АКАДЕМИЈА ТЕХНИЧКО-УМЕТНИЧКИХ СТРУКОВНИХ СТУДИЈА БЕОГРАД

ОДСЕК ВИСОКА ШКОЛА ЕЛЕКТРОТЕХНИКЕ И РАЧУНАРСТВА

**Ђорђе Миленковић**

**ЈЕДНА ИМПЛЕМЕНТАЦИЈА СИСТЕМА ЗА ЧУВАЊЕ ЛОЗИНКИ НА ПАМЕТНИМ КАРТИЦАМА**

**- мастер рад -**



Београд, март 2022.

Кандидат: **Ђорђе Миленковић**

Број индекса: **РИН-65/17**

Студијски програм: **Рачунарско инжењерство**

Тема: **Једна имплементација система за чување лозинки на паметним картицама**

Основни задаци:

**1. Преглед у област истраживања**

**2. Пројектовање система за чување лозинки**

**3. Имплементација система за чување лозинки**

Београд, март 2022. године. Ментор:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Др Немања Мачек, проф. ВИШЕР

**РЕЗИМЕ:**

У овом мастер раду представљена је имплементација система за чување лозинки на паметним картицама. Дат је теоријски увид у сваки део система а потом је описан практичан део који се састојао од реализације два програма, за рачунар и за паметну картицу.

**Кључне речи:** картица, програм, систем, лозинке

**ABSTRACT:**

In this master thesis is presented the implementation of system for storing passwords on smart cards. Theoretical insight into each part of the system was given, and then the practical part was performed, which consisted of the realization of two programs, for a personal computer and for a smart card.

**Key words:** card, program, system, passwords

САДРЖАЈ

[1. Увод 1](#_Toc99120345)

[2. Анализа рада 2](#_Toc99120346)

[3. Опис паметних картица 3](#_Toc99120347)

[4. JavaCard платформа 4](#_Toc99120348)

[4.1. Развој програма за JavaCard платформу 5](#_Toc99120349)

[4.2. JavaCard аплети 6](#_Toc99120350)

[5. Комуникација са паметном картицом 7](#_Toc99120351)

[5.1. APDU протокол 7](#_Toc99120352)

[5.2. Структура одговора са паметне картице 9](#_Toc99120353)

[6. Основни криптографски појмови 11](#_Toc99120354)

[6.1. Напади на шифрате 12](#_Toc99120355)

[6.2. Симетрична криптографија 12](#_Toc99120356)

[6.3. AES алгоритам 13](#_Toc99120357)

[6.4. Дифи – Хелманов алгоритам 16](#_Toc99120358)

[7. Програмски језик Јава 18](#_Toc99120359)

[7.1. JavaFX технологија 18](#_Toc99120360)

[8. Протокол за успоставу сигурног комуникационог канала између терминала и паметне картице 21](#_Toc99120361)

[9. Опис имплементираног софтверског решења 24](#_Toc99120362)

[9.1. Програм Tерминал 24](#_Toc99120363)

[9.1.1. Опис програмског кoда 24](#_Toc99120364)

[9.1.2. Опис графичког корисничког интерфејса 32](#_Toc99120365)

[9.2. Програм паметне картице 34](#_Toc99120366)

[9.3. Програм за персонализацију паметне картице 39](#_Toc99120367)

[10. Закључак 42](#_Toc99120368)

[11. Литература 43](#_Toc99120369)

[12. Индекс појмова 44](#_Toc99120370)

# Увод

Кроз целу историју човечанства, постојала је потреба за сигурном разменом информација. Проблемом сигурне комуникације бавили су се већ Египћани и Индијци пре више од 3000 година и од тада до данас основна идеја се није променила – пренети неку поруку са једног места на друго што је сигурније могуће, тј. направити алгоритам који би омогућио скривање оригиналне поруке тако да буде потпуно (у идеалном случају) неразумљива особама које би неовлашћено дошле у њен посед.

Поред тога, одувек је постојала потреба за сигурним чувањем података, да би у случају компромитације, подаци били неупотребљиви страни која их је компромитовала. У данашње време поверљиви подаци се најчешће чувају на рачунарима, флеш меморијама, у рачунарима у облаку итд. Њихова мана је што у сваком тренутку могу бити компромитовани и не постоји гаранција да поверљиви подаци неће бити изложени другим странама.

Главна предност паметних картица у односу на друге медијуме за чување података, као што су меморијске картице, је та што омогућавају сигурно чување поверљивих података. Да би се подаци из сигурне меморије на паметној картици прочитали, неопходно је успоставити сигуран канал комуникације између картице и терминала. Сигуран комуникациони канал је неопходан јер приликом читања података са картице, може доћи до пресретања комуникације и преузимања самих података. Циљ овог рада је приказати један систем за сигурно чување поверљивих података као што су кориснички налози и њихове лозинке на паметним картицама и њихово сигурно читање са паметних картица.

# Анализа рада

Узимајући у обзир да су подаци као што су кориснички налози и лозинке веома осетљиви подаци изложени компромитацији, циљ рада је понудити решење где би се такви подаци сигурно чували и користили без могућности компромитације. Данашња софтверска решења за управљање лозинкама се углавном ослањају на рачунаре у облаку, што би била њихова највећа мана. Решење које је реализовано у овом мастер раду се не ослања ни на један интернет сервис и може радити независно од доступоности интернета тако да корисник има потпуну контролу над својим подацима.

Главни допринос рада је реализација два програма за две независне платформе: рачунар и паметну картицу. Идеја је да се помоћу програма на рачунару подаци о корисничким налозима и лозинкама прикупљају и потом смештају на паметну картицу у сигурну меморију. Поред тога, идеја рада је и имплементирати сигуран комуникациони канал између рачунара и паметне картице како би подаци могли да се преносе сигурно и без компромитације.

Овакво решење може наћи своју примену у свим ситуацијама кад је неопходна висока заштита и сигурност података, њихово безбедно преношење, складиштење и читање.

Рад обухвата 12 поглавља. У првом поглављу дат је историјски значај сигурности информација и мотивација за израду оваквог рада. У трећем поглављу описане су основне карактеристике паметних картица. У четвртом поглављу је описана JavaCard технологија, будући да је сама реализација система укључивала развој програма за паметне картице које су базиране на JavaCard технологији. У петом поглављу описана је комуникација са паметном картицом и описан је протокол по ком се комуникација одвија. У шестом поглављу дат је опис основних криптографских појмова. Описан је AES алгоритам и Дифи – Хелманов алгоритам будући да су ти алгоритми искоришћени у реализацији рада. У седмом поглављу дат је кратак опис програмског језика Јава, с обзиром да је тај програмски језик искоришћен за реализацију софтверског решења рада. Осмо поглаве представља опис протокола са успоставу сигурног комуникационог канала између рачунара и паметне картице, који је уједно и део система. Девето поглавље садржи опис имплементираног софтверског решења, описан је програмски кôд апликације за рачунар и за паметну картицу и представљен је графичко кориснички интерфејс апликације за рачунар. У десетом поглављу изнети су закључци реализације рада. Једанаесто поглавље представља литературу коришћену за израду рада, док дванаесто поглавље представља индекс појмова.

# Опис паметних картица

Паметне картице представљају једну од најмањих рачунарских платформи која се данас користи. Иако су веома малих димензија, памерне картице функционишу слично као персонални рачунари јер могу да складиште податке, манипулишу подацима и обављају функције као што су математичке операције, криптографске операције итд. Паметне картице се могу поделити у две форме: контактне и безконтактне. Контактне паметне картице захтевају уметање или дирактан контакт са читачем паметних картица док безконтактне функционишу на принципу магнетног поља и могу да се користе када су прислоњене близу безконтактног читача картица. Паметне картице су највише заступљене у следећим индустријама:

* Телекомуникације (78%)
* Банкарство (7%)
* Здравство (3%)
* Саобраћај и транспорт (2%)
* Остало (10%)

Паметне картице су израђене у потпуности од пластике или папира, осим малог дела предвиђеног за чип картице. Интегрисана кола, или чипови, за паметне картице се производе од силицијумских плочица, баш као и други рачунарски чипови. Ови чипови су везани за алиминијумске површине (да би се обезбедила читљивост) и обложени су епокси смолом пре уграђивања у пластику картице.

Пошто паметне картице садрже централну процесорску јединицу, оперативни систем и различите типове меморије на њеном чипу, њихова анатомија је веома слична персоналним рачунарима. Чип једне паметне картице садржи следеће компоненте:

* **Централна процесорска јединица –** Централна процесорска јединица чипа паметне картице тумачи и извршава инструкције добијене од оперативног система картице.
* **Read Only меморија (ROM) –** ROM меморија складишти податке који се, једним уписани, не могу променити или обрисати. Из тог разлога, произвођачи чипова паметних картица у ROM меморији чувају оперативни систем картице, као и различите функције тестирања и дијагностике.
* **Random Access меморија (RAM) –** RAM меморија може да складишти и мења податке током једне сесије комуникације са читачем картица. Подаци могу да се уписују, мењају и бришу све док паметна картица има напајање тј. док је повезана са читачем картица. Када се напајање изгуби, сви подаци се трајно губе.
* **Меморија апликације (EEPROM) –** За податке који требају бити измењени или обрисани, као што су подаци у апликацији, произвођачи чипова користе избрисиву програмабилну меморију само за читање (EEPROM). EEPROM меморија је веома слична хард диску персоналног рачунара јер подаци у њој могу остати и након губитка напајања и могу бити модификовани током сесије.

# JavaCard платформа

JavaCard платформа се састоји из два дела. Први део представља API за приступ картици и укључује неке криптографске функције. Други део специфицира JavaCard виртуалну машину (JCVM) која је дизајнирана као једноставна Јава виртуелна машина у коју је уграђен подскуп Јава програмског језика. Због свог малог меморијског простора, JavaCard платформа подржава само пажљиво изабран, прилагођен подскуп карактеристика Јава језика. Овај подскуп садржи карактеристике које су прилагођене за писање апликација за паметне картице и друге мале уређаје уз очување објектно оријентисаних принципа Јава програмског језика.

Табела 1.1 приказује поређење карактеристика Јава језика прилагођеног за паметне картице и Јава језика опште намене.

*Табела 1.1 Ограничења JavaCard платформе у односу на Јава језик опште намене*

|  |  |  |
| --- | --- | --- |
| **Функција** | **ЈavaCard** | **Јава** |
| Клонирање класа | Не | Да |
| Клонирање објеката | Не | Да |
| Целобројна вредност дужине 4 бајта | Опционо | Да |
| Податак типа boolean, byte, short | Да | Да |
| Податак типа long, float, double, character | Не | Да |
| Динамично преузимање класа | Не | Да |
| Динамично управљање меморијом | Не | Да |
| Динамично креирања објеката | Да | Да |
| Управљање изузецима | Да | Да |
| Поља објеката | Да | Да |
| Низови | Једнодимензионални | Мултидимензионални |
| Интерфејси | Да | Да |
| Оператори | Сви | Сви |
| Пакети | Да | Да |
| Нити | Не | Да |
| Виртуалне методе | Да | Да |

## Развој програма за JavaCard платформу

Улога JavaCard виртуелне машине се најбоље може објаснити у контексту процеса развоја софтвера за JavaCard платформу. Постоји неколико компоненти које чине JavaCard систем а то су JavaCard виртуелна машина, конвертор за JavaCard платформу, инсталациони алат за инсталирање програма на картицу и сам програм који се покреће на паметној картици, као што је приказано на сликама 4.1а) и 4.1б).

  
*Слика 4.1а) JavaCard платформа[3]*

  
*Слика 4.1б) JavaCard платформа[3]*

Развој JavaCard програма почиње као и било који други Јава програм: програмер пише једну или више Јава класа, компајлира изворни код стварајући једну или више .class датотека. Програм се развија, тестира и покреће на персоналном рачунару у развојном окружењу које представља симулацију за емулацију паметне картице. Затим кад је аплет спреман за инсталирање на паметну картицу, .class датотеке се конвертују у CAP (конвертовани аплет) датотеку помоћу JavaCard конвертора.

JavaCard конвертор прима .class датотеке, из једног или више Јава пакета, као улазне податке и креира JavaCard CAP фајл. JavaCard CAP фајл садржи само пакете аплета, само библиотеке пакета које се користе у пројекту или комбинацију пакета аплета и библиотека. Поред тога, пакети аплета или библиотека у JavaCard CAP фајлу могу бити јавни или приватни.

