ВИСОКА ШКОЛА ЕЛЕКТРОТЕХНИКЕ И РАЧУНАРСТВА СТРУКОВНИХ СТУДИЈА

**Анђелковић Милош**

**Програмско решење за шифроваање кратких порука**

**-завршни (специјалистички) рад-**



Београд, окотобар 2016.

Кандидат: **Анђелковић Милош**

Број индекса: **НРТ-35/13**

Студијски програм: **Нове рачунарске технологије**

Тема: **Програмско решење за шифровање кратких порука**

Основни задаци:

**1.Обајснити приницпе шифровања и стандардене крипотграфске алгоритме**

**2. Дефинисати интернет окружење за размену кратких порука између више корисника, алгоритме заштите који се примењују и начин размене кључева**

**3. Практична реализација програма и анализа**

Београд, септембар 2016. Ментор:

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

др Зоран Бањац, проф. ВИШЕР

**РЕЗИМЕ:**

У дипломском раду су описани принципи шифорвања и неки од стандардних криптографских алгоритама. Описани си неки од алгоритама који обезбеђују сигурност у комуникацији који су коришћени у изради практичног дела. У практичном делу дипломског су описани алати и развојно окружење коришћени за израду апликације као и сама апликација.

**Kључне речи:** критографија, сигурност, комуникација, апликација

**АBSTRACT:**

In this paper principles of encrypting and some of the standard cryptographic algorithms are described. Some of the algorithms that provide securyti in communication that are used in the practical approach are alos described. In the practical approach tools and integrated development enviorment (IDE) used for making of the application are described along with the application itself.

**Keywords:** cryptography, security, communication, application

Садржај

[1 Увод 1](#_Toc464013082)

[2 Криптографија 4](#_Toc464013083)

[2.1 Симетрични крипотсистеми 6](#_Toc464013084)

[2.1.1 Блоковске шифре 6](#_Toc464013085)

[2.2 Асиметрични криптосистеми 11](#_Toc464013086)

[2.3 TLS и SSL 12](#_Toc464013087)

[2.4 Керберос 14](#_Toc464013088)

[2.4.1 Логовање корисника 14](#_Toc464013089)

[2.4.2 Аутентификација клијента 14](#_Toc464013090)

[2.4.3 Аутентификација клијента на сервис 15](#_Toc464013091)

[2.4.4 Клијентов захтев за сервисом 15](#_Toc464013092)

[2.4.5 Недостаци и ограничења 16](#_Toc464013093)

[3 Апликација 17](#_Toc464013094)

[3.1 Опрема и алати 17](#_Toc464013095)

[3.1.1 Qt](#_Toc464013096) 17

[3.1.2 Crypto++](#_Toc464013097) 18

[3.1.3 Остало 19](#_Toc464013098)

[3.2 Дијаграми коришћења 19](#_Toc464013099)

[3.3 Кориснички интерфејс 22](#_Toc464013100)

[3.4 Код 23](#_Toc464013101)

[3.4.1 Заједнички делови 23](#_Toc464013102)

[3.4.2 Серверски део 30](#_Toc464013103)

[3.4.3 Клијентски део 32](#_Toc464013104)

[3.5 Недостаци 33](#_Toc464013105)

[4 Закључак 35](#_Toc464013106)

[5 Индекс појмова 36](#_Toc464013107)

[6 Литература 37](#_Toc464013108)

# Увод

*“Не постоји знање које није моћ.”*

* Ралф Валдо Емерсон

Информације су у савремено доба, поготово од популаризације интернета и паметних уређаја, постале врло одступне скоро свима. Довољно је да погледате у свој телефон и добијете инфромације о времену, а уз само неколико клика и о временској прогнози, реду вожње или најновијим вестима. Такође је подједнако лако да се контактира нека удаљена особа или да се дође до информациј где је и шта ради. Ово нарвно доноси велике предности од којих је најочигледнија, већ наведена, доступност скоро увек и скоро свуда, као и брзо преношење и ширење информација. Наравно све ово има и своје мане којих просечан корисник информационог система није баш увек свестан, чак што више технички необразовани људи врло често нису.

Најочигледнија мана јесте непровереност информација на које наилазимо. Наиме веома често људи верују ономе шта прочитају и виде, чак иако је те очигледно нетачно. Још већи проблем предстаља чињеница да се информације, па тако и лажне, невероватном брзином шире по мрежи. Путем друштвених мрежа људи су константо „повезани“ једни са другима и двољно је да једна особа окачи неку погрешну информацију да би је десетине ако не и стотне других види. Неки од њих ће сматрати ту информацију занимљивом и вредном дељења са осталима, и тако врло брзо број људи који ће знати, можда чак и користити, погрешну информацију експоненцијално расте. 1993. године Лиса Холст, да би доказала колико брзо и лако се дезинформације шире преко интернета и колико их људи лако прихватају, сатавила је лисут апсурдних „чињеница“, међи којима је можда најпознатија статисика да просечан човек за годину дана у просеку поједе осам паука у току сна. Те „чињенице“ су се наравно прошириле невероватном брзином, а у статистичку „чињеницу“ везану за паукове неки и дан данас верују иако је више пута објашњено да је у потпуности нетача. Међутим даљим истраживањем није могуће пронаћи више информација о Лиси Холст или магазину „PC Professional“, за који је она наводно писала чланке, што даље доводи до питања да ли је просто немогуће пронаћи те информација? Да ли је цела прича о Лиси Холст измишљена да би се проверило колико заиста људи верују у скоро све? Да ли то значи да је статисика о пауцима у ствари тачна, а цела прича о Лиси измишљена да људи не би поверовали у њу? У целој тој причи људи су се у главном концентрисали на то да ли је или није статистика тачна, да се скоро нико није ни запитао ко је и шта Лиса Холст. Могуће је да је једна нетачна или тачна информација побијена неком измишљеном причом. Увек постоји могућност да је неко случајно поделио неки нетачан податак или је то учино само да би се нашалио, али постоје и они који их намерно шире, било да би нашкодили некоме или једноставно зато што воле да праве неред.

