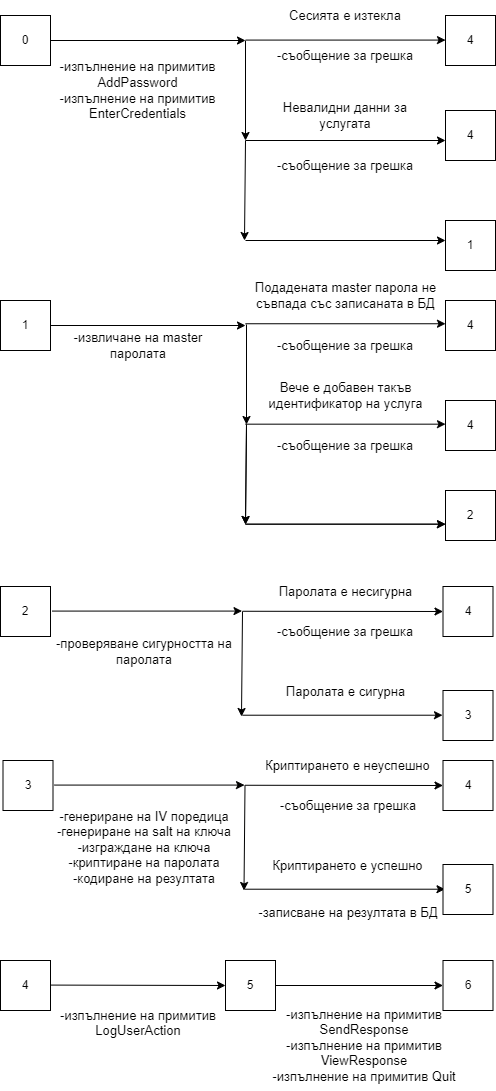
Таблица на състоянията и преходите при процедура за добавяне на данни за автентикация

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **a1** | **a2** | **a3** | **a4** | **a5** | **a6** | **a7** | **a8** |
| **s1** | s2 | - | - | - | - | - | - | - |
| **s2** | - | s3 | s3 | s4 | - | - | - | - |
| **s3** | - | - | - | - | - | - | - | s7 |
| **s4** | - | - | - | - | s5 | - | - | - |
| **s5** | - | - | - | - | - | s6 | - | - |
| **s6** | - | - | - | - | - | - | s7 | - |
| **s7** | - | - | - | - | - | - | - | - |

Следният краен автомат отразява основните състояния и преходи



Нека представим алгоритъма и с със граф-схема описва алгоритъма:



Ще проверим пълнотата и непротиворечивостта на алгоритъма като изградим Матрична схема на алгоритъма(МСА).

Логически условия:

* p1 –валидност на сесията (ДА)
* p2 –валидност на въведените данни за услугата (ДА)
* p3 – дали подадената master парола съвпада със записаната в БД (ДА)
* p4 –дали не съществува такъв идентификатор на услуга в БД (ДА)
* p5 – дали паролата за добавяне не е намерена сред списък от несигурни пароли (ДА)
* p6 – дали криптирането е успешно (ДА)

Оператори:

* A0 – начало на алгоритъма
* A1 – изпълнение на AddPassword, изпълнение на EnterCredentials
* A2 – извличане на master паролата от БД
* A3 – проверка сигурността на паролата
* A4 – криптографски операции по криптиране на подадената парола
* A5 – записване на резултата в БД
* A6 – изпълнение на LogUserAction
* A7 – изпълнение на SendResponse, изпълнение на ViewResponse, изпълнение на Quit
* Ak - край на алгоритъма

Бележка: с "^” означаваме операцията дизюнкция(“логическо или”) и с “V” означаване операцията конюнкция(“логическо и”), а с “!” означаваме операцията отрицание. Следната диаграма представлява матричната схема на алгоритъма:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **A1** | **A2** | **A3** | **A4** | **A5** | **A6** | **A7** | **Ak** |
| **A0** | 1 |  |  |  |  |  |  |  |
| **A1** |  | p1^p2 |  |  |  | ­­!p1 V !p2 |  |  |
| **A2** |  |  | p3­^p4 |  |  | !p3 V !p4 |  |  |
| **A3** |  |  |  | p5 |  | !p5 |  |  |
| **A4** |  |  |  |  | p6 | !p6 |  |  |
| **A5** |  |  |  |  |  |  | 1 |  |
| **A6** |  |  |  |  |  |  | 1 |  |
| **A7** |  |  |  |  |  |  |  | 1 |

Проверяваме условието за пълнота на алгоритъма за редовете A1, A2, A3 и A4 - дизюнкцията на всички елементи за всеки от изброените редове е равна на 1, следователно алгоритъмът е пълен.

