АКАДЕМИЈА ТЕХНИЧКО-УМЕТНИЧКИХ СТРУКОВНИХ СТУДИЈА БЕОГРАД

ОДСЕК ВИСОКА ШКОЛА ЕЛЕКТРОТЕХНИКЕ И РАЧУНАРСТВА

**Миленковић Ђорђе**

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

**- мастер рад -**



Београд, ххх 2022.

Кандидат: **Миленковић Ђорђе**

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 истраживанја**

**2. Пројектовање система за чување лозинки**

**3. Имплементација система за чување лозинки**

Београд, ххх 2022. године. Ментор:

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

Др Немања Мачек, проф. ВИШЕР

**РЕЗИМЕ:**

ХХХХХХХХХХХХХХХххххххххххххххххххххххххххххххххххххххххххххххххххххххххххххх

**Кључне речи:** ХХХХХХххххххххххххххххххх

**ABSTRACT:**

Ххххххххххххххххххххххххххххххххххххххххххххххххххххххххххххххххххххххххххххххххх

**Key words:** Хххххххххххххххххххххх

САДРЖАЈ

1. Uvod
2. Opis smart kartica
3. JavaCard
4. Uvod u kriptografiju
5. Simetricni algoritmi
6. AES algoritam
7. Difi Helman
8. Java programski jezik
9. JavaFX
10. Implementirano resenje

# 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.

Главна предност смарт 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 смарт картици прочитали, неопходно је успоставити сигуран канал комуникације између картице и терминала.

Циљ овог рада је приказати један систем за сигурно чување поверљивих података као што су кориснички налози и њихове лозинке на паметним картицама.

# 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, 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 у епокси смолом пре уграђивања у пластику картице.

Пошто паметне картице садрже централну процесорску јединицу (ЦПУ), 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 (ЦПУ) –** Централна процесорска јединица чипа паметне картице тумачи и извршава инструкције добијене од оперативног система.
* **Реад Онли Меморија (РОМ) –** Реад онлз меморија складишти податке који се, једним уписани, не могу променити или обрисати. Из тог разлога, произвођачи чипова паметних картица у РОМ меморији чувају оперативни систем картице, као и различите функције тестирања и дијагностике.
* **Рандом аццесс меморз (РАМ) –** РАМ 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 (ЕЕПРОМ) –** За податке који требају бити измењени или обрисани, као што су подаци у апликацији, произвођачи чипова користе избрисиву програмабилну меморију само за читање (ЕЕПРОМ). ЕЕПРОМ 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 АПИ за приступ картици и укључује неке криптографске функције. Други део специфицира Јава Цард Виртуалну Машину (ЈЦВМ) која је дизајнирана као једноставна Јава виртуелна машина у коју је уграђен подскуп Јава програмског језика. Због свог малог меморијског простора, Јава Цард 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 неке карактеристике Јава језика прилагођеног за паметне картице и Јава језика опште намене.

|  |  |  |
| --- | --- | --- |
| **Функција** | **Јава Цард** | **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 |

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

## 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, Конвертер 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 3.1 и 3.2.

  
Слика 3.1. JavaCard Платформа

  
Слика 3.2. JavaCard Платформа

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 .цласс 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, .цласс датотеке се конвертују у CAP (конвертовани аплет) датотеку помоћу JavaCard конвертора.

JavaCard конвертор прима .цласс 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 или библиотека. Поред тога, пакети аплета или библиотека у JavaCard CAP фајлу могу бити јавни или приватни.

Поред .цласс 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 бајткод.

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 ембеддед система, ЈаваЦард аплети не морају да се учитавају у РОМ меморију.

Главна класа аплета мора наследити класу javacard.framework.Applet класу. Основна класа Апплет 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.

ЈаваЦард 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 ИСО/ИЕЦ 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 пројемник. Овај процес наитменичног преноса и пројема назива се полу-дуплекс комуникација. Потпуна дуплекс комуникација, у којој обе стране могу истовремено да преносе и примају податке, тренутни 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 мастер – слејв, са терминалом као мастер и картицом као слејв. Након што је паметна картица обрадила 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 4.1.

  
Слика 4.1. Стање паметне картице током активирања и комуникације са терминалом

## АПДУ протокол

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

Као што је приказано на слици 4.2. АПДУ 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 (ЦЛА), бајта инструкције (ИНС) и два бајта параметра (П1 и П2). Бајт класе се такође користи за идентификацију апликација. Бајт класе се може користити и као идентификатор да су подаци који се шаљу АПДУ командом шифровани.