Поред .class датотека, JavaCard конвертор као улазне параметре прима и извозне фајлове (.exp фајлове). Један извозни фајл садржи информације о садржају других пакета које конвертоване класе импортују. Извозне датотеке се не учитавају на паметну картицу и интерпретер на паметним картицама их директно не користи. Оне се користе у сврхе верификације и повезивања. Садрже потписе метода и поља класа док не садрже сам бајткод програма.

Након конверзије, CAP фајл је спреман за инсталирање на паметну картицу. За инсталирање аплета на паметну картицу користи се посебан програм који се покреће на персоналном рачунару. Неопходно је да читач картица буде повезан са рачунаром и да картица буде конектована са читачем картица. Програм учитава CAP фајл и и шаље одговарајуће команде на паметну картицу. Паметна картица прима команде које између осталог представљају садржај CAP фајла, припрема и инсталира аплет за покретања од стране JavaCard виртуалне машине. Виртуелна машина не захтева прихватање и манимпулацију CAP фајлом. Она само извршава програм на паметној картици након што је аплет успешно инсталиран.

## JavaCard аплети

JavaCard аплете не треба мешати са Јава аплетима само зато што деле исто име. JavaCard аплет је Јава програм који се покреће и извршава на паметној картици. Он није намењен за покретање у интернет прегледачу као што су за то намњени обични Јава аплети. За разлику од многих embedded система, JavaCard аплети не морају да се учитавају у ROM меморију.

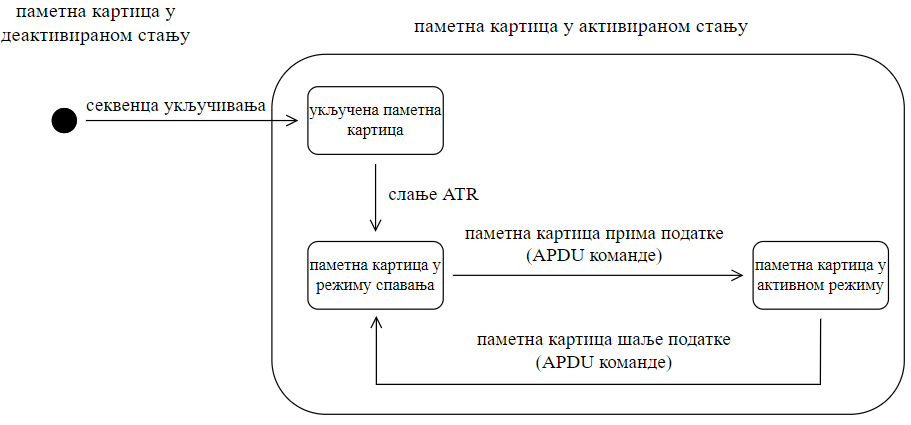
Главна класа аплета мора наследити класу javacard.framework.Applet класу. Основна класа Applet је супер класа за све инсталиране аплете који се налазе на паметној картици. Аплет који ради на картици је заправо инстанца главне класе аплета. Сваки аплет садржи свој идентификациони број и његов пакет садржи свој идентификациони број. Помоћу ових идентификационих бројева, програм који комуницира са картицом зна који аплет треба да покрене.

JavaCard окружење подржава рад са више аплета на паметној картици. На једној паметној картици може постојати више аплета, а један аплет може имати више инстанци. На пример једна инстанца аплета за новчаник може да креира објекат за амерички долар, а други за британску фунту.

# Комуникација са паметном картицом

Предуслов за све интеракције између паметне картице и рачунара је комуникација између две стране. Са конвенционалним протоколима ISO/IEC Т = 0 и Т = 1, само једна комуникациона линија је доступна за то. Ова веза чини основу за размену података између паметне картице и рачунара. Будући да постоји само једна линија, терминал и паметна картица морају да комуницирају наизменично, при чему супротна страна делује као пријемник. Овај процес наизменичног преноса и пријема назива се полу-дуплекс комуникација. Потпуна дуплекс комуникација, у којој обе стране могу истовремено да преносе и примају податке, тренутно није имплементирана у свету паметних картица.

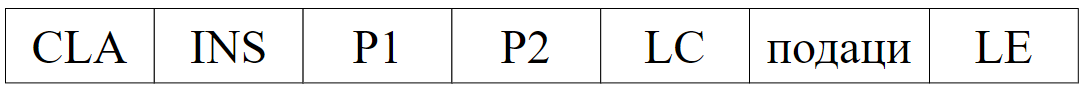
Комуникацију са картицом увек иницира терминал. Картица увек реагује на команде примљене са терминала, што значи да картица никад не шаље податке без спољног иницирања. Ово резултира чистим односом мајстор – роб, са терминалом као мајстором и картицом као робом. Након што паметна картица обради команду која јој је послата и одговор послала терминалу, враћа се у стање мировања. У овом стању је може пробудити само друга наредба послата са терминала. Овакав тип комуникације је приказан на слици 5.1.

  
*Слика 5.1 Стање паметне картице током активирања и комуникације са терминалом*

## APDU протокол

APDU протокол се користи за размену свих података између паметне картице и терминала. APDU је акроним од Application Protocol Data Unit, који означава међународно стандардизовану јединицу података слоја апликације. Постоји разлика између APDU команде (C-APDU) које представљају наредбе послате на картицу и APDU одговора (Р-APDU) које представљају одговоре на наредбе које картица враћа. APDU команда се протоколом преноса транспарентно преноси, што значи без модификације. APDU команде су у складу са ISO/IEC 7816-4 стандардом дизајниране тако да буду независне од преносног протокола. Сходно тома, садржај и формат APDU команде може остати непромењен када се користи другачији протокол преноса.

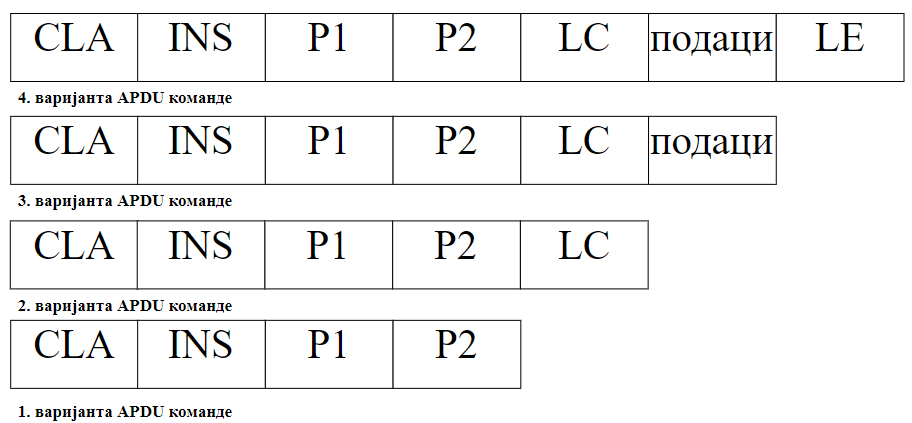
Као што је приказано на слици 5.2, APDU команда се састоји од заглавља и тела. Тело може имате променљиву дужину или може бити потпуно одсутно ако је поље за податке празно. Заглавље се састоји од 4 елемента: бајта класе (CLA), бајта инструкције (INS) и два бајта параметра (P1 и P2). Бајт класе се такође користи за идентификацију апликација. Бајт класе се може користити и као идентификатор да су подаци који се шаљу APDU командом шифровани.



*Слика 5.2 Структура APDU команде*

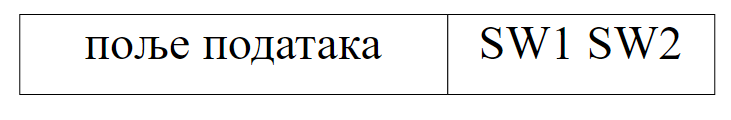
Други бајт у заглављу APDU команде је бајт инструкције (INS). Тај бајт означава функцију коју аплет треба да изврши. Два бајта параметра (P1 и P2) примарно се користе за пружање више информација за функцију изабрану бајтом инструкције. На пример једна функција може у зависности од прослеђених параметара P1 и P2 да уради једну или другу ствар. Након заглавља следи тело APDU команде које може бити празно изузев параметра за дужину тела поруке и параметра за дужину података који се враћају са картице. Параметар за дужину (LC поље) одређује дужину података који ће бити послати APDU командом. Параметар за дужину (LE поље) одређује очекивану дужину података који се враћају са картице. Поред параметара за дужину, тело се састоји од поља за податке који се шаљу на картицу. Ако је вредност LЕ поља „00“, терминал не очекује да ће картица вратити податке.

Поља LC и LE обично имају дужину од једног бајта, али се могу претворити у поља дужине три бајта у зависности од потребе. Ово омогућава да се одреде дужине података до 65536 бајтова. Стандард дефинише ову спецификацију дужине три бајта као резервисану за будућу употребу. Међутим, већ постоје неки оперативни системи паметних картица са великим простором меморије који подржавају спецификације дужине три бајта. Претходно описани елементи APDU команде се могу комбиновати са 4 случаја који су приказани на слици 5.3.

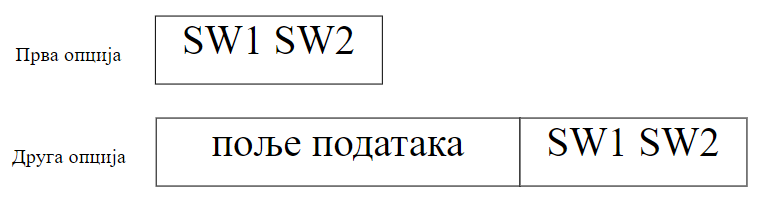
  
*Слика 5.3 Четри могуће варијанте APDU команде*

## Структура одговора са паметне картице

APDU одговор, који картица шаље као одговор на APDU наредбу терминала се састоји од обавезног заглавља и опционог тела, као што је приказано на слици 5.4.

  
*Слика 5.4 Структура APDU команде као одговор са паметне картице*

Тело одговора се састиоји од поља података чија је дужина одређена LE пољем претходне APDU команде коју је терминал послао. Без обзира која је вредност поља ЛЕ у команди која се шаље са терминала ка паметној картици, картица може у сваком тренутку да прекине извршавање команде због грешке. То је означено са два статусна бајта SW1 и SW2 која су приказана на слици 5.5.



*Слика 5.5 Две варијанте структуре одговора са паметне картице ка терминалу*

ISO7816 стандардом је прописано да уколико картица успешно изврши своју функцију и пошаље терминалу одговор, вредност SW1 и SW2 буду 90 и 00. Ако се након извршене наредбе прими повратни статус код „63xx“ или „65xx“ то значи да је меморија на картици (EEPROM или флеш меморија) измењена. Ако је примљен било који други повратни код који започиње са „6x“, то значи да је извршење програма прерано окончанo, без промена у меморији. ISO7816 стандард дефинише и одређене изузетке који се могу догодити током извршења програма на картици. У табели 5.1 приказани су неки од њих.

*Табела 5.1 Изузеци дефинисани ISO7816 стандардом*

|  |  |
| --- | --- |
| Статусни код изузетка | Опис изузетка |
| 67 00 | Погрешна дужина података |
| 69 82 | Безбедоносни услов није испуњен |
| 69 85 | Услови коришћења нису испуњени |
| 6А 86 | Погрешни P1 и P2 параметри |
| 6А 82 | Датотека на картици није пронађена |
| 63 89 | Дужина кључа није валидна |
| 63 C1 | Погрешан пин, још 1 покушај |
| 63 C2 | Погрешан пин, још 2 покушаја |
| 63 C3 | Погрешан пин, још 3 покушаја |
| 6A 81 | Функција на картици није подржана |

# Основни криптографски појмови

Реч криптографија води порекло од грчких речи „kriptos“ што значи скривено и „grafos“ што значи писати. У дословном преводу, реч криптографија значи „скривено писање“. Шифровање обухвата математичке поступке модификације података такве да шифроване податке могу прочитати само корисници са одговарајућим кључем. Процес шифровања трансформише отворени текст, оригиналну поруку или датотеку, помоћу кључа у заштићен, шифрован текст, тј. шифрат. Дешифровање је обрнут процес: шифровани подаци се помоћу кључа трансформишу у оригиналну поруку или датотеку. Шифровани подаци су заштићени од неовлашћеног приступа (корисник без одговарајућег кључа нема приступ шифрованим подацима) и као такви се могу пренети преко несигурног канала или чувати на диску који није заштићен од неовлашћеног приступа. Алгоритам за шифровање може се сматрати сигурним уколико сигурност шифрата зависи само од тајности кључа, а не и од тајности алгоритма.