Проверяваме условието за непротиворечивост за редовете А1, А2, А3 и А4 - конюнкцията на всеки два елемента за всеки от изброените редове е равна на 0, следователно алгоритъмът е непротиворечив.

## **Процедура по генериране данни за автентикация**

Алгоритъмът за генериране на парола е последователен. Процедурата позволява на потребител да остави на системата задачата да създаде паролата за уеб услуга или приложение, което използва. Това действие е достъпно само за потребители, които успешно са преминали процеса по автентикация, избрали са да извършат това действие и имат валидна сесия, както при процедурата за добавяне на парола. Системата дава на потребителя възможност да зададе желаната дължина на генерираната парола, като при конзолната версия на приложението това се случва чрез числов параметър в командата, а при десктоп приложението чрез въвеждане на стойността посредством слайдер. Потребителят трябва да въведе и своята master парола, тъй като тя отново участва във формирането на ключа за криптирането на генерираната парола.

Следната граф-схема описва процедурата:



Ще проверим пълнотата и непротиворечивостта на алгоритъма като изградим Матрична схема на алгоритъма(МСА).

Логически условия:

* p1 –валидност на сесията (ДА)
* p2 –валидност на въведените данни за услугата (ДА)
* p3 – дали подадената master парола съвпада със записаната в БД (ДА)
* p4 –дали не съществува такъв идентификатор на услуга в БД (ДА)
* p5 – дали генерирането на парола е успешно (ДА)
* p6 – дали криптирането е успешно (ДА)

Оператори:

* A0 – начало на алгоритъма
* A1 – изпълнение на GeneratePassword, изпълнение на EnterCredentials
* A2 – извличане на master паролата от БД
* A3 – генериране на парола
* A4 – криптографски операции по криптиране на подадената парола
* A5 – записване на резултата в БД
* A6 – изпълнение на LogUserAction
* A7 – изпълнение на SendResponse, изпълнение на ViewResponse, изпълнение на Quit
* Ak - край на алгоритъма

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **A1** | **A2** | **A3** | **A4** | **A5** | **A6** | **A7** | **Ak** |
| **A0** | 1 |  |  |  |  |  |  |  |
| **A1** |  | p1^p2 |  |  |  | ­­!p1 V !p2 |  |  |
| **A2** |  |  | p3­^p4 |  |  | !p3 V !p4 |  |  |
| **A3** |  |  |  | p5 |  | !p5 |  |  |
| **A4** |  |  |  |  | p6 | !p6 |  |  |
| **A5** |  |  |  |  |  |  | 1 |  |
| **A6** |  |  |  |  |  |  | 1 |  |
| **A7** |  |  |  |  |  |  |  | 1 |

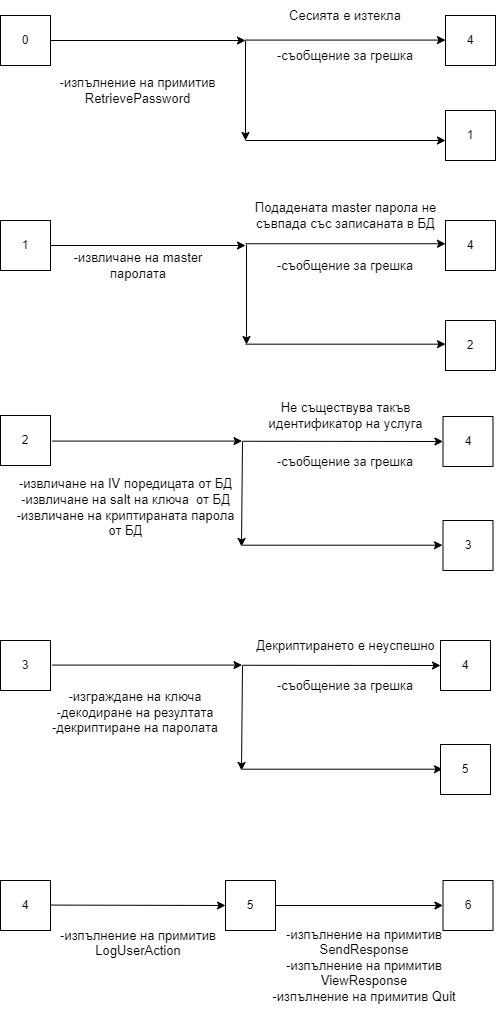
Проверяваме условието за пълнота на алгоритъма за редовете A1, A2, A3, A4 - дизюнкцията на всички елементи за всеки от изброените редове е равна на 1, следователно алгоритъмът е пълен.