При прикупљању и коришћењу информација са интернета увек је добро да се користи више извора. У колико се подациј о истом догађају, ствари или особи превише разликују на два извора онда неко ту греши или шири дезинформације. Треба користити и извор „од поверења“ чије би информације требало да су тачне, сигурне и проверене, али чак ни у том случају ништа не треба прихватати такво какво је, већ увек имати отворен ум и бити свестан могућности да није све онако како је нама приказано. Не мисли се наравно на то да човек треба да буде превише скептичан или параноичан и сматра да сви покушавају да га слажу или преваре, али исто тако не треба баш без и мало размишљања прихватити све шта нам се сервира.

Следећи велики проблем на који се наилази јесте количина личних информација које људи свесно или несвесно остављају на интернету. Свака информација је корисна, можда не за сваког али постоји сигурно неко ко може да је употреби или злоупотреби. У колико је прикупљено довољно наизглед небитних и бескорисних података и добро обрађено и изфилтрирано, могуће је доћи до веома конкретних и битних инфорамција. Популаризацијом друштвених мрежа људи су почели све више и више својих приватних информација и слика да деле са људима које чак ни не знају. Чак и људима са којима нису пријатељу преко друштвене мреже је омогућен приступ великој количини информација. Све чешће људи осећају потребу да са свима поделе то где су, с' ким и шта раде, што и није баш најпаметније. Поред прогонитељ који преко мреже могу да прате скоро сваки корак, позната је чињеница да велико компаније чије услуге користимо попут „Facebook“ и „Google“ без нашег знања прикупљају велике количине информација о нама. У Кини постоје веома стриктна правила везана за интернет. Конторлише се тачно чему корисници имају приступ и велика количина саобраћаја се филтрира. Они немају приступ неким од најпознатијих сајтова попут „Google“, „Facebook“, „Youtube“ и многи други, па уместо њих користе њихове алтернативе. Постоји апликација која се зове „WeChat“ која би требала да мења све апликације за друшвене мреже, али ради и много више од тога омогућујући људима да преко ње наручују храну, одћеу и слично, па чак и да плаћају. Са стране корисника то је одличан ствар јер коришћењем само једне апликације могу да обаве све шта им је потребно, нема потреба за констаним улажењм и излажењм из различитих апликација као ни за укључивањам више њих чиме се успорава рад уређаја, већ је све на једном месту. Велика мана тога јесте у томе што је све на једном месту, односно све информације су на једном месут и влада Кине, која се не устручава у томе, може да прикупи абнормалне количине информација о појединицима међу којима су ко је, одакле, шта ради, где излази, с' ким се виђа и још многе друге.

Постоји наравно добар разлог зашто се толике инфромацие прикупљају о просечном човеку који не би требало да је вредан толике пажње а то је безбедност. Свако од нас је потенцијални криминалац или терориста и у случају да се испостави да неко заиста то јесте могуће је коришћењм ових информација сазнати нешто више о том појединцу, што би помогло у његовом хватању, као и његових евентуалних сарадника, али и превенцији сличних ситуација у будућности. Чињеница је да има превише људи и да службе нису заинтересоване за сваког просечног човека понаособ, али све те информације могу да буду веома корисне и као такве их је неопходно чувати.

Претопстављамо, наравно, да компанија која поседује све те информације о нама нема намеру да их на било који начин злоупотреби. Али шта ако неко од запослених који им има приступ није морално чврст или има баш жељу да неком науди. Нарвано да се врше веома строге проверe када се запошљавају људи који у треба да имају приступ огромним базама података, али могућност увек постоји. Држимо се претопставке да нико из компаније ипак неће да злоупотреби наше информације, шта ако неко са стране некако успе да дође до њих. Колико год да је систем сигуран увек постоји могућност да постоји неки порпуст који неко ко је довољно вешт може да искористи. На крају крајева постоји могућност упада и крађе самих хард дискова на којима су сви ти подаци смештени. То су наравно све могућности на које ми никако не можемо да утичемо али могу веома да утичу на нас.

Даље постоји прблем комуникације, како преко мобилне мреже тако и преко интернета. Најочигледнији проблем јесте прислушкивање коминикације. Пресретањем порука или прислушкивањем разговора долази се до прилично личних информација, које у кралњем случају могу да буду и тајне. Највећи проблем јесте у томе што корисици могу да буду потпуно несвесни ове врсте напада јер што се њих тиче комуникације се одвија нормално. У случају пресретања порука могуће су њене измене и прослеђивање лажних порука. Могуће је приметити мало кашњење у коминикацији али је увек могуће да је разлог томе преоптерећеност мреже. Даље се поставља питање идентитета, односно да ли је онај с' ким комуницирамо онај којим се представља. Код разговора могуће је приметити разлике у гласу али шта је са порукама. Могуће је да је неко на неки начин успео да добије приступ профилу или телефону нашег саговорника, лажни профили на друштвеним мрежама су такође опште позната ствар. Како ми у том случају можемо да будемо сигурни да је наша комуникација безбедна?

Постоје наравно још многи други проблеми као што су вируси и други нежељени програми који некако успеју да уђу на наш рачунара и праве проблеме, који могу бити од неких безазлених шала, до озбиљнијих проблема као што је брисање битних информација или пројеката, као и крађа истих. Затим чување података на такозваним „облацима“, за које се показало да могу да буду компромитовани, мада да ли су наши рачунари нешто безбеднији. Листа се наставаља са лажним сајтовима и мејловима који у себи крију вирусе и иде до плаћања преко интернета, на шта се посебно обраћа пажња, јер ипак новац је у питању.