Алгоритми за шифровање се деле на симетричне (исти кључ се користи и за шифровање и за дешифровање података) и алгоритме са јавним кључем (подаци се шифрују јавним кључем, а дешифрују приватним).

Функција шифровања симетричним алгоритмом *Е* на основу кључа *к* и улазних података *p* производи шифрат *c*. Функција дешифровања *D* на основу истог кључа *к* и шифрата *c* производи оригиналну поруку *p*. Симетрични алгоритми су брзи и као такви се могу користити за шифровање већих датотека или имплементацију у крипто системе датотека. Најпознатији су DES (Data Encryption Standard), AES (Advanced Encryption Standard), IDEA (International Data Encryption Algorithm), Blowfish, Twofish и други.

Функција шифровања алгоритмом са јавним кључем *Е* на основу јавног кључа *к1* и улазних података *п* производи шифрат *ц*. Функција дешифровања *Д* на основу приватног кључа *к2* и шифрата *ц* производи оригиналну поруку *п*. Јавни кључ је познат оним особама са којима корисник жели да комуницира, док је тајни кључ познат само кориснику који је овлашћен да дешифрује поруке. Приватни и јавни кључ су математички повезани, али се приватни кључ не може одредити на основу јавног кључа. Асиметрични алгоритми су спорији и примењују се за дигитално потписивање и шифровање кључева симетричних алгоритама којима су шифроване датотеке. Најпознатији алгоритми за шифровање са јавним кључем су RSA и ELGamal.

Дигитални потпис је електронска верзија потписа, на основу којег се може идентификовати пошиљалац и доказати веродостојност поруке. Дигитални потписи уско су повезани са појмовима хеш и једносмерна хеш функција. Једносмерна хеш функција на основу улазног податка било које дужине производи резултујући низ тачно одређене дужине – хеш који, условно речено, једнозначно идентификује улазни податак. При томе се, због строге једносмерности хеш функције, оригинални подаци не могу одредити. Најчешће коришћене хеш функције су MD5 и SHA1. Приликом потписивања, пошиљалац најпре једносмерном хеш функцијом рачуна хеш *х1* поруке *п*, коју после тога потписује својим приватним кључем (условно се може схватити као шифровање приватним кључем). Пошиљалац шаље оригиналну поруку и дигитални потпис примаоцу. Прималац одређује хеш *х2* примљене поруке и проверава примљени потпис *с1* јавним кључем пошиљаоца (условно се може схватити као дешифровање јавним кључем). Упоређивањем вредности *х1* и *х2* проверава се идентитиет пошиљаоца.

## Напади на шифрате

Циљ напада на шифрат је откривање отвореног текста, или, још чешће кључа којим је отворени текст шифрован. Основна претпоставка криптоанализе је да криптоаналитичар зна који се криптосистем користи (Керцкхоффсов принцип). Наравно, ова претпоставка, у конкретном случају, не мора бити тачна, али се сложеност процедуре битно не мења чак и ако криптоаналитичар треба да провери неколико могућих криптосистема. Дакле, ми претпостављамо да тајност шифрата у потпуности лежи у кључу. Напади се могу класификовати у следеће категорије:

* Само шифрат. Криптоаналитичар поседује само шифрате неколико порука шифрованих помоћу истог алгоритма. Његов је задатак да открије отворени текст што већег броја порука или, у најбољем случају, да открије кључ којим су поруке шифроване.
* Познат отворени текст. Криптоаналитичар поседује шифрат неке поруке и њему одговарајући отворени текст. Његов задатак је да открије кључ или неки алгоритам за дешифровање порука шифрованих тим кључем.
* Одабран отворени текст. Криптоаналитичар је добио привремени приступ алату за шифровање, тако да може добити шифрат одабраног отвореног текста. Овај напад је јачи од претходног.
* Одабрани шифрат. Криптоаналитичар је добио приступ алату за дешифровање, тако да може добити отворени текст одабраног шифрата Ово је типичан напад на криптосистеме са јавним кључем.
* Поткупљивање, уцена, крађа и сличне активности. Овај напад не спада у математичке облике криптоанализе, али је врло ефикасан и често се употребљава.

## Симетрична криптографија

Симетрична криптографија је најстарији облик криптографије, стара је готово колико и људска комуникација. За процес шифровања у симетричној криптографији потребно је знати алгоритам шифровања и тајни кључ. Некад су се алгоритми држали у тајности, али се показало да скривање алгоритама не доприноси сигурности. Сви савремени симетрични алгоритми јавно су објављени. Због тога их је у потпуности могуће тестирати и проверити њихову отпорност на нападе, то јест могуће их је анализирати.

Начин коришћења симетричне криптографије најлакше је показати следећим примером. Пошаљилац и промаоц поседују заједнички тајни кључ, који само они знају, као и заједнички алгоритам шифровања који ће се користити. Када пошаљилац жели да пошаље поруку примаоцу, он зашифрује оригиналну поруку коришћењем тајног кључа и претходно договореног алгоритма. Тиме добија шифрат поруке који може да пошаље примаоцу јавним каналом. Промаоц прима шифрат и дешифрује га својим кључем како би добио оригиналну поруку. Уколико неко прислушкује њихову комуникацију, прима само шифровану поруку јер је једино она послата јавним каналом тако да је тајност комуникације очувана. Мана симетричне криптографије је што се подразумева да су се две стране, пошаљилац и примаоц, унапред договорили о вредности кључа за шифровање и дешифровање који мора остати у тајности од неауторизованих корисника. Код таквог преноса, где се користи јединствени кључ, могуће је извршити напад сировом силом (енгл. brute-force attack), који подразумева испробавање свих могућих комбинација тајног кључа све док се не пронађе коришћена комбинација. Алгоритми са симетричним кључем могу да се посматрају кроз две групе:

* Секвенцијални алгоритми
* Блоковски алгоритми

Секвенцијални алгоритми у поступку шифровања и дешифровања делују у једном тренутку на један бит (понекад бајт) отвореног текста или шифрата. При томе у поступку шифровања резултат трансформације једног бита отвореног текста је потпуно независтан од вредности отвореног текста. Основна јединица отвореног текста може бити и бајт (уместо бита), а разлози су историјски јер се на тај начин може представити једно слово.

Код блоковских алгоритама се отворени текст, пре шифровања дели на скупине узастопних бита одређене дужине (64 бита, 128 бита...) који се називају блокови. Блоковски алгоритам шифровања делује на све бите унутар блока, а добијени шифрат је по поравилу исте дужине као и блок отвореног текста, а пожељно је да вредност сваког бита шифрата зависи од вредности свих бита блока отвореног текста и свих бита кључа шифровања. Блоковски алгоритми треба да задовоље:

* Својство дифузије. То значи да познавање неког пара блокова отвореног текста *Pi* и њему припадајућег шифрата *Ci*, не сме да омогући да се из неког другог блока шифрата *Cj* одреди блок отвореног текста *Pj*. Такође, мале промене у блоку отвореног текста треба да изазову веома велике промене у припадајућем блоку шифрата.
* Својство конфузије. Код напада потпуном претрагом кључева, сви кључеви треба да буду подједнако вероватни.
* Комплетност. Сваки бит шифрата треба да буде функција сваког бита кључа.

## AES алгоритам

С обзиром на то да је у практичној реализацији система за чување лозинки на паметним картицама коришћен AES алгоритам, у наставку ће бити описане његове карактеристике.

Крајем деведесетих година утврдило се да постојећи симетрични алгоритми нису безбедни и започета је потрага за новим решењем које ће касније постати стандард за блоковске алгоритме. Расписан је конкурс на ком је победничко решење прихваћено као стандард (2001. године) под називом Advanced Encryption Standard – AES.

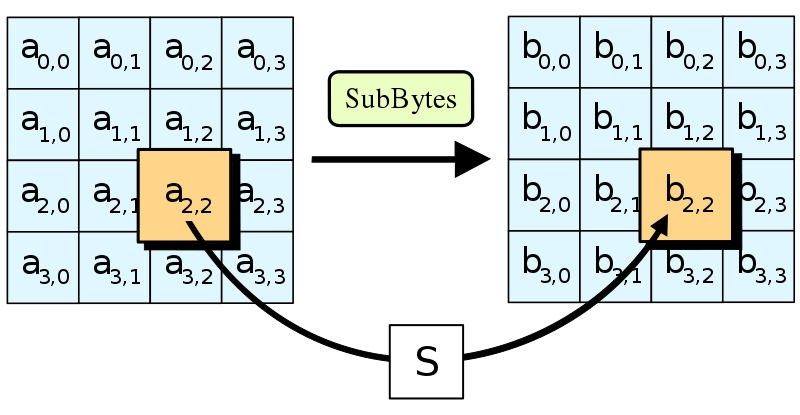
AES је итеративна блоковска шифра. Показао се као брз уз могућу паралелну имплементацију. За сада нису познати ефикасни напади на AES алгоритам.

Блок отвореног текста је дужине 128 бита (мада је оригинално решење дозвољавало и дужине од 192 и 256 бита), дужина кључа може да се бира између 128, 192 и 256 бита, а број рунди алгоритма је између 10 и 14 у зависности од дужине кључа. Отворени текст се прво дели на блокове дужине 128 (192 или 256) бита. Сваки блок се шифрује посебно и као резултат се добија блок шифрата исте дужине као и блок отвореног текста.

Свака рунда се састоји од 4 функције:

* SubBytes (нелинеарни слој)
* ShiftRow (слој линеарног мешања)
* MixColumn (нелинеарни слој)
* AddRoundKey (додатни слој кључа)

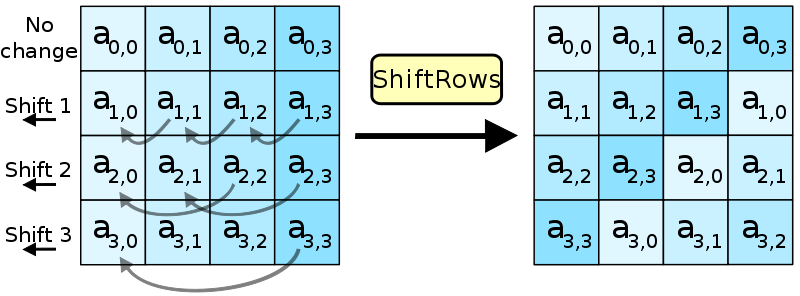
Функција SubBytes је једина нелинеарна трансформација. Подаци из једног блока отвореног текста се прво запишу у матрицу а, одређених димензија, а потом се нелинеарном функцијом добија нова матрица б. На слици 6.1 је шематски приказана трансформација.



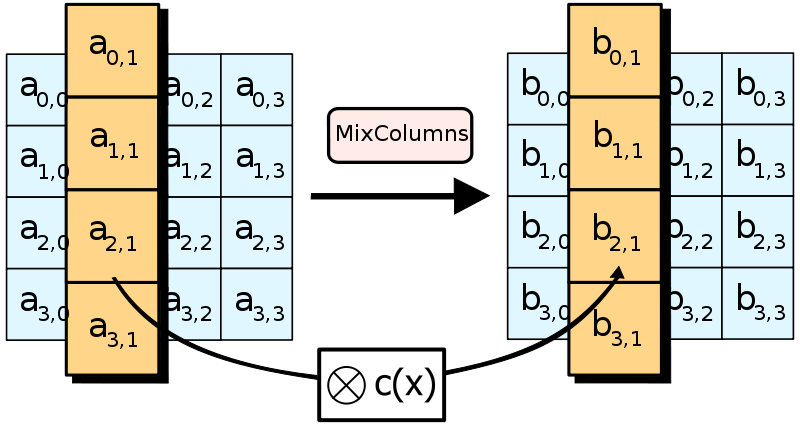
*Слика 6.1 SubBytes функција AES алгоритма. Преузето са <https://sh.wikipedia.org/wiki/Datoteka:AES-SubBytes.svg>*

SubBytes функција као улазни аргумент има вредност бајта отвореног текста (нпр. 0x3C). На основу виша 4 бита (3) се адресира врста табеле а на основу нижа 4 бита (C) колона табеле. Табела садржи 256 различитих вредности, а њихов садржај је унапред дефинисан. Вредност бајта у матрици б у који ће се пресликати бајт из матрице а се добија из табеле са унапред дефинисаним подацима.