Проверяваме условието за непротиворечивост за редовете А1, А2, А3, А4 - конюнкцията на всеки два елемента за всеки от изброените редове е равна на 0, следователно алгоритъмът е непротиворечив.

## **Процедура по извличане на данни за автентикация**

Алгоритъмът за извличане на вече записана парола е последователен. Потребителят трябва да избере за коя от записаните си услуги желае да види паролата. За целта той трябва да въведе своята master парола, тъй като тя участва в операциите по декриптирането и извличането на желаната парола в „чист вид“.

Следната граф-схема описва процедурата:



Ще проверим пълнотата и непротиворечивостта на алгоритъма като изградим Матрична схема на алгоритъма(МСА).

Логически условия:

* p1 –валидност на сесията (ДА)
* p2 – дали подадената master парола съвпада със записаната в БД (ДА)
* p3 –дали съществува такъв идентификатор на услуга в БД (ДА)
* p4 – дали декриптирането е успешно (ДА)

Оператори:

* A0 – начало на алгоритъма
* A1 – изпълнение на RetrieveCredentials
* A2 – извличане на master паролата от БД
* A3 – извличане на данните, нужни за декриптиране
* A4 – криптографски операции по декриптиране на подадената парола
* A5 – изпълнение на LogUserAction
* A6 – изпълнение на SendResponse, изпълнение на ViewResponse, изпълнение на Quit
* Ak - край на алгоритъма

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **A1** | **A2** | **A3** | **A4** | **A5** | **A6** | **Ak** |
| **A0** | 1 |  |  |  |  |  |  |
| **A1** |  | p1 |  |  | ­­!p1 |  |  |
| **A2** |  |  | p2­ |  | !p2 |  |  |
| **A3** |  |  |  | p3 | !p3 |  |  |
| **A4** |  |  |  |  | !p4 | p4 |  |
| **A5** |  |  |  |  |  | 1 |  |
| **A6** |  |  |  |  |  |  | 1 |

Проверяваме условието за пълнота на алгоритъма за редовете A1, A2, A3 и A4 - дизюнкцията на всички елементи за всеки от изброените редове е равна на 1, следователно алгоритъмът е пълен.

Проверяваме условието за непротиворечивост за редовете А1, А2, А3 и A4 - конюнкцията на всеки два елемента за всеки от изброените редове е равна на 0, следователно алгоритъмът е непротиворечив.

## **Процедура по премахване на данни за автентикация**

Алгоритъмът за премахване на вече записана парола е последователен. Потребителят трябва да избере за коя от записаните си услуги желае да премахне. След това с цел увеличена сигурност, се изисква той да въведе своята master парола. В този случай тя не участва в никакви криптографски операции, но се явява един вид допълнителна тайна, която увеличава защитата от затриване на лична информация. Един възможен сценарий е в случай на нерегламентиран достъп до системата от човек, различен от потребителя или дори при оставен активен софтуер не е възможно нарочно изтриване на записаните данни за услугите без знанието на master паролата.

Следната граф-схема описва процедурата:



Ще проверим пълнотата и непротиворечивостта на алгоритъма като изградим Матрична схема на алгоритъма(МСА).

Логически условия:

* p1 –валидност на сесията (ДА)
* p2 – дали подадената master парола съвпада със записаната в БД (ДА)
* p3 –дали съществува такъв идентификатор на услуга в БД (ДА)

Оператори:

* A0 – начало на алгоритъма
* A1 – изпълнение на RemoveCredentials
* A2 – извличане на master паролата от БД
* A3 – търсене на идентификатора за премахване
* A4 – премахване на записаните данни за паролата
* A5 – изпълнение на LogUserAction
* A6 – изпълнение на SendResponse, изпълнение на ViewResponse, изпълнение на Quit
* Ak - край на алгоритъма

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **A1** | **A2** | **A3** | **A4** | **A5** | **A6** | **Ak** |
| **A0** | 1 |  |  |  |  |  |  |
| **A1** |  | p1 |  |  | ­­!p1 |  |  |
| **A2** |  |  | p2­ |  | !p2 |  |  |
| **A3** |  |  |  | p3 | !p3 |  |  |
| **A4** |  |  |  |  |  | 1 |  |
| **A5** |  |  |  |  |  | 1 |  |
| **A6** |  |  |  |  |  |  | 1 |

Проверяваме условието за пълнота на алгоритъма за редовете A1, A2 и A3 - дизюнкцията на всички елементи за всеки от изброените редове е равна на 1, следователно алгоритъмът е пълен.

Проверяваме условието за непротиворечивост за редовете А1, А2 и А3 - конюнкцията на всеки два елемента за всеки от изброените редове е равна на 0, следователно алгоритъмът е непротиворечив.