Слика 4.2. Структура АПДУ команде

Други бајт у заглављу АПДУ команде је бајт инструкције (ИНС). Тај бајт означава функцију коју аплет треба да изврши. Два бајта параметра (П1 и П2) примарно се користе за пружање више информација за функцију изабрану бајтом инструкције. На пример једна функција може у зависности од прослеђених параметара П1 и П2 уради једну или другу ствар. Након заглавља следи тело АПДУ команде које може бити празно изузев параметра за дужину тела поруке и параметра за дужину података који се враћају са картице. Параметар за дужину (ЛЦ поље) одређује дужину података који ће бити послати АПДУ командом. Параметар за дужину (ЛЕ 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 ЛЕ поља „00“, терминал не очекује да ће картица вратити податке.

Поља ЛЦ и ЛЕ 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 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 АПДУ команде се могу комбиновати са 4 случаја који су приказани на слици 4.3.

  
Слика 4.3. Четири могуће варијанте АПДУ команде

## Структура одговора са паметне картице

АПДУ одговор који картица шаље као одговор на АПДУ наредбу се састоји од обавезног заглавља и опционог тела, као што је приказано на слици 4.2.1.

  
Слика 4.2.1. Структура АПДУ команде као одговор са паметне картице

Тело одговора се састиоји од поља података чија је дужина одређена ЛЕ пољем претходне АПДУ команде. 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 СВ1 и СВ2 која су приказана на слици 4.2.2.



Слика 4.2.2. Две варијанте структуре одговора са паметне картице ка терминалу

У колико је вредност СВ1 и СВ2 9000, картица је успешно извршила своју функцију и послала терминалу одговор. Ако се након извршене наредбе прими повратни статус код „63џџ“ или „65џџ“ то значи да је меморија на картици (ЕЕПРОМ или флеш меморија) измењена. Ако је примљен било који други повратни код који започиње са „6џ“, то значи да је извршење програма прерано окончану, без промена у меморији. ИСО7816 стандард дефинише и одређене изузетке који се могу догодити током извршења програма на картици. У табели 4.2.3. приказани су неки од њих.

|  |  |
| --- | --- |
| 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 П1 и П2 параметри |
| 6А 82 | Датотека на картици није пронађена |

Табела 4.2.3. Изузеци дефинисани ИСО7816 стандардом.

# Основни криптографски појмови

Реч криптографија води порекло од грчких речи „kriptos“ што значи скривено и „графос“ 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 п производи шифрат ц. Функција дешифровања Д на основу истог кључа к и шифрата ц производи оригиналну поруку п. Симетрични алгоритми су брзи и као такви се могу користити за шифровање већих датотека или имплементацију у крипто системе датотека. Најпознатији су ДЕС (Дата Енцрyптион Стандард), AES (Адванцед Енцрyптион Стандард), ИДЕА (Интернатионал Дата Енцрyптион Алгоритхм), Блоwфиш, Тwофиш 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 РСА и ЕлГамал.

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 ма 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 МД5 и ША1. Приликом потписивања, пошиљалац најпре једносмерном хеш функцијом рачуна хеш х1 поруке п, коју после тога потписује својим приватним кључем (условно се може шватити 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 шватити 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, односно 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. У колико 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 бруте-форце аттацк), 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 Ри и њему припадајућег шифрата Ци, не сме да омогући да се из неког другог блока шифрата Цј одреди блок отвореног текста Пј. 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 се крајем деведесетих година утврдило да постојећи симетрични алгоритми нису безбедни, тражило се ново решење за стандардни блоковски алгоритам. 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 5.3.1 је шематски приказана трансформација.



Слика 5.3.1. SubBytes функција AES алгоритма

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 5.3.2. је приказан поступак цикличног померања.

  
Слика 5.3.2. ShiftRows функција AES алгоритма

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 дат на слици 5.3.4.

  
Слика 5.3.4. MixColumn функција AES алгоритма

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 5.3.5.

   
Слика 5.3.5. AddRoundKey функција AES алгоритма

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, односно 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

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 Ц++, који је директан потомак језика Ц. Већи део својих особина Јава је наследила од ова два језика. Из језика Ц 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 (*енгл. Java Virtual Machine*). Машински језик за Јава Виртуалну Машину се зове Јава бајткод (*енгл. Java bytecode*).

## JavaFX платформа

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 runtime укључује JavaFX Desktop, JavaFX mobile i JavaFX TV. У позадини JavaFX-а су бројне компоненте које коришћењем постојећих библиотекадефинишу функционалност за креирање графичко корисничког интерфејса. На слици 6.1. је приказана архитектура JavaFX технологије.