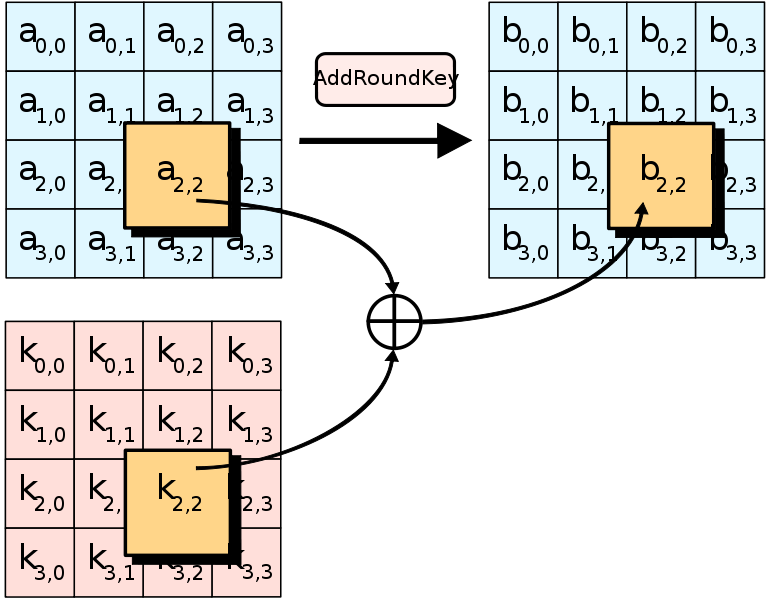
ShiftRow функција је линеарна, она циклично помера последња три реда матрице, добијене након деловања функције SubBytes, за 1, 2 и 3 места. На слици 6.2 је приказан поступак цикличног померања.

  
*Слика 6.2 ShiftRows функција AES алгоритма. Преузето са <https://en.wikipedia.org/wiki/File:AES-ShiftRows.svg>*

MixColumn је такође нелинеарна функција, битна за сигурност AES алгоритма. Премешта колоне по дефинисаним правилима и при томе сваку колону множи са одговарајућом матрицом. Шематски приказ је приказан на слици 6.3.

  
*Слика 6.3 MixColumn функција AES алгоритма. Преузето са <https://en.wikipedia.org/wiki/File:AES-MixColumns.svg>*

Код AddRoundKey функције из целокупног кључа се издваја подкључ који се користи за шифровање у итеративном процесу. Од подкључа се дефинише матрица која се потом сабира по модулу 2 са матрицом добијеном након примене претходних операција као што је приказано на слици 6.4.

   
*Слика 6.4 AddRoundKey функција AES алгоритма. Преузето са <https://en.wikipedia.org/wiki/File:AES-AddRoundKey.svg>*

Један од основних проблема у криптографији са тајним кључевима је дистрибуција или размена криптографских кључева. Два корисника који у међусобној комуникацији размењују шифроване податке морају пре почетка комуникације да изаберу тајни кључ. За размену тајног кључа учесници у комуникацији морају користити безбедан канал. Ако се деси да сигуран канал за размену кључа не постоји, тада размена кључа представља проблем који може значајно угрозити безбедност комуникације. Упркос опште прихваћеном мишљењу да је овај проблем нерешив, једна група ентузијаста (Whitfield Diffie, Martin Hellman и Ralph Merkle) је крајем 70-их година понудила решење.

## Дифи – Хелманов алгоритам

Дифи – Хелманов алгоритам представља алгоритам за размену кључева. Користи се за размену заједничког симетричног кључа. Није намењен за шифровање или дигитално потписивање. Сигурност Дифи – Хелмановог алгоритма се заснива на рачунској сложености израчунавања (једносмерне функције) дискретног логаритма.

Дифи и Хелман су тражили математичке функције за које редослед шифровања и дешифровања није битан, нпр. . Овакве функције постоје, већина их је двосмерна и самим тим могу се лако израчунати. Међутим, овакве функције нису пожељне у криптографији. Од интереса су једносмерне функције, тачније неки облици ових функција.

Једносмерне функције релативно лако могу да се израчунају, али њихова инверзна вредност може да се одреди само изузетно сложеним поступком. За познато *g* и *x*, где је , може да се одреди *n*, . Међутим ако је , *n* се такође одређује преко логаритма или дискретног логаритма.

Математичка основа Дифи – Хелмановог алгоритма се може приказати на следећи начин. Нека је *p* велики прост број и *g* такво да се за свако , може наћи *n* тако да је: . Вредности *p* и *g* су јавне вредности, што значи да се могу размењивати јавним каналом. Алгоритам се састоји из следећих корака:

* Алиса и Боб (али и Труди знају јавне вредности *p* и *g*).
* Алиса бира тајну вредност *a* (велики случајан цео број).
* Боб бира тајну вредност *b* (велики случајан цео број).
* Алиса јавно шаље вредност Бобу.
* Боб јавно шаље Алиси.
* Обоје рачунају заједничку тајну вредност

Та заједничка тајна вредност може да се користи као симетрични кључ. Поставља се питање да ли Труди може на основу размењених порука да сазна тајну вредност ? Труди зна вредност и . Труди може да израчуна вредност али добијена вредност није једнака . Све док Труди не може да израчуна инверзни дискретни логаритам, она не може да дође до тајне вредности .

Дифи – Хелманов алгоритам је осетљив на напад типа човек у средини. За овај проблем постоји више решења. Суштина је да се уведе механизам аутентификације на основу којих би обе стране биле сигурне у порекло порука. Једно од могућих решења је и додатно шифровање.

# Програмски језик Јава

С обзиром на то да је у реализација софтверског решења система за чување лозинки на паметним картицама написана у Јава програмском језику, у наставку ће бити описане каристике Јавe. Јава је објектно оријентисани програмски језик који је развила компанија Sun Microsystems почетком деведесетих година. Тренутно је један од најчешће коришћених програмских језика. Јава је сродник језика C++, који је директан потомак језика C. Већи део својих особина Јава је наследила од ова два језика. Из језика C Јава је прузела синтаксу. Једна од темељних водиља у развоју овог језика, која је довела до опште прихваћености је идеја „напиши једном и покрени било где“. То значи да су јава програми независни од оперативног система, тј. могу се покренути на било ком оперативном систему.

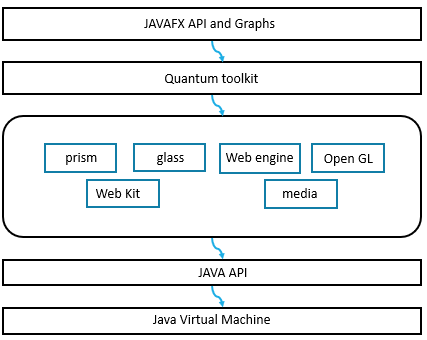
Јава данас има запажен утицај на интернету, али такође представља срж многих апликација и уређаја са којима се свакодневно сусрећемо. Јава је присутна у мобилним уређајима, персоналним рачунарима, видео играма, навигационим системима, пословним апликацијама итд. Јава програмско окружење постало је главни избор за робусне и велике пословне системе.

Осим стандардних библиотека које су доступне у оквиру Јава платформе, око Јава платформе створен је богат екосистем који је изнедрио читав низ других библиотека, развојних оквира па чак и нових програмских језика који су данас у употреби.

Програми писани у Јави се преводе у машински језик, али у мешински језик рачунара који заправо не постоји. Овај такозвани рачунар се зове Јава виртуална машина. Машински језик за Јава виртуалну машину се зове Јава бајткод.

## JavaFX технологија

Технологија JavaFX је замишљена као основна технологија која би омогућила развој богатих графичких апликација за широк спектар уређаја, као што су рачунари, мобилни телефони, телевизори, аутомобилски системи и слично. JavaFX омогућава једноставно и елегантно коришћење декларативног приступа развоју графичког интерфејса, коришћењем специјалног језика FXML. JavaFX окружење укључује JavaFX Desktop, JavaFX mobile i JavaFX TV. У позадини JavaFX-а су бројне компоненте које коришћењем постојећих библиотека дефинишу функционалност за креирање графичко корисничког интерфејса. На слици 7.1 је приказана архитектура JavaFX технологије.

  
*Слика 7.1 JavaFX архитектура. Преузето са [www.educba.com/what-is-javafx/](https://www.educba.com/what-is-javafx/)*

Почетна тачка приказане архитектуре и уједно тачка при конструкцији JavaFX апликације јесте scene graph. Другим речима може се рећи да је JavaFX графичко корисничко окружење реализовано као сцена. Сцена је колекција виртуелних елемената који се конструишу коришћењем JavaFX API-ја. Елемент сцене се назива чвор. Сваки чвор има свој јединствени идентификациони број и може реаговати на корисничку акцију. Над чворовима се могу дефинисати различити ефекти и трансформације. Са изузетком коренског чвора, сваки чвор може имати једног родитеља и произвољан број потомака.

За конструисање сцене користе се функционалности дефинисане у JavaFX API-јима. Оне омогућавају довољну слободу при развоју апликације са богатим графичким окружењем. JavaFX омогућује коришћење свих моћних функционалности Јава језика, као што су генерички типови, анотације, подршка за вишенитно програмирање итд.

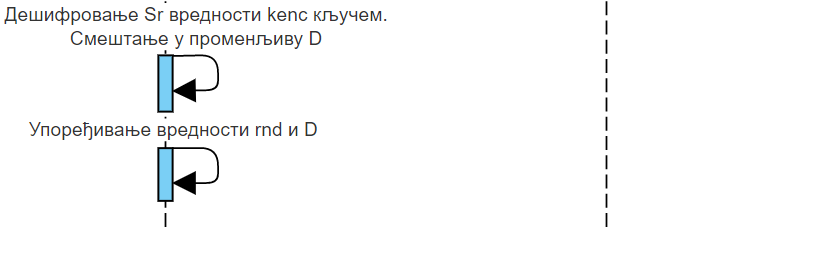
Да би се JavaFX сцена приказала и евентуално анимирала потребно ју је рендеровати. Prism је систем који обавља посао рендеровања. Prism у потпуности упошљава хардвер графичког система за рендеровање на оним машинама где је тако нешто подржано. Ако хардверско рендеровање није подржано, користи се Јава 2D. Пре него што се ослони на овај резервни механизам рендеровања, који подржава свака виртуелна машина, JavaFX ће покушати да упосли Direct X на Windows оперативном систему или Open GL на Mac или Linux оперативним системима.

С обзиром на то да је један од главних циљева Јаве платформска независност, те да различите платформе на себи својствен начин реализују графичко корисничко окружење, у неком тренутку мора доћи до раздвајања кода на сегменте прилагођене конкретним подржаним платформама. Управо то се дешава у овом слоју. Он служи као платформски завистан слој који повезује JavaFX са оперативним системом и тако омогућава најнизе графичке функционалности као што су руковање прозорима, тајмерима и догађајима итд.

JavaFX омогућава подршку како за аудио датотеке, тако и за мултимедијалне видео датотеке. Media engine користи засебну нит за процесуирање мултимедије. Заснива се на мултимедијалном оквиру отвореног кода Gstreamer. Web engine слој је одговоран за обраду HTML садржаја. JavaFX поседује UI компоненту под називом web viewer, који користи Web engine. Са друге стране, Web engine се заснива на webkit-u, web engine-u отвореног кода који подржава HTML, CSS, JavaScript, DOM и SVG.