Чињеница је да је немогуће направити потпуно сигуран систем и да ће злонамерни људи увек наћи начин да оборе систем или дођу до неких информација до којих не би требало. Цела сигурност се заснива на одговарању на питања типа: „Шта ако ...?“ и у једном тренутку се дође до тога да на неко питање не може да се одговори или једноставно систем не би могао да функционише нормалоно или уопште када би могао да одговори на свако питање. Потпуно сигуран систем је идеал којим се тежи али је неопходно правити компромисе.

Идеја овог пројекта јесте да се направи систем који омогућује корисницима колико-толико безбедну комуникацију преко мреже. Постоји сервер на који се корисници логују, чиме се обезбеђује аутентификација, односно корисници који нису пријављени на систем не могу да комуницирају са пријављеним корисницима. Све поруке се шифрују, чиме се обезбеђује интегритет, тако да нападачи и ако пресретну поруке не могу да дођу до њиховог садржаја, такође било која измена се детектује немогућствром дешифровања или добијањем бесмисленог текста. За сваку комуникацију корисници добијају свој кључ за шифровање и дешифровање (биће у даљем тексту објашњено) који се не чува нигде, чак ни на серверу. Систем, наравно, има и своје мане које ће даље бити детаљније разматране.

# Криптографија

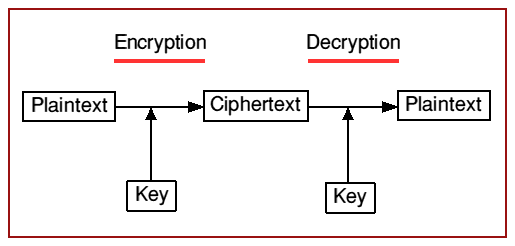
Криптографија је наука која се бави трансформацијом података тако да је њихово значење доступно само овлашћеним странам. Информације које се пренесе комуникационим каналом, који у општем случају није безбедан, могу услед прислушкивања да дођу до лица која не би требала да имају приступ тим информацијама. У колико су у питању финансијске, војне или државне тајне ово може да представља велики проблем, мада и злоупотреба приватних информација како би се нанела штета неком појединцу није занемарљив проблем. Употребом криптографије и одговарајућих криптографских система и алгоритама овај проблем је могуће превазићи.

Појавом писма појавила се и потреба да се сачува тајност садржаја неких писама, тада је и настала криптографија. Од самог почетка шифровање података користило се у војне сврхе. Међу првима је криптографију почео да користи Јулије Цезар, једна од најпознатијијх шифара приосте замене, Цезарова шифра, добила је име баш по њему. Он је све поруке које је слао својим војсковођама шифровао тако што је свако слово поруке замењивао, „померао“, словом које је 3 или више места после тог слова у абецеди. Тако шифровану поруку могли су да дешифрују и разумеју само они који су знали за колико места су слова померена, односно они којима је порука била намењена. Прву расправу о криптографији написао је Леоне Батиста Алберти 1467. Он је такође творац шифарског круга и неких других решења двоструког прикривања тексат која су прихватили усавршили немачки, енглески и француски бирои. Пола века након тога објављено дело Јоханеса Тритхемуса, прва књига о криптофгафији. У 16. веку значајан допринос дају милански доктор Гироламо Кардано, математичар Бартисто Порта и француски дипломата Блаисе де Вигенер.

Тек за време Другог светског рата појавила се машина која је шифровала поруке на до тада невиђен начин. Машину су направили немци и звала се Енигма. Међутим колико год да је она била револуционарана савезници су ипак успели да дешифрују поруке шифроване Енигмом. Оно шта је много више утицало на развој криптографије јесте употреба рачунара. Како су временом рачунари постајали све бржи и моћнији, број операција у секунди је скочио са неколико стотина на неколико милиона па чак и милијарди операција у секунди, 20. јуна. 2016. Sunway TaihuLight поставио је рекорд од 93 petaflops-a, што је 93\*1015 операција у секунди. Ово је потпуно променило начин шифровања и дешифровања порука, ове брзине омогућавају много брже разбијање шифрата што је довело да развијања нових сигурнијих и компликованијих алгоритама за шифровање.

Основни термини који се користе у криптографији су:

* Отворени текст (plaintext)- порука коју је потребно заштити.
* Шифровање (encryption)- операција којом се отворени текст мења тако да неовлашћеним странама буде потпуно неразумљив, односно да не могу извући никакве информације.
* Шифрат (ciphertext)- резултат шифровања, излазана вредост трансфорамације отвореног текста.
* Алгоритам шифровања- скуп правила који се користи за шифроавање отвореног текста.
* Кључ шифровања (key)- од њега зависе операције алгоритма шифровања, улазна вредност као и отворени текст.
* Дешифровање (decryption)- операција којом се из шифрата добија отворени текст
* Алгоритам дешифровања- скуп правила који се користи за дешифровање шифрата.



***Слика 2.1.:*** *Процес шифровања и дешифроавања*

Дужина кључа, односно број симбола којима је представљен, зависи од конкретне имплементације алгоритма и представља један од параметара сигурности система. Сигурност криптографског система почива искључиво на тајности кључа. Сам алгоритам је јаван, односно опште познат. Овакав принцип се користи јер у колико сигурност ситема зависи од алгоритам постоји могућност да неко украде или на неки други начин дође до алгоритам, што се у пракси показало као вероватно. У том случају потребно је мењани комплетан систем, док у колико сигурност система зависи искључиво од кључа у колико неко на неки начин дође до кључа довољно је само променити кључ и систем може да настави да се користи без икаквих већих промена. Такође јавни алгоритми су доступни великом броју криптографа који могу да открију и пријаве слабости.