  
Слика 6.1. JavaFX архитектура

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 АПИ-ја. Елемент сцене се назива чвор. Сваки чвор има свој јединствени ID 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 АПИ-јима. Оне омогућавају довољну слободу при развоју апликације са богатим графичким окружењем. JavaFX омогућује коришћење моћних функционалности Јава језика, као што су генерички типови, анотације, подршка за вишенитно програмирање итд.

Да би се JavaFX сцена приказача 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 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 7.1. 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 7.1. 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 АПДУ 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 Диги – 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 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 ххх, и читач картица произвођача Omnikey, модел Omnikey 5522. Програм који се користио за инсталирање програма на паметну картицу је JCShell Tool од произвођача NXP.

## Програм терминала

Терминал је програм који се извршава на персоналном рачунару и који комуницира са паметном картицом. 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. На слици 8.1.1. приказана је структура пројектa Tерминал.



*Слика 8.1.1. 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 појединих функционалности програма.

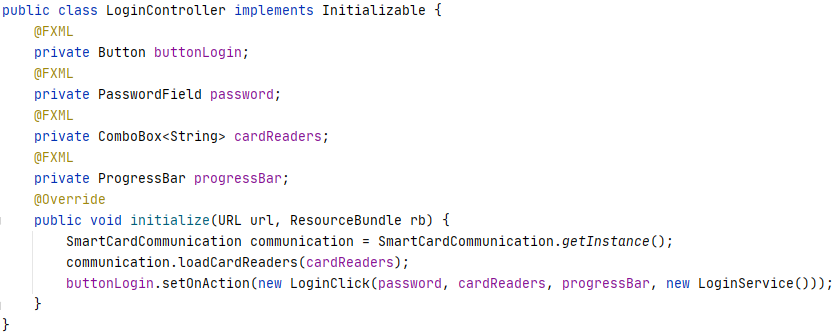
Класа Main представља главну класу програма и од ње почиње извршавање комплетног програма. На слици 8.1.2 налази се код методе класе Main.



*Слика 8.1.2. Покретање програма и извршавање main методе*

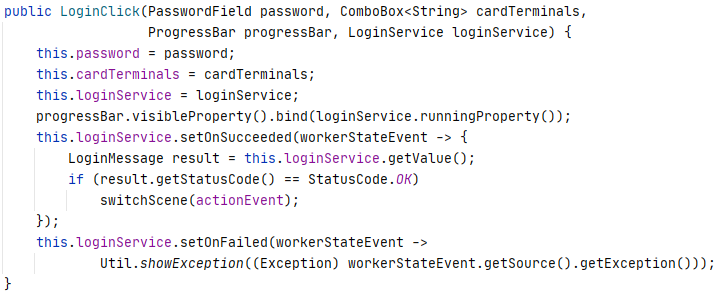
Извршавање програма Терминал почиње методом main која позива методу launch. Launch метода извршава сву потребну иницијализацију за покретања програма JavaFX технологије. Метода start се позива након што је извршена сва потребна иницијализација и у њој се дефинишу својства за главну сцену апликације. У наставку ће бити описан кôд програма за извршавање операције пријављивања на систем и дохватања свих података са паметне картице.

На слици 8.1.3. приказана је класа LoginController која служи за иницијализацију свих елемената на форми која служи за пријављивање корисника.



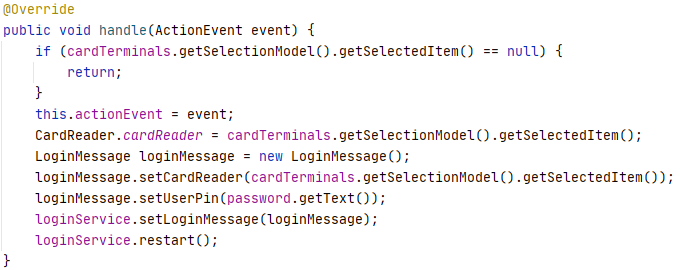
*Слика 8.1.3. Класа LoginController*

Класа која служи за обрађивање догађаја приликом корисниковог клика на дугме Login је класа LoginClick. На слици 8.1.4. приказан је конструктор класе LoginClick. У конструктору се инцијализује сервис коме припада логика операције за пријављивање и покривање случаја ако је пријављивање успешно или неуспешно.



*Слика 8.1.4. Конструктор класе LoginClick*

Метода handle из класе LoginController представља методу која се извршава на клик дугмета Login на форми за пријављивање. Метода покреће нову програмску нит из класе LoginService у којој се одвија комуникација са паметном картицом. Метода handle се налази на слици 8.1.5.



*Слика 8.1.5. Метода 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. На слици 8.1.6. налази се кôд класе LoginService.



*Слика 8.1.6. Класа LoginService*

Метода createTask користи инстанцу класе SmartCardCommunication за позивање методе establishSecureChannel и потом методе verifyPin.