# Протокол за успоставу сигурног комуникационог канала између терминала и паметне картице

С обзиром на то да су кориснички налози и лозинке веома осетљиви подаци, у раду је било неопходно успоставити сигуран комуникациони канал између терминала и паметне картице. Протокол је имао за циљ креирање заједничког симетричног кључа којим би се даља комуникација шифровала и дешифровала. У самој реализацији, искоришћен је Дифи – Хелманов алгоритам за размену кључева, док је као алгоритам шифровања коришћен AES алгоритам. На слици 8.1 је приказан секвенцијални дијаграм протокола за успоставу сигурног комуникационог канала између терминала и паметне картице а након слике и објашњење сваког корака.





rnd и D

*Слика 8.1 Секвенцијални дијаграм реализације протокола успоставе сигурног комуникационог канала*

Редослед извршавања протокола је следећи:

1. Терминал генерише случајну тајну вредност дужине 16В и смешта је у променљиву *rnd*.
2. Терминал генерише другу случајну вредност дужине 128В и смешта је у променљиву *b*.
3. Рачуна се јавна вредност *B* по формули , где су *G* и *P* јавни параметри Дифи – Хелмановог алгоритма.
4. Терминал иницира комуникацију са паметном картицом тако што APDU протоколом шаље команду за селекцију аплета и након тога шаље јавну вредност *B*.
5. Паметна картица прима вредност *Б* и генерише своју тајну вредност дужине 128В и смешта је у променљиву *а*.
6. Картица рачуна вредност *А* по формули , где су *G* и *P* јавни параметри Дифи – Хелмановог алгоритма.
7. Картица рачуна вредност К по формули .
8. Картица изводи кључ за шифровање и дешифровање из параметра *К* и смешта га у променљиву *kenc*.
9. Картица одговара терминалу тако што шаље израчунату вредност *А*.
10. Терминал прима одговор од картице и рачуна параметар свој параметар *К* по формули .
11. Из параметра К, терминал изводи свој кључ за шифровање и дешифровање *kenc*.
12. Терминал помоћу кључа *kenc* шифрује вредност *rnd*, коју је изгенерисао у првом кораку, смешта шифрат у променљиву *S* и шаље шифрат картици.
13. Картица прима шифрат *S*, дешифрује својим кључем *kenc* и добија вредност *rnd* коју је терминал зашифровао.
14. Картица шифрује вредност *rnd* својим кључем *kenc* и резултат шифровања смешта у променљиву *Sr*.
15. Картица одговара терминалу тако што шаље вредност *Sr*.
16. Терминал дешифрује вредност *Sr* својим *kenc* кључем и резултат дешифровања смешта у променљиву *D*.
17. Терминал упоређује вредности *D* и *rnd* и уколико су њихове вредности једнаке протокол је успешно завршен и сигуран комуникациони канал је успостављен. Уколико вредности нису једнаке, описани кораци се понављају. Уколико се сигуран комуникациони канал не успостави ни после петог покушаја, корисник треба подићи картицу са читача и поново покренути успоставу протокола.

# Опис имплементираног софтверског решења

Софтверско решење система за чување лозинки на паметним картицама представља реализацију два програма за две независне платформе: Паметну картицу и персонални рачунар.

Поред тога, за коришћење оваквог система, неопгходно је поседовати паметну картицу, читач картица као и одговарајући софтвер који се користи за инсталирање програма на картицу. У раду је изабрана паметна картица произвођача NXP, модел NXP JCOP 4 JavaCard 3.0.5 Classic, и читач картица произвођача Omnikey, модел Omnikey 5522.

## Програм Tерминал

Терминал је програм који се извршава на персоналном рачунару и који комуницира са паметном картицом. Основна функција програма је сигурна комуникација са паметном картицом, учитавање података о корисничким налозима и лозинкама на паметну картицу и читање података са паметне картице. Програм се састоји из два екрана, први за пријављивање на систем и други главни екран за приказ података, додавање нових и брисање постојећих. Пре покретања програма, од корисника се очекује да повеже читач картице са рачунаром, као и да поседује паметну картицу са инсталираним аплетом и смештену на читач картице.

### Опис програмског кoда

Програм је написан у Java програмском језику, користећи JavaFX технологију. Развојно окружење коришћено за развој програма је IntelliJ Idea Comunity. Java програмски језик је изабран за развој због добре подршке за комуникацију са паметном картицом као и због напредних технологија за развој графичко корисничког интерфејса.

Пројекат је подељен у 8 пакета и садржи 17 класа. Због бољег одржавања апликације као и због саме организације, у пројекту су одвојени делови за дизајн екрана, догађаје приликом корисникове интеракције, комуникацију са паметном картицом и моделе податка који се користе у комуникацији са паметном картицом. На слици 9.1 приказана је структура пројектa Tерминал.



*Слика 9.1 Структура пројекта Терминал*

Главни пакет пројекта је пакет *application* и у оквиру њега налазе се остали пакети апликације. Пакет *exceptions* је пакет који садржи класу CardInitException која служи за управљање изузецима који се могу десити при комуникацији са паметном картицом. У пакету *fxml* налазе се сви fxml фајлови који представљају структуру графичко корисничког интерфејса апликације. Fxml фајлови су део JavaFX технологије. Пакет *services* садржи класе за контролу догађаја на интеракцију корисника и садржи логику извршавања догађаја. У пакету *style* налази се css фајл за стилизовање графичког интерфејса. Пакет *controllers* садржи све класе које представљају инцијализацију свих графичких компоненти на екране. Пакет *entity* садржи класе које представљају моделе за комуникацију са паметном картицом. У пакету *events* налазе се све класе које представљају иницијализацију догађаја започетих од стране корисника. Класа *SmartCardCommunication* представља класу у којој се налазе све функције за комуникацију са паметном картицом. У наставку ће бити описан кôд функционалности програма за пријаву корисника на систем, успоставу сигурног комуникационог канала и дохватање података са паметне картице.

Класа Main представља главну класу програма и од ње почиње извршавање комплетног програма.

public class Main extends Application {  
 @Override  
 public void start(Stage stage) {  
 try {  
 FXMLLoader loader = new FXMLLoader(getClass().getResource("fxml/LoginScene.fxml"));  
 Parent root = loader.load();  
 Scene loginScene = new Scene(root);  
  
 stage.setTitle("Password Management");  
 stage.setScene(loginScene);  
 stage.setResizable(false);  
 stage.show();  
 } catch (Exception e) {  
 Logger.*getLogger*(LoginController.class.getName()).log(Level.*SEVERE*, null, e);  
 }  
 }  
 public static void main(String[] args) {  
 *launch*(args);  
 }  
}

*Део кода који се односи на покретање програма и извршавање main методе*

Извршавање програма Терминал почиње методом main која позива методу launch. Launch метода извршава сву потребну иницијализацију за покретања програма JavaFX технологије. Метода start се позива након што је извршена сва потребна иницијализација и у њој се дефинишу својства за главну сцену апликације.

Класа LoginController служи за иницијализацију свих елемената на форми за пријављивање корисника.

public class LoginController implements Initializable {  
 @FXML  
 private Button buttonLogin;  
 @FXML  
 private PasswordField password;  
 @FXML  
 private ComboBox<String> cardReaders;  
 @FXML  
 private ProgressBar progressBar;  
 @Override  
 public void initialize(URL url, ResourceBundle rb) {  
 SmartCardCommunication communication = SmartCardCommunication.*getInstance*();  
 communication.loadCardReaders(cardReaders);  
 buttonLogin.setOnAction(new LoginClick(password, cardReaders, progressBar, new LoginService()));  
 }  
}

*Део кода који се представља класу LoginController*

Класа која служи за обрађивање догађаја приликом корисниковог клика на дугме Login је класа LoginClick.У конструктору се инцијализује сервис коме припада логика операције за пријављивање и покривање случаја ако је пријављивање успешно или неуспешно.

public LoginClick(PasswordField password, ComboBox<String> cardTerminals,  
 ProgressBar progressBar, LoginService loginService) {  
 this.password = password;  
 this.cardTerminals = cardTerminals;  
 this.loginService = loginService;  
 progressBar.visibleProperty().bind(loginService.runningProperty());  
 this.loginService.setOnSucceeded(workerStateEvent -> {  
 LoginMessage result = this.loginService.getValue();  
 if (result.getStatusCode() == StatusCode.*OK*)  
 switchScene(actionEvent);  
 });  
 this.loginService.setOnFailed(workerStateEvent ->  
 Util.*showException*((Exception) workerStateEvent.getSource().getException()));  
}

*Кôд који се односи на конструктор класе LoginClick*

Метода handle из класе LoginController представља методу која се извршава на клик дугмета Login на форми за пријављивање. Метода handle покреће нову програмску нит из класе LoginService у којој се одвија комуникација са паметном картицом.

@Override  
public void handle(ActionEvent event) {  
 if (cardTerminals.getSelectionModel().getSelectedItem() == null) {  
 return;  
 }  
 this.actionEvent = event;  
 CardReader.*cardReader* = cardTerminals.getSelectionModel().getSelectedItem();  
 LoginMessage loginMessage = new LoginMessage();  
 loginMessage.setCardReader(cardTerminals.getSelectionModel().getSelectedItem());  
 loginMessage.setUserPin(password.getText());  
 loginService.setLoginMessage(loginMessage);  
 loginService.restart();  
}

*Кôд који се односи на методу handle из класе LoginController*

LoginService класа је класа која представља одвојену програмску нит у којој се извршава комуникација са паметном картицом. Класа наслеђује класу Service и као повратну вредност из нити, програм враћа објекат класе LoginMessage, која представља модел из ког се издвајају подаци који се шаљу на картицу.

public class LoginService extends Service<LoginMessage> {  
 private LoginMessage loginMessage;  
 public LoginMessage getLoginMessage() {  
 return loginMessage;  
 }  
 public void setLoginMessage(LoginMessage loginMessage) {  
 this.loginMessage = loginMessage;  
 }  
 @Override  
 protected Task<LoginMessage> createTask() {  
 return new Task<LoginMessage>() {  
 @Override  
 protected LoginMessage call() throws Exception {  
 SmartCardCommunication communication = SmartCardCommunication.*getInstance*();  
 loginMessage.setStatusCode(StatusCode.*OK*);  
 communication.setCardTerminal(loginMessage);  
 communication.establishSecureChannel(loginMessage);  
 communication.verifyPin(loginMessage);  
 return loginMessage;  
 }  
 };  
 }  
}

*Кôд који представља класу LoginService*

Метода createTask користи инстанцу класе SmartCardCommunication за позивање методе establishSecureChannel и методе verifyPin. Класа SmartCardCommunication користи Singleton пројектни образац јер садржи само једну инстанцу у целом пројекту.

Комуникација са паметном картицом се започиње позивом методе selectApplet, методе за селекцију аплета на паметној картици. Терминал шаље на паметну картицу преко APDU протокола команду за селекцију аплета која се састоји од заглавља и идентификационог броја аплета на паметној картици.

private void selectApplet(LoginMessage loginMessage) throws CardException {  
 Card connection = cardTerminal.connect("T=1");  
 CardChannel cardChannel = connection.getBasicChannel();  
 CommandAPDU commandAPDU = new CommandAPDU(0x00, 0xA4, 0x04, 0x00, hexStringToByteArray(Util.*appletID*));  
 ResponseAPDU responseAPDU = cardChannel.transmit(commandAPDU);  
 if (responseAPDU.getSW() != 0x9000) {  
 loginMessage.setStatusCode(StatusCode.*NoSelectingApplet*);  
 throw new CardException("Error in selecting applet");  
 }  
}

*Кôд који се односи на методу selectApplet*

Након успешног селектовања аплета, Терминал започиње успоставу сигурног комуникационог канала. У функцији getSymetricKey се извршава сав протокол за успоставу сигурног комуникационог канала између Терминала и паметне картице. Након успешно извршене функције сигуран комуникациони канал је успостављен и могућа је даља комуникација са паметном картицом и извршавање програма. Уколико се догоди грешка током извршавања протокола, није могуће користити програм и потребно га је поново покренути.

byte[] bByte = new byte[129];  
random.nextBytes(bByte);  
BigInteger b = new BigInteger(bByte);  
b = b.mod(P);  
CommandAPDU commandAPDU = new CommandAPDU(0x0C, 0x88, 0x00, 0x00,  
 G.modPow(b, P).toByteArray());  
ResponseAPDU responseAPDU = cs.transmit(commandAPDU);  
if (responseAPDU.getSW() != 0x9000) {  
 if (responseAPDU.getSW() == 0x6F00) {  
 loginMessage.setStatusCode(StatusCode.*CardInternalError*);  
 throw new CardException("Something went wrong! Internal Error!");  
 } else {  
 loginMessage.setStatusCode(StatusCode.*GenericError*);  
 throw new CardException(responseAPDU.toString());  
 }  
}