Криптографија мора да обезбеди следеће:

* Интегритет односно ведостојност података који се шифрују, односи се на то да не дође до било какве неовлашћене измене или брисања податакака. У колико ипак некако дође до неовлашћене промене података неопходно је то детектовати, односно мора да постоји начин да се провери да ли су подаци мењани. Ову врсту напада могу да изврше и легални корисници који случајно или намерно покушавају да прекораче своја ограничења.
* Тајност, односи се на то да је садрђај података доступан само овалшћеним лицим односно онима који поседују кључ.
* Провера идентитета, оноси се на то да корисници пре приступа систему морају прво да се пријаве на систем.
* Немогућност избегавања одговорности, односи се на то корисник не може да порекне нешто за шта се зна да је баш он урадио. Врло битно код савремених система где се велики број новчаних трансакција обавља преко интернета.

Крипто-анализа је процес којим се из шифрата долази до информација о отвореном тексту без познавање кључа, односно само на основу шифрата. У ширем смислу обухвата и проучава слабости крипотграфских елемената, као што су, на пример, хеш функције или протоколи аутентификације. Технике криптоанализе називају се напади. Постоје два типа напада потпуна претрага кључева и скраћени напади. Потпуна претрага кључева представља испробавање сваке могуће вредности кључа из птостора кључева. Простор кључева је скуп свих могућих вредности кључева. Скраћени напади захтевају мање времена и обраде од потпуне претраге кључева. Неки од скраћених напада су:

* Напада на основу познавање шифтаза- у колико алогритам шифровања има слабости могуће је доћи до поруке само анализом шифрата.
* Напада на основу познатог дела отвореног текста- поруке које се шаљу најчешће имају унапред познате делове, као што су заглавља или слично. То може да се искористи за дешифроваењ остатка шифрата.
* Напад на основу изабраног отвореног текста- нападач себи шаље одређену поруку док корисник није за рачунаром(lunchbreak attack). Како нападач зна отворени текст и шифрат може да дође до кључа и дешифрује остале шифрате.
* Напда на основу адаптивно одабраног отвореног текста- нападач бира отворени текст, уради анализу, затим изабере нови текст на основу резулата. У колико добије очекиване резултате значи да је на добром путу.

Шифарски систем је сигуран у колико је најефикаснији напад потпуна претрага кључева. Простор кључева мора да буде довољно велики.

Постоје две врсте сигурности: рачунска и безусловна. Систем је рачунски сигуран у колико је цена разбијања шифрата већа од вредности шифроване информације или ако је време разбијања дуже од времена које информација треба да буде тајна. Систем је безусловно сигуран у колико не може да буде разбијен ни уз неограничене ресурсе и неограничено време.

Безусловно сигурна шифра је шифра код које се ни потпуном претрагом кључева не може доћи до отворене поруке.

## Симетрични крипотсистеми

Крајем Другог светског рата почела је употреба рачунара у криптоанализи што је даље довело до употребе рачунара и за шифровање и дешифровање. Код модерне криптографије отворени текст се преставља низом битова и алгоритми за шифровање и дешифровање раде над њима. И даље се користе комбинације транспозиције и супстуције. Две највеће предности модерене криптографије у односу на класичну, односно две највеће предности које доноси употреба рачунара, јесу могућнос реализације много сложенијих алгоритама за шифровање и дешифоровање и много већа брзина шифровања и дешифровања.

Симетрични криптосистеми користе исти кључ за шифровање и дешифровање, тако да обе стране које комуницирају морају пре почетка комуникације на неки сигуран и тајан начин да размене кључ. Постоје две врсте симетричних шифара а то су секвенцијалне и блоковске.

### Блоковске шифре

У општем случају отворени текст се дели на блокове одређене фиксне дужине и генеришу се блокови шифрата исте дужине. Над блоком отовреног текста се више пута понавља функција одређени број рунди. Улазни параметри сваке рунде су кључ и излаз из претходен рудне, и сама функција зависи од њих.

Блоковске шифре комбинују својаства дифузије, конфузије и комплетности. Дифузија значи да на основу познавања пара блока отвореног текста и шифрата не може да се одреди блок отвореног теста неког другог блок шифрата. Захтевано је да мале промене у блоку отвореног текста доведу до непредвидљивих промен у датом блоку шифрата. Својство конфузије подразумева су сви кључеви подједнако вероватни у случају напада потпуном претрагом кључева. Комплетност подразуева да је сваки бит шифрата функција сваког бита кључа.

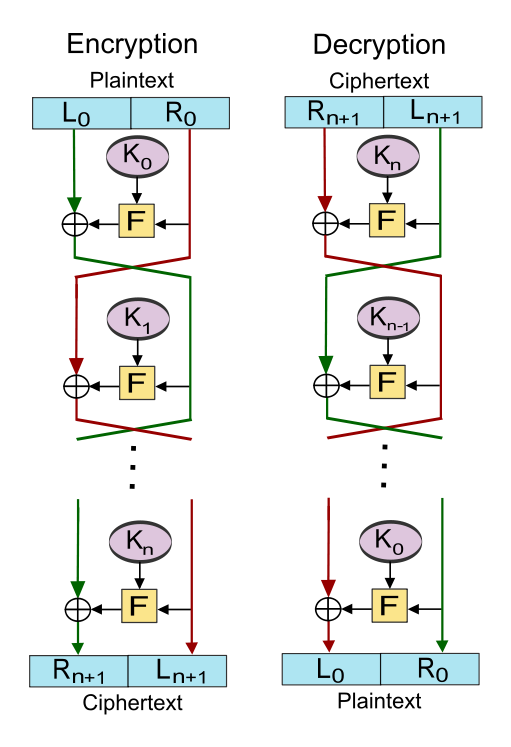
Блоковске шифре се најчешће реализују софтверски и треба да буду сигурне и ефикасне, што захтева висок ниво криптографског умећа.