*Део кода који се односи на део методе getSymetricKey у ком се рачуна параметар В и који се шаље на паметну картицу*

Терминал добија вредност А, дужине 256 бајтова са паметне картице, и ту вредност користи за рачунање параметра К из ког може да израчуна кључ за шифровање и дешифровање.

byte[] data = responseAPDU.getData();  
byte[] a = new byte[data.length + 1];  
System.*arraycopy*(data, 0, a, 1, data.length);  
BigInteger A = new BigInteger(a);  
BigInteger K = A.modPow(b, P);  
byte[] sha1;  
if (K.toByteArray().length != 128) {  
 byte[] kbyte = K.toByteArray();  
 byte[] kOK = new byte[128];  
 int bytes = K.toByteArray().length - 128;  
 if (bytes > 0) {  
 System.*arraycopy*(kbyte, bytes, kOK, 0, 128);  
 } else {  
 System.*arraycopy*(kbyte, 0, kOK, -bytes, 128 + bytes);  
 }  
 sha1 = Arrays.*copyOf*(kOK, kOK.length + 4);  
} else {  
 sha1 = Arrays.*copyOf*(K.toByteArray(), K.toByteArray().length + 4);  
}  
sha1[sha1.length - 1] = 0x01;  
sha1 = md.digest(sha1);  
sha1 = Arrays.*copyOf*(sha1, 16);  
sKeyS = new SecretKeySpec(sha1, "AES");

*Део кода који се односи на рачунање параметра К и извођење кључа за шифровање и дешифровање*

Кад Терминал израчуна свој кључ, oн потом шифрује претходно изгенерисану случајну вредност и такав шифрат шаље паметној картици. Паметна картица потом дешифрује вредност шифрата, и поново је шифрује својим претходно изгенерисаним кључем. Након шифровања картица шаље Терминалу свој шифрат, Терминал прихвата вредност шифрата и дешифрује га својим кључем. Након дешифровања Терминал упоређује добијену вредност са својом претходно изгенерисаном случајном вредношћу.

byte[] enc = new byte[randData.length + 1];  
enc[0] = (byte) randData.length;  
System.*arraycopy*(randData, 0, enc, 1, randData.length);  
enc = encrypt(enc, sKeyS);  
commandAPDU = new CommandAPDU(0x0C, 0x89, 0x00, 0x00, enc);  
responseAPDU = cs.transmit(commandAPDU);  
if (responseAPDU.getSW() != 0x9000) {  
 if (responseAPDU.getSW() == 0x6F00) {  
 loginMessage.setStatusCode(StatusCode.*CardInternalError*);  
 throw new CardException("Something went wrong! Internal Error!");  
 } else {  
 loginMessage.setStatusCode(StatusCode.*GenericError*);  
 throw new CardException(responseAPDU.toString());  
 }  
}  
byte[] dec = decrypt(responseAPDU.getData(), sKeyS);  
int len = 16;  
byte[] decRandData = new byte[len];  
System.*arraycopy*(dec, 2, decRandData, 0, len);  
if (Arrays.*equals*(randData, decRandData)) {  
 sameKeys = true;  
}

*Део кода који се односи на шифровање претходно изгенерисане случајне вредности, слање на паметну картицу, дешифровање вредости са картице и упоређивање две вредности*

Завршетком функције getSymetricKey сигуран комуникациони канал је успостављен и могућа је даља комуникација са паметном картицом то јест размена података.

private byte[] encrypt(byte[] message, SecretKey sKey) throws  
 NoSuchPaddingException, NoSuchAlgorithmException,  
 InvalidKeyException, BadPaddingException, IllegalBlockSizeException {  
 Cipher cipher = Cipher.*getInstance*("AES");  
 cipher.init(Cipher.*ENCRYPT\_MODE*, sKey);  
 return cipher.doFinal(message);  
}  
  
private byte[] decrypt(byte[] encrypted, SecretKey sKey) throws  
 NoSuchPaddingException, NoSuchAlgorithmException,  
 InvalidKeyException, BadPaddingException, IllegalBlockSizeException {  
 Cipher cipher = Cipher.*getInstance*("AES/ECB/NoPadding");  
 cipher.init(Cipher.*DECRYPT\_MODE*, sKey);  
 return cipher.doFinal(encrypted);  
}

*Кôд који се односи на методе за шифровање и дешифровање података при комуникацији са паметном картицом*

Након успоставе сигурног комуникационог канала, програм захтева повлачење свих података са паметне картице и учитавање на главни екран апликације. Програм прелази на главни екран апликације и позива методу getUserAccountDataFromCard, методу за повлачење свих података са паметне картице. Метода се налази у класи SmartCardCommunication, позива је нова нит из класе ReadDataFromCardService, док податке обрађује и приказује на графичко корисничком интерфејсу класа ReadDataFromCard.

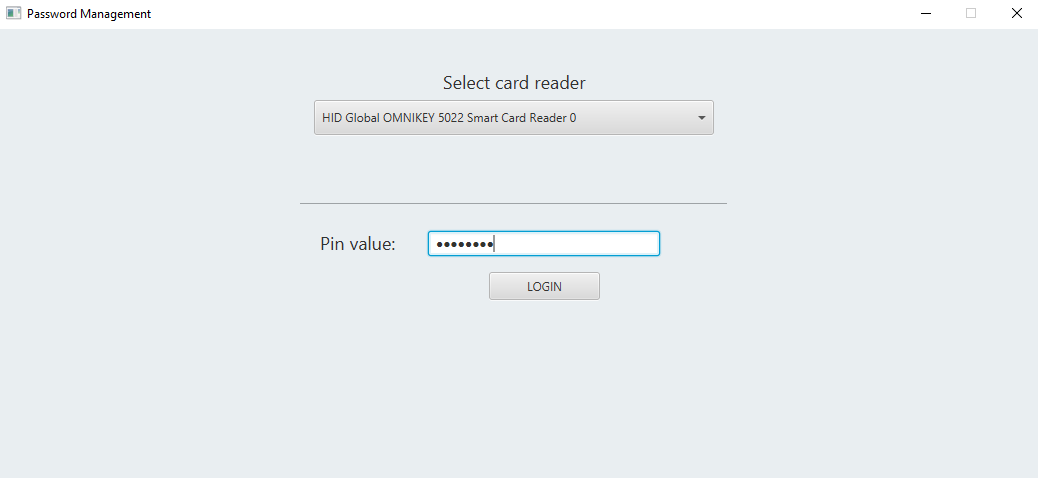
public void getUserAccountDataFromCard(UserAccountMessage userAccountMessage) throws CardException, IllegalBlockSizeException,  
 InvalidKeyException, BadPaddingException, NoSuchAlgorithmException, NoSuchPaddingException {  
 try {  
 userAccountMessage.setStatusCode(StatusCode.*GenericError*);  
 userAccountMessage.setUserAccountList(null);  
 Card connection = cardTerminal.connect("T=1");  
 CardChannel cs = connection.getBasicChannel();  
 CommandAPDU commandAPDU = new CommandAPDU(0x0C, 0xCC, 0x00, 0x00);  
 ResponseAPDU responseAPDU = cs.transmit(commandAPDU);  
 if (responseAPDU.getSW() != 0x9000) {  
 if (responseAPDU.getSW() == 0x6982) {  
 userAccountMessage.setStatusCode(StatusCode.*UnauthorizedUser*);  
 throw new CardException("Unauthorized user!");  
 } else if (responseAPDU.getSW() == 0x6F00) {  
 userAccountMessage.setStatusCode(StatusCode.*CardInternalError*);  
 throw new CardException("Something went wrong! Internal Error!");  
 } else {  
 userAccountMessage.setStatusCode(StatusCode.*CardGenericError*);  
 throw new CardException(responseAPDU.toString());  
 }  
 }  
 byte[] dec = decrypt(responseAPDU.getData(), this.secretKey);  
 int len = (dec[0] & 0xff) \* 256 + (dec[1] & 0xff);  
 if (len == 0)  
 return;  
 byte[] data = new byte[len];  
 System.*arraycopy*(dec, 2, data, 0, len);  
 int lenUserAccountListData = (data[0] & 0xff) \* 256 + (data[1] & 0xff);  
 byte[] userAccountListByte = new byte[lenUserAccountListData];  
 if (lenUserAccountListData > data.length) {  
 userAccountMessage.setStatusCode(StatusCode.*WrongLength*);  
 throw new CardException("Wrong additional data format on card!");  
 }  
 System.*arraycopy*(data, 2, userAccountListByte, 0, lenUserAccountListData);  
 String s = new String(userAccountListByte);  
 userAccountMessage.setUserAccountList(s);  
 userAccountMessage.setStatusCode(StatusCode.*OK*);  
 } catch (CardException ex) {  
 Logger.*getLogger*(SmartCardCommunication.class.getName()).log(Level.*SEVERE*, null, ex);  
 throw new CardException(ex.getMessage());  
 }

}

*Кôд који представља методу getUserAccountDataFromCard*

### Опис графичког корисничког интерфејса

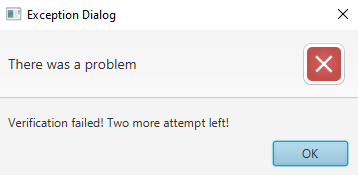
Графичко кориснички интерфејс апликације се састоји из два екрана. Први представља почетну форму за пријављивање на систем, док друга представља главни екран апликације где корисник има могућност манипулације са подацима, обраду, приказивање и смештање података на паметну картицу. На слици 9.2 приказан је почетни екран програма, који представља форму за пријављивање на систем.



*Слика 9.2 Изглед форме за пријаву на систем*

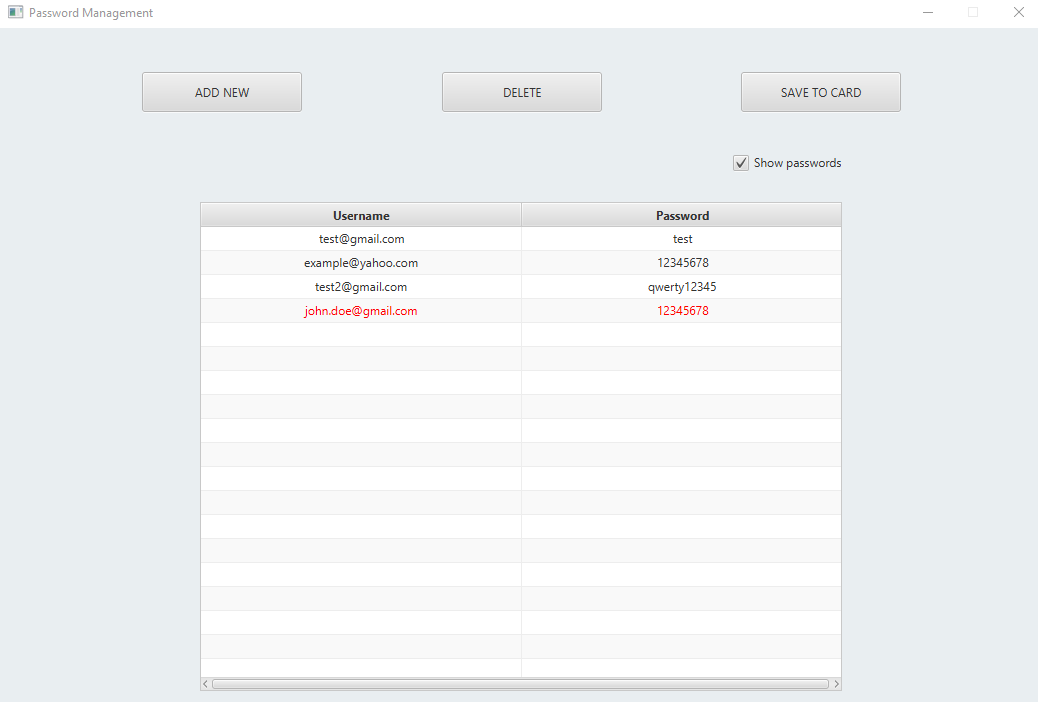
Корисник прво бира из падајућег менија читач картица који директно преноси команде на паметну картицу. Падајући мени је изабран јер корисник може имати више читача картица прикључених у рачунар.

Након избора читача, корисник укуцава вредност пин-а и кликом на дугме Login, покреће процедуру за проверу пин-а на паметној картици. Уколико је пин погрешан, кориснику ће изаћи посебан прозор (слика 9.3) где ће бити обавештен да је пин погрешан и приказаће му се колико још покушаја има за пријаву на систем. Иницијално корисник има три покушаја за пријаву на систем. Ако три пута унесе погрешан пин, паметна картица ће се закључати, и њено даље коришћење неће бити могуће без слања посебне команде за откључавање пин-а на паметној картици. Уколико је пин тачан, апликација прелази на главни екран.



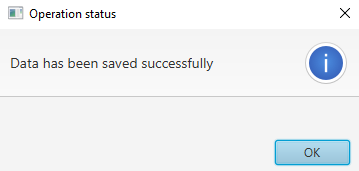
*Слика 9.3 Прозор који обавештана корисника о погрешном пин-у и броју преосталих покушаја за пријаву на систем*

На слици 9.4 приказан је главни екран апликације. Екран се састоји од три дугмета, једног checkbox-a и листе за приказ података.



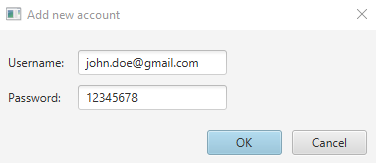
*Слика 9.4 Главни екран апликације*

Корисник има увид у све податке који су сачувани на паметну картицу и ти подаци су приказани у листи на главном екрану. Checkbox Show passwords служи за већу безбедност самог приказа података о лозинкама. Кликом на то поље, кориснику се уместо вредности лозинки приказују звездице и на тај начин постају нечитљиве. Кликом на дугме SAVE TO CARD остварује се комуникација са паметном картицом и подаци се APDU командама, преко унапред успостављеног сигурног канала комуникације, смештају на паметну картицу. Уколико се подаци успешно сместе на паметну картицу, кориснику ће изаћи обавештење у виду новог прозора, који је приказан на слици 9.5.

. 

*Слика 9.5 Обавештење о успешно смештеним подацима на паметну картицу*

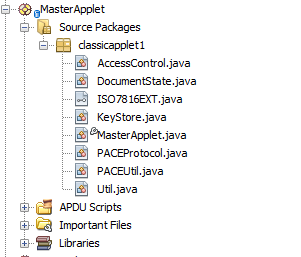
Додавање новог корисничког имена и лозинке се дешава кликом на дугме ADD NEW. У том тренутку, кориснику ће изаћи нови екран који је приказан на слици 9.6 и на коме корисник може да унесе ново корисничко име и лозинку. Кликом на дугме OK, корисник ће у листи на главном екрану видети додато ново корисничко име и лозинку, при чему ће тај податак бити црвене боје. Ново корисничко име и лозинка су бити црвене боје јер представљају податак који није смештен на паметну картицу. Успешним смештањем података на картицу, црвена боја прелази у црну као и сви остали подаци.



*Слика 9.6 Додавање новог корисничког имена и лозинке*

## Програм паметне картице

Програм који се извршава на паметној картици је програм написан у Јава програмском језику, користећи JavaCard технологију. Пројекат је урађен у развојном окружењу NetBeans 8.2. Пројекат садржи 1 пакет и 8 класа. На слици 9.7 приказана је структура пројекта паметне картице. Паметна картица преко APDU протокола може да прима команде са Терминала, извршава функције и шаље одговоре Терминалу.



*Слика 9.7 Структура пројекта паметне картице*

Главна класа програма је класа MasterApplet. У њој се налазе дефиниције свих функција које паметна картица може да изврши. У наставку ће бити приказан кôд појединих функција програма. У конструктору MasterApplet класе се извршава иницијализација свих променљивих потребних за даљи рад програма.

protected MasterApplet() {  
 transientBytes = JCSystem.makeTransientByteArray(NUM\_TRANSIENT\_BYTES,  
 JCSystem.CLEAR\_ON\_RESET);  
 transientShorts = JCSystem.makeTransientShortArray(NUM\_TRANSIENT\_SHORTS,  
 JCSystem.CLEAR\_ON\_RESET);  
 transientObjects = JCSystem.makeTransientObjectArray(NUM\_TRANSIENT\_OBJECTS,  
 JCSystem.CLEAR\_ON\_RESET);  
 temp = JCSystem.makeTransientByteArray(TEMP\_SIZE, JCSystem.CLEAR\_ON\_DESELECT);  
 accessControl = new AccessControl();  
 paceProtocol = new PACEProtocol(temp);  
 locked = false;  
  
 documentNumber = new byte[8];  
 *// FOR AES Keys* A = new byte[PACE\_KEY\_LENGTH];  
 a = new byte[PACE\_KEY\_LENGTH];  
 B = new byte[PACE\_KEY\_LENGTH];  
 param\_K = new byte[PACE\_KEY\_LENGTH];  
  
 userAccountData = new byte[2000];  
}

*Кôд који представља конструктор класе MasterApplet*

Process метода је главна метода програма у којој се обрађује APDU команда и одређује даљи ток програма. На основу параметра ins, програм извршава функцију и шаље одговор Терминалу. Програм садржи 6 функција које служе за успостављање сигурног комуникационог канала, верификацију пин-а, смештање података о корисничким налозима и лозинкама у меморију картице, као и читање података из меморије.

public void process(APDU apdu) {  
 if (this.selectingApplet()) {  
 return;  
 }  
 byte[] buffer = apdu.getBuffer();  
 byte p2 = buffer[OFFSET\_P2];  
 transientBytes[TN\_CURRENT\_CLA] = buffer[OFFSET\_CLA];  
 byte ins = buffer[OFFSET\_INS];  
 try {  
 switch (ins) {  
 case (byte)0x20:  
 processVerify(apdu);  
 break;  
 case (byte)0x2C:  
 processResetRetryCounter(apdu);  
 break;  
 case (byte) 0xDC:  
 processPutUserAccountlData(apdu);  
 break;  
 case (byte) 0xCC:  
 processGetUserAccountData(apdu);  
 break;  
 case (byte) 0x88:  
 generateKey(apdu);  
 break;  
 case (byte) 0x89:  
 validateKey(apdu);  
 break;  
 default:  
 ISOException.throwIt(SW\_INS\_NOT\_SUPPORTED);  
 }  
 } catch (CardRuntimeException ex) {  
 this.resetChaining();  
 throw ex;  
 }  
}

*Кôд који представља Process методу*

Метода generateKey и validateKey служе за успостављање сигурног комуникационог канала. У методи generateKey се креира тајни кључ, који се потом валидира у методи validateKey.

private void generateKey(APDU apdu) {  
 byte[] buffer = apdu.getBuffer();  
 RSAPrivateKey secret\_a = (RSAPrivateKey) KeyBuilder.buildKey(KeyBuilder.TYPE\_RSA\_PRIVATE, KeyBuilder.LENGTH\_RSA\_1024, false);  
 RandomData randomData = RandomData.getInstance(RandomData.ALG\_SECURE\_RANDOM);  
 randomData.generateData(a, (short) 0, (short) 128);  
 secret\_a.setExponent(a, (short) 0, (short) PACE\_KEY\_LENGTH);  
 secret\_a.setModulus(param\_p, (short) 0, (short) PACE\_KEY\_LENGTH);  
 Cipher cipher = Cipher.getInstance(Cipher.ALG\_RSA\_NOPAD, false);  
 cipher.init(secret\_a, Cipher.MODE\_DECRYPT);  
 cipher.doFinal(param\_g, (short) 0, PACE\_KEY\_LENGTH, A, (short) 0);  
 short keyStart = OFFSET\_CDATA;  
 if (buffer[(short) (keyStart)] == (byte) 0x00)  
 keyStart++;Util.arrayCopyNonAtomic(buffer, keyStart, B, (short) 0, (short) PACE\_KEY\_LENGTH);cipher.doFinal(B, (short) 0, PACE\_KEY\_LENGTH, param\_K, (short) 0);Util.arrayCopyNonAtomic(param\_K, (short) 0, buffer, (short) 0, (short) PACE\_KEY\_LENGTH);  
 deriveKey(buffer, (short) 0, (short) PACE\_KEY\_LENGTH, (byte) 0x01, (short) 0);  
 paceProtocol.setEncKey(buffer, (short) 0);  
 apdu.setOutgoing();  
 apdu.setOutgoingLength((short) A.length);  
 apdu.sendBytesLong(A, (short) 0, (short) A.length);  
}

*Кôд који представља методу generateKey*

Метода validateKey служи за валидацију израчунатог тајног кључа према протоколу за успостављање сигурног комуникационог канала. Представља последњи корак у успостави сигурног комуникационог канала између Терминала и паметне картице. Метода дешифрује примљену вредност са терминала и потом је шифрује својим тајним кључем. Добијени шифрат потом шаље терминалу.

private void validateKey(APDU apdu) {  
 byte[] buffer = apdu.getBuffer();  
 short byteRead = apdu.setIncomingAndReceive();  
  
 paceProtocol.decryptPACE(buffer, OFFSET\_CDATA, byteRead, temp, (short) 1);  
  
 short lc = Util.makeShort((byte) 0x00, temp[1]);  
  
 temp[0] = (byte) ((lc >> 0x08) & 0xff);  
 temp[1] = (byte) (lc & 0xff);  
 short le = paceProtocol.encryptPACE(temp, (short) 0, (short) (lc + 2),  
 buffer, (short) 0);  
  
 apdu.setOutgoing();  
 apdu.setOutgoingLength(le);  
 apdu.sendBytesLong(buffer, (short) 0, le);  
}

*Кôд који представља методу validateKey*

Метода processVerify служи за валидацију унетог пин-а на паметној картици. Пин који је сетован у процесу персонализације паметне картице се налази у сигурној меморији картице. Корисник уноси вредност пин-а и пин се претходно успостављерним сигурним комуникационим каналом шаље на картицу, где се методом processVerify валидира. Метода прво дешифрује шифрат и добијену вредност упоређује са пин-ом у сигурној меморији. Уколико вредности нису једнаке, паметна картица враћа одговор и податак о томе колико је још покушаја валидације кориснику остало на располагању. Уколико је пин исправан, картица шаље статус о успешном логовању. Уколико корисник 3 пута за редом унесе погрешан пин, паметна картица ће се закључати. Откључавање картице је могуће позивом методе processResetRetryCounter.

private void processVerify(APDU apdu) {  
 byte[] buffer = apdu.getBuffer();  
 byte byteRead = (byte) apdu.setIncomingAndReceive();  
 if (locked) {  
 short le = paceProtocol.decryptPACE(buffer, OFFSET\_CDATA, byteRead,  
 buffer, (short) (OFFSET\_CDATA - 1));  
 byteRead = buffer[(short) (OFFSET\_CDATA - 1)];  
 }  
 if (buffer[OFFSET\_P1] != 0x00) {  
 ISOException.throwIt(SW\_WRONG\_P1P2);  
 }  
 byte p2 = buffer[OFFSET\_P2];  
 OwnerPIN selectedPin = accessControl.getPin(p2);  
 if (selectedPin == null) {  
 ISOException.throwIt(SW\_WRONG\_P1P2);  
 }  
 byte remaining = selectedPin.getTriesRemaining();  
 if (byteRead > 0) {  
 if (remaining == 0x00) {  
 ISOException.throwIt(ISO7816EXT.SW\_AUTHENTICATION\_METHOD\_BLOCKED);  
 }  
 if (selectedPin.check(buffer, OFFSET\_CDATA, byteRead) == false) {  
 remaining = selectedPin.getTriesRemaining();  
 ISOException.throwIt((short) (ISO7816EXT.SW\_VERIFICTION\_FAILED | remaining));  
 }  
 } else {  
 ISOException.throwIt(SW\_SECURITY\_STATUS\_NOT\_SATISFIED);  
 }  
}