#### Фејстел шифра

Названа је по Немачком криптографу Хорсту Фејстелу, који је радио за IBM и био пионир у развоју дизајна блоковских шифара. Фејстел шифра је дизајн блоковске шифре а не сама шифра. Његова иницијална истраживања су довел да развоја DES (Data Encryption Standard) алгоритма 1970.

Идеја и принцип рада Фејстел шифре су следећи:

* Отворени текст се дели на блокове одређене дужине (*P*).
* Сваки блок се дели на леви и десни део који су означени као *L* и *R*.
* У свакој рунди се рачуна , , где је *F* функција рунде а *Ki*подкључ који се добија комбиновањем битова кључа *K.*
* Шифрат је излаз последље рунде
* Дешифровање је инверзно
* *F* може да буде било која функциј али се збогбезбедности пажљиво бира.



***Слика 2.1.1.1.1:*** *Принцип рада Фејстел шифре*

#### DES

Развијен је „у мрачном добу компјутерсек ере“, односно седамдесетих, и базиран на IBM-овој Луцифер шифри.

Средином седеамдестих крипографија је била слабо позната ван војних и државних кругова, што је значило да корисници криптографије нису били упознати са стварном јачином зачтите криптографских производа, која је иначе била врло слаба. Али је било јасно да постоји потреба за заштитом дигиталних података, због све већег развоја и све веће употребе рачунара. У таквим условима је Национални биро за стандарде, или НБС, издао захтев за предлог нових шифара. Победник би постао стандард америчке владе и *de facto* индустријски стандард. Један од озбиљнијих кандидата на том конкурсу је била IBM-ова Луцифер шифра. Проблем је био у томе што у НБС-у није било стручњака за криптографију, па су морали да се окрену Националној сигурној агенцији, која се бави дизајнирањем и прављењем крипографских механизама за војску и владу, за помоћ.

Иако нису били одушевљени идејом да се мешају у ту причу ипак су се под притискуом умешали под условом да њихово учеће буде тајно. Када је информација о томе процурела појавиле се у се сумље да ће НСА оставити „задња врада“, backdoor, односно начин да разбију шифру. Међутим за 30 година крипоанализе нису откривена ни једна „задња врата“ или нека друга врста пропуста у дизајну алгоритма.

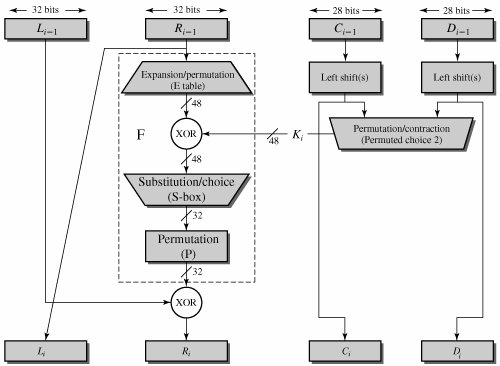
На крају Луцифер је постао DES уз неколико не баш суптилних промена, од којих су неке смањење кључа са 128 на 64 бита, од којих се 8 одбацује тако да је стварна дужина кључа 56 бита, и кутије замене или S-кутије.

Циљ је био да се уведе принцип конфузије, шифрат зависи од отвореног текста и кључа на сложен начин, дифузије, сваки бит шифрата је функција свих битова отвореног текста и свих битова кључа. Поред тога захтеван је и „лавински ефекат“, мале промене отвореног тексата доводе до великих промена шифрата, промена једног бита кључа или отвореног текста мења 50% битова шифрата.

DES је Фејстел шифра са следећим параметрима:

* Дужина блока отовреног текста је 64 бита.
* Дужина кључа је 56 бита.
* Има укупно 16 рунди пре генерисања једног блока шифрата.
* Дужина подкључа сваке рунде је 48 бита.

Функција *F* кодDES шифре се састоји од: експанзије (E-кутије), нелинеарних трансфорамације (S-кутије), пермутација (P-кутије), сабирању по модулу 2 и одабира бита кључа (К-кутија). Шифровање прати правила приказана у једначини:



***Слика 2.1.1.2.1.:****Шема DES алгоритма*

Експанзија (Е-кутија) пермутује и проширује свој улаз са 32 на 48 бита. Мења се редослед битова а неки се понављају ради постизања лавинског ефекта, тачно мапирање је одређено екпсанзионом табелом. Битови који се добију као излаз се XOR-ују са 48 бита подкључа те рунде.

S-кутија је реализована као нелинеарна стриктура што их чини тешким за анализу и најважнијим елеменатом на којем почива безбедност DES алгоритма. Има их осам и улаза сваке Ѕ-кутије се деоводи 6 битова а на излазу се добије 4, чиме се величина враћа на 32 бита. Ѕ-кутија се посматра као матрица од 4 реда и 16 врста, са једном 4-битном вредности која може да се нађе на било којој од 64 позиција. Свака Ѕ-кутија има своју табелу са константим вредностима. Када се на улаз доведе 6 бита први и последњи се издвајају и конвертују у децималну врендсот и они представљају индекс реда. Унутрашња четири бита се такође конвертују у децималну вреност и представљају индекс колоне. Пошто су добијени вредност колоне и реда узима се вредност из Ѕ-кутије, која се дели са 2 а остатак при дељењу се конверује у бинарну вредност. Када се добије излаз из свих Ѕ-кутија оне се пермутују на основу пермутационе матрице.

Р-кутија представља пермутацију излаза Ѕ-кутије по дефинисаном правилу.

К-кутије генеришу подкључеве , дужине 48 бита, за сваку рунду на основу кључа *К* дужине 56 бита. Генерисање се одвија у три фаза. У првој фази се прво одбацују крајње десни битови кључа. У следећем кораку кључ се представља пемутационом матрицом која се назива пермутациони избор 1. Битови се пермутују онако како је то задато у матрици иницијалне пермутације и као резулатат се добија скраћени и пермутовани кључ. У другој фази се генерише 16 подкључева, по један за сваку рунду. Кључ се дели на две половине леву и десну, дужине 28 бита, и свака се циклично помера одређени број пута. Тај нови кључ је улаз у последњу фазу. У последњној фази користи се изабрана пермутација 2, која садржи 48 позиција. За дате позиције се издвајају битови из кључа и као крајњи резулата се добија подкључ за рунду.

Како су линеране једначине лаке за решавање, сигурност DES алгоритма почива на једнином нелинеарном делу што су Ѕ-кутије, а у нешто мањој мери и од распореда кључа. За 30 година крипо анализе није откривена ни један пропуст или backdoor, како се у почетку сумљало због умешасти НСА. Најевећа слабост овог алгоритма јесте у томе што је дужина кључа веома мала. Иако су развијени напади који теоријски захтевају мање времена и посла од потпуне претраге кључева, сви до сада направљени програми за разбијање DES-a користе потпуну претрагу кључева. 1993. Мајкл Винер је показао да је могуће наравити хардвер којим је могуће разбити DES нападом типа отоворени текст и то за 35 сати са буџетом од 100 000 долара, 3,5 сати са буџетом од милион долара и 21 минута са буџетом од од десет милона долара. 2006. тим са Универзитета Бохуму и Кил у Немачкој развио је COPACABANA, комерцијално доступну машину за разбијање DES-a. 2008. развијена је COPACABANA RIVYERA која је смањила време разбијања шифре на мање од јендог дана. Једна оваква машина може да се произведе за 10 000 долара.

#### Троструки DES

Прво се дошло на идеју да се користи двоструки DES, односно да се порука два пута шифрује чиме се дуплира величина кључа а самим тим и порстор кључева. Међутим нападом типа „meet-in-the-middelе“ могуће је разбити двоуструки DES мање више слично DES-у. Напад је типа изабраног отвореног текста. У општем случају то изгледа овако:

Циљ је пронаћи оба кључа. За свако могуће К1 шифрује се Р и резултати се записују у табелу у облику парова. Затим се претпостави К2 и дешифру је С1 и на тај начин се добије неко Х. Ако Х постоји у табели пар кључева се проверава над новим отовореним текстом и шифратом, у колико се добије очекивани резултат онда је пронађен прави пар кључева. На овај начин се број неопходних покушаја да би се сигурно дошло до кључа смањује са 2112 на 256. Иако је величина потребне табеле огромна овај напад је изводљив.

Због несигурности двоструког DES-a појавио се троструки. И даље се користи два кључа, јер је дужина кључа од 112 бита довољна, али је прицнип рада другачији. Шифровање и дешифровањ код троструког DES-a су дефинисани на следећи начин:

Разлог због којег се шифрује једним кључем, затим дешифрује другим па поново шифрује првим је компатибилност, јер троструки DES код којег су оба кључа идентична, је у ствари обичан DES.

#### АЕЅ

Током деведестих дошло се до закључка да је DES превише несигуран јер је простор кључева од 56 бита једноставно превише мали. Неки посебно развијени програми су могли да открију кључ за неколико сати. Национални институт за стандарде и технологију издао је захтев за криптографски предлог алгоритма или AES (Advanced Encryption Standard). Било је много квалитетних предлога али је алгоритам под називом „Rijndael“ победио.

АЕЅ алгоритам има компликовану математичку структуру и није Фејстел шифра. Отпоран је на све познате нападе, веома је брз, могућ је паралелни дизајн као и имплементација на моногим процесорима и паметним картицама. Параметри АЕЅ-а су:

* Величина кључа је 128, 198 или 256 бита.
* Величина блока отовреног текста је 128, 198 или 256 бита (не зависи од величине кључа).
* Има 10 до 16 рунди, зависно од дужине кључа

Свака рунда се састоји од четири функције:

* ByteSub, нелинерани слој.
* ShiftRow, слој линеарног мешања.
* MixColumns, нелинеарни слој.
* AddRoundKey, додатни слој кључа.

АЕЅ улазне податке третира као матрице димезије 4х4, односно све операције шифровања и дешифровања се врше над матрицама. Улазни подаци се копирају у матрицу стања над којом се врше разне операције, а завршно стање те матрце, односно вредност матрице након извршених свих операција, се копира у излазни блок података, односно блок шифрата.

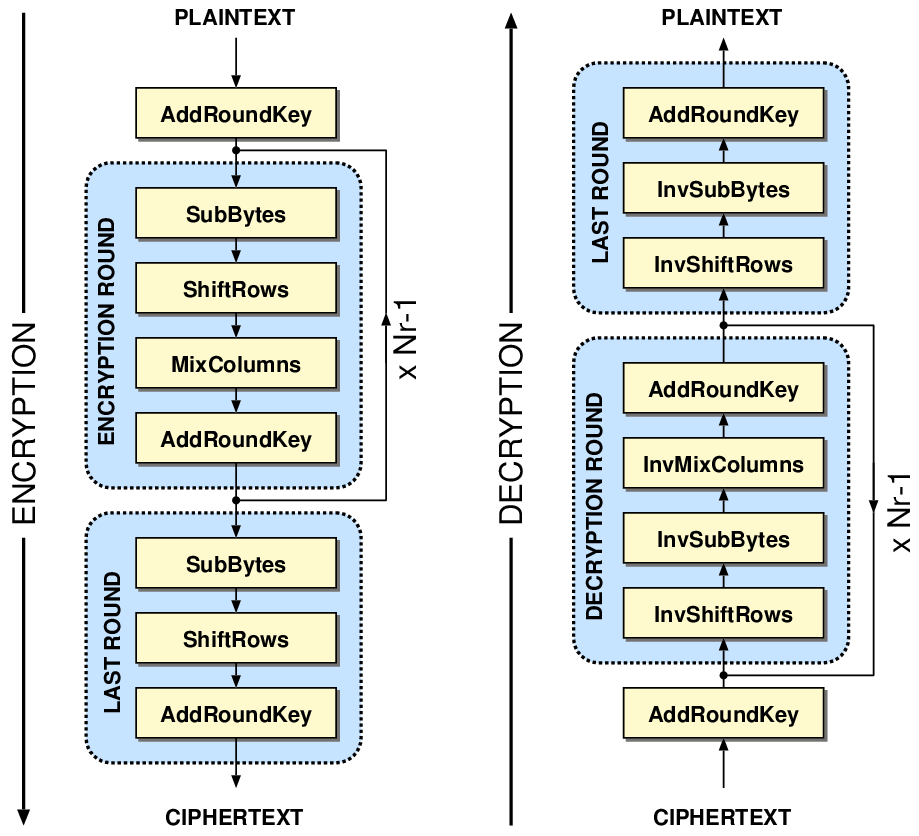
ByteSub је нелинерна функција и сличан је Ѕ-кутији код DES алгоритма. Може да се посматра као нелинеарна, инвертибилна композиција две математичке функције или као табела. Табела садржи 256 унапред дефинисаних вредности. Блок улазних података се записује у облику матрице а затим се нелинеарном функцијом добије друга матрица. На основу виша четири бита се адресира врста а на основу нижа четири колона табеле. Вредност бајта у коју ће се вредност из полазне матрице пресликати се добија из табеле.

ShiftRow је укратко циклично померање бајтова у последња три реда матрице. Сваки бајт друге колоне се циклично помера за једно место у лево, слично се бајтови друге и треће колоне циклично померају за два и три места у лево. Уопштено бајтови у реду n се циклично померају за n-1 места. Постоји мала разлика за блокове од 256 бита, први ред се и овде не помера али се други, трећи и четврти померају за 1,3 и 4 места. Међутим ово не важи за АЕЅ јер он не користи блокове те величине, али уопштено важи за Rijndael алгоритам. Битност овог корака је у томе да се избегне да колоне буду линеарно независне.

MixColumn је операција мешања колона матрице. Колоне се премештају по дефинисаним правилима и при томе се свака XOR-ује са одговарајућом матрицом. Цела операција је нелинеарна али инвертибилна. Мешање колона, MixColumn, и померање редова, ShiftRow, представљају основни извор дифузије.

AddRoundKey je XOR-овање добијене матрица са подкључем. Слично DES-у од целог кључа се бира подкључа за сваку рунду. Подкључ се пресликва у матрицу, која је исих димензија као и матрица која се добија као резулатат претходних операција, и која се затим XOR-ује са добијеном матрицом.

Све операције су инвертибилне и као резулата цео алгоритам је инвертибилан, због чега је могуће дешифровати исто као што се шифрује.



***Слика 2.1.1.4.1.:*** *Шема шфорвања и дешифровања АЕЅ алгоритмом*

#### Режим рада блоковских шифара

Режим рада блоковских шифара представља начин на који се блоковска шифра примењује на низ узастопних блокова. Наиме у колико се користи кључ*K* блоковкса шифра је исто што и кодна књига јер једнозначно преслликава блокове отвореног текста у блокове шифрата. Водећи се идејом кодне књиге дошло се до коришћења блоковских шифара у режиму „електронске кодне књиге“ или EBC. Шифровање и дешифровање изгледају овако:

Међутим овај режим се не користи јер има озбиљних сигурностних пропуста. Како се блокови отовреног текста једнозначно пресликавају у блкове шифрата, може се на основу идентичних блокова шифрата закључити да су њихови блокови отвореног текста такође идентични. У колико нападач зна део отвореног текста свако подударање са познатим блоком открива нови блок. Чак и ако се не зна део отвореног текста шифрат не би требао да одаје никакве информације о отвореном тексту. Ова слабост се превазилази коришћењем CBC или CTR режимом рада.

Код CBC, или Chiper Block Chaining, суседни блокови се уланчавају. Сваки блок се XOR-ује са шифратом претходног блока, за први се генерише нека случајна вредност која се назива иницијални вектор, *IV*. Као резултат блокови истог отвореног текста дају различите шифрате. Формула за шифровање је:

Формула за дешифровање:

Међутим због уланчавања у колико се шифрат преноси неким комуникационим каналом и у току преноса дође до неке грешке, неки бит промени своју вредност, отоврени текст може да постане непоправљив. У колико само један бит промени вредност онда два блока отвореног текста постају неисправна, јер сваки блок зависи само од два узастопна блока шифрата.

Код CTR, или режима бројача, се такђе користи иницијални вектор*IV* али на другачији начин. Иницијални векотр се шифрује и XOR-ује са блоком отвореног текста. Затим се иницијални векотр увећа за 1 и поступак шифровња и XOR-овања се понавља за следећи блок, и тако за све блокове. Формула за шифровање је:

Формула за дешифровање је:

## Асиметрични криптосистеми

Други тип криптосистема представљају асиметрични кирптосистеми односно ситеми са јавним и тајним кључем. Они неће бити детаљно разматрани као симетрични али ће бити објашњене неке основне особине и карактеристике због њихове употребе код SSL протокола.

За разлику од симетричниш криптосистема асиметрични користе два кључа, јавни и тајни. Јавни и тајни кључ се генеришу у пару, јер порука шифрована јавним кључем може да буде дешифрована једино његовим паром, односно тајним кључем који је парњак тог јавног кључа. Ови крипто системи су у главном засновани на математичким проблемима који немају ефикасно решење као што су једносмерне функције са замком. Кандидати за такве функције су дисктерни експоненет, његова инверзна функција је дикретни логаритам, производ целих бројева, његова инверзна функција је факторизација добијеног броја. Особине ових функција су те да се лако рачунају у једном смеру али им је тешко израчунати инверзну функцију осим ако се не зна тајна вредност.

Још једна битан разлика у односу на симетричне криптосистеме је та да није потребан тајни и сигуран начин за размену кључа. Користи се Дафи-Хелманов алгоритам за размену кључева. Принцип рада је следећи:

* Нека је *р* велики прост број и *q* тако да се за свако може наћи *n* тако да .
* Корисници, зваћемо их Алиса и Боб ради лакшег објашњавања, али и нападач знају *p* и *q.*
* Алиса бира тајну вредност *а*, која је неки велики случајан цео број.
* Боб бира тајну вредност *b,* која је такође велико сличајан цео број.
* Алиса јавно шаље Бобу.
* Боб јавно шаље Алиси.
* Обоје могу да израчунају вредност што у ствари представља кључ.
* Нападач зна и им може једино да израчуна што није кључ.

RSA алгоритам који се данас корити функционише на следећи начин. Ако су *С* шифрат *М* отворени текст, *N* и *е* представљају јавни кључ а *d* тајни, шифровање изгеда овако:

A дешифровање:

Вредности *N*, *е* и *d* треба да буду такве да за свако *М<N.*

Јавни и тајни кључ се генеришу на следећи начин:

* Изаберу се два велика проста броја *p* и *q.*
* Формира се
* Израчунава се (Ојлерова функција).
* Бира се *е* тако да је мање од и узајамно просто са .
* Нађе се *d* тако да је , односно
* *e*, *N* представљају јавну кључ.
* *d* представља тајни.

Дигитални потпис се добија када се отоврени текст шифрује тајним кључем. Тада се дигитални потпис дешифрује јавним кључем чиме се потврђује идентитет пошиљаоца. У главном се за дигитални потпис и шифровање користе различити парови јавног и тајног кључа.

Проблем који се јавља јесте утврђивање идентитета, односно власника јавног кључа. У ову сврху се користе стране од поверења које треба да гарантује за све регистроване кориснике и да на захтев потврди њихов идентитет. Сертификациона страна је трећа страна од поверења. Корисници генеришу сертификат који сертификациона страна потписује.

С' обзиром на то да су системи са јавним кључем два до три пута спорији од симетричних систем, дужина кључа код RSA је 1024 бита а код AES-а 256, системи са јавним кључем се користе за размену симетричног кључа па се онда прелази на симетричну криптографију.

## TLS и SSL

TLS (Transport Layer Security) и његов претходник SSL (Secure Socket Layer) су криптографски протоколи који обезбеђују сигурну комуникацију преко мреже. Постоји више верзија оба, иако је TLS настао од SSL-a и у великоје мери га заменио и даље се често користи термин SSL када се мисли на TLS, и користе се у велоком броју апликација за претраживање интернета, електронску пошту, слање факса преко интернета, ћаскање преко интерента и VOIP (Voice-Over-IP), већи сајтови користе TLS за сигурну комуникацију између сервра и претраживача.

Развије је од стране Netscape Communicaions деведесетих, за сигуран пренос података између Netscape Navigator претраживача и сервера, како би обезбедили заштићеност осетљивих података као што су бројеви кедитних картица. Прва верзија никада није јавно објављена а верзија 2.0 која је објављена фебруара 1995. је имала велики број сигурносних пропуста. Верзија 3.0 је захтевала потпуни редизајн и објаљена је 1996. Иако никада није био формално стандардизован постао је *de facto* стандард за сигурност у комуникацији преко интернета. Након што га је IETF (Internet Engineering Task Force) стандардизовала верзија 3.1 је објављена као TLS 1.0, име је промењено из правних разлога, и увдена су побољшања која су побољшала слабости. Многи од напада на SSL су засновани на самој имплементацији, али POODLE слабаост је позната мана саме 3.0 верзије која је омогућавала нападачу да дешифрује осетљиве податке као што су колачићи (cookies). TLS није слаб на ову врсту напада али има још предности које га чине сигурнијим од SSL-а. Најновија верзија је 1.2, а документација за 1.3 је објављена.

SSL се састоји од два под-протокола: протокол за „руковање“ и протокол записа који омогућују успостављање шифроване SSL конекције. У току „процеса иницијалног руковања“ клијент шаље листу алгоритама које може да користи и насумичну вредност *Rа* сервер, да би потврдио свој идентитет, клијенту шаље свој дигитални сертификат, насумичну вредност *Rb*и одабрани алгоритам. Сертификат је у Х.509 формату који дефинише Public-Key Cryptography Standards (PKCS). Аутентификација користи шифру са јавним кључем да валидира сертификат и потврди да је сервер тај којим се представља. Када је сервер аутентификован постављају се подешавања за шифровање и кључ који ће се користити за шифровање свих података које буду размењивали, чиме се обезбеђује поверљивост података и интегритет.