*Кôд који представља методу processVerify*

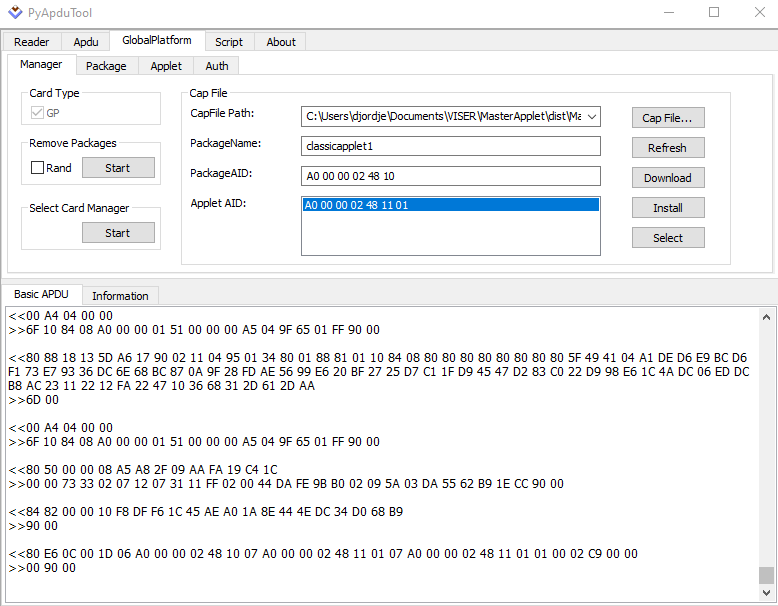
Метода processPutUserAccountData је метода која служи за смештање података о корисничким налозима и лозинкама у сигурну меморију паметне картице. Да би се ова метода извршила, неопходно је пре свега успоставити сигуран комуникациони канал између Терминала и паметне картице. Након тога, корисник мора унети одговарајући пин и валидирати га на паметној картици. Метода processPutUserAccountData прво проверава да ли је успостављен сигурни комуникационо канал, затим проверава да ли је пин успешно валидиран на паметној картици. Уколико су ова два услова задовољена, картица дешифрује шифрат добијен од стране Терминала. Успешно дешифроване податке, картица смешта у заштићену меморију и потом шаље одговор Терминалу да су подаци успешно смештени на паметну картицу. Подаци се смештају као низ бајтова, чијим декодовањем можемо да извучемо податке о корисничким налозима и лозинкама. Максимална дужина података који могу да се сместе на паметну картицу је фиксне дужине и износи 2000 бајтова.

private void processPutUserAccountlData(APDU apdu) {  
 byte[] buffer = apdu.getBuffer();  
 short len = apdu.setIncomingAndReceive();  
 short offset = apdu.getOffsetCdata();  
 short total = apdu.getIncomingLength();  
 short recieved = len;  
 Util.arrayCopyNonAtomic(buffer, offset, temp, (short) 0, len);  
 while (recieved < total) {  
 len = apdu.receiveBytes((short) 0);  
 if (len == 0) {  
 break;  
 }  
 Util.arrayCopyNonAtomic(buffer, (short) 0, temp, recieved, len);  
 recieved += len;  
 }  
 if(!locked) {  
 ISOException.throwIt(ISO7816.SW\_CONDITIONS\_NOT\_SATISFIED);  
 }  
 if (!accessControl.checkIsMasterPinVerified()) {  
 ISOException.throwIt(ISO7816.SW\_SECURITY\_STATUS\_NOT\_SATISFIED);  
 }  
 Util.arrayCopyNonAtomic(temp, (short) 0, temp, (short)16, recieved);  
 paceProtocol.decryptPACE(temp, (short) 16, recieved, temp, (short) 0);  
 len = Util.makeShort(temp[0], temp[1]);  
 if (len >= (short) userAccountData.length) {  
 ISOException.throwIt(ISO7816.SW\_WRONG\_LENGTH);  
 }  
 Util.arrayFillNonAtomic(userAccountData, (short) 0, (short) userAccountData.length, (byte) 0);  
 Util.arrayCopyNonAtomic(temp, (short) 2, userAccountData, (short) 0, len);  
}

*Кôд који представља методу processPutUserAccountData*

## Програм за персонализацију паметне картице

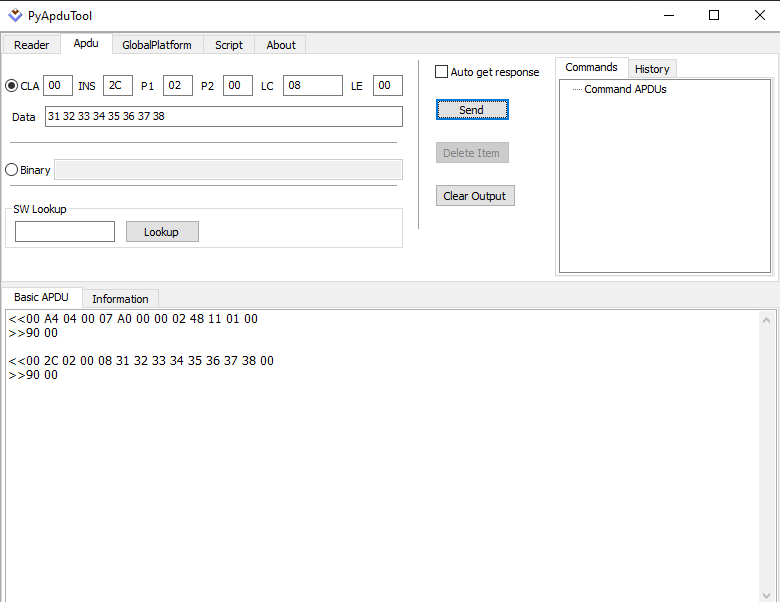
Софтверско решење за инсталирање аплета на паметну картицу и сетовање пин-а представља софтверско решење под називом PyApduTool. PyApduTool је програм отвореног кода који служи за повезивање са паметном картицом, комуникацију са паметном картицом, управљање пакетима и аплетима, интеграционо тестирање аплета на картици итд. Програм се може бесплатно преузети са интернета и није потребна никаква лиценца за његово коришћење. На слици 9.8 приказан је изглед програма са делом за спуштање аплета на паметну картицу.



*Слика 9.8 Део програма за спуштање аплета на паметну картицу*

Кликом на дугме Cap File корисник селектује cap датотеку аплета који се спушта на паметну картицу. Затим кликом на дугме Download, програм учитава cap датотеку и шаље одговарајуће команде на паметну картицу. Кликом на дугме Install програм шаље команду за инсталирање аплета и тиме се завршава процес спуштања аплета на паметну картицу. У секцији Basic APDU корисник може имати увид у све команде које се шаљу на картицу, као и одговоре које картица шаље програму.

На слици 9.9 приказан је део програма у ком корисник може ручно да уписује команде и шаље их на паметну картицу. Конкретна команда која је приказана је команда за сетовање пин-а. Корисник прво селектује аплет и потом уноси команду која садржи вредност пин-а у пољу Data. Кликом на дугме Send корисник шаље команду и очекује одговор од паметне картице. Уколико одговор има вредност 9000, пин је успешно сетован.



*Слика 9.9 Део програма за слање команди на паметну картицу*

# Закључак

У овом мастер раду имплементиран је систем за сигурно чување лозинки на паметним картицама. Представљено је решење проблема сигурне комуникације, сигурног чувања поверљивих података и њиховог читања. Реализована су два одвојена програма, програм Терминал и програм паметне картице. Пре саме реализације софтверског решења, појединачно је дат теоријски увид у сваки део система због бољег разумевања функционисања самог система.

Изнете су основне чињенице зашто је неопходно имати сигуран комуникациони канал за пренос поверљивих података. Дат је опис основних карактеристика паметних картица и разлози зашто су оне добар медијум за чување поверљивих података. С обзиром на то да је у имплементацији решења за сигурно чување лозинки изабрана паметна картица која подржава JavaCard технологију, описана је сама JavaCard технологија и њена архитектура. Описан је начин комуникације између терминала и паметне картице. Након тога, дат је увид у основне криптографксе појмове, типове криптографија и набројани су основни алгоритми шифровања. Посебан фокус је дат на AES алгоритму шифровања јер се тај алгоритам користио у самој реализацији софтверског решења. Описан је Дифи-Хелманов алгоритам за размену кључева јер се протокол за успоставу сигурног комуникационог канала ослањао управо на овај алгоритам. Узимајући у обзир то да је софтверско решење писано у Java програмском језику, тај језик је укратко описан и набројане су његове главне карактеристике. Описан је протокол за успоставу сигурног комуникационог канала између Терминала и паметне картице као део комплетног система. Приказан је секвенцијални дијаграм тока протокола и објашњен је сваки корак протокола.

Након теоријских појмова описана је и практична реализација софтверског решења комплетног система. Прво је дат опис самог решења и начин коришћења. Потом је описан пројекат Терминал, приказан је дизајн програма, начин израде и изворни кôд појединих функционалности програма. Поред тога, приказан је и графичко-кориснички интерфејс програма са описом. Након програма Терминала, описан је програм на паметној картици. Описан је начин реализације, дизајн програма као и изворни кôд. Након Терминала и програма на паметној картици, описан је и програм за персонализацију паметне картице. Имајући у виду да је у питању готово решење, описано је његово коришћење и намена.

С обзиром на то да се област заштите података, криптографије и криптоанализе константо развија, са развојем технологије постоји могућност за унапређење система у виду коришћења алгоритама отпорнијих на нападе, као и коришћења паметних картица јачих хардверских карактеристика.

# Литература

[1] Драган Плескоњић, Немања Мачек, Марко Царић, Борислав Ђорђевић „*Сигурност рачунарских система и мрежа*“, Микро књига, Београд, 2007

[2] Wolfgang Rankl, Wolfgang Effing „*Smart Card Handbook: Fourth Edition*“, John Wiley & Sons, 2010

[3] Java Card Platform Virtual Machine Specification, Classic Edition Version 3.1

[4] Zhiqun Chen „*Java Card Technology for Smart Cards*“, O’Reilly, 2000

[5] Wolfgang Rankl „*Smart Card Applications*“, John Wiley & Sons, 2007

[6] Massimo Bertaccini „*Cryptography Algorithms*“, Pack Publishing, 2022

[7] <https://docs.oracle.com/en/java/>

# Индекс појмова

AES 15, 17, 18, 19, 23

APDU 11, 12, 13, 25, 30, 36, 37, 38, 43

CAP 9, 10

JavaCard 8, 9, 10, 26, 37, 44

JavaFX 21, 22, 26, 27, 28

Tерминал 26

Алгоритам 15, 20

аплет 9, 10, 12, 43

Јава 8, 9, 10, 21, 22, 37

картица 7, 10, 11, 12, 13, 14, 25, 26, 32, 35, 37, 38, 40, 41, 43, 44

кључ 15, 16, 19, 20, 25, 31, 32, 39

криптографија 15, 16, 44

лозинка 37

протокол 11, 25, 31, 44

систем 3, 5, 7, 9, 22, 26, 27, 34, 35, 44

**ИЗЈАВА О АКАДЕМСКОЈ ЧЕСТИТОСТИ**

|  |  |
| --- | --- |
| **Студент (име, име једног родитеља и презиме):** | **Ђорђе, Драган, Миленковић** |
| **Број индекса:** | **РИН-65/17** |

Под пуном моралном, материјалном, дисциплинском и кривичном одговорношћу изјављујем да је мастер рад, под насловом:

„Једна имлементација система за чување лозинки на паметним картицама“

1. резултат сопственог истраживачког рада;
2. да овaj рад, ни у целини, нити у деловима, нисам пријављиво/ла на другим високошколским установама;
3. да нисам повредио/ла ауторска права, нити злоупотребио/ла интелектуалну својину других лица;
4. да сам рад и мишљења других аутора које сам користио/ла у овом раду назначио/ла или цитирао/ла у складу са Упутством;
5. да су сви радови и мишљења других аутора наведени у списку литературе/референци који је саставни део овог рада, пописани у складу са Упутством;
6. да сам свестан/свесна да је плагијат коришћење туђих радова у било ком облику (као цитата, прафраза, слика, табела, дијаграма, дизајна, планова, фотографија, филма, музике, формула, вебсајтова, компјутерских програма и сл.) без навођења аутора или представљање туђих ауторских дела као мојих, кажњиво по закону (Закон о ауторском и сродним правима), као и других закона и одговарајућих аката Високе школе електротехнике и рачунарства струковних студија у Београду;
7. да је електронска верзија овог рада идентична штампаном примерку овог рада и да пристајем на његово објављивање под условима прописаним актима Високе школе електротехнике и рачунарства струковних студија у Београду;
8. да сам свестан/свесна последица уколико се докаже да је овај рад плагијат.

У Београду, \_\_. \_\_. 202\_. године

Својеручни потпис студента

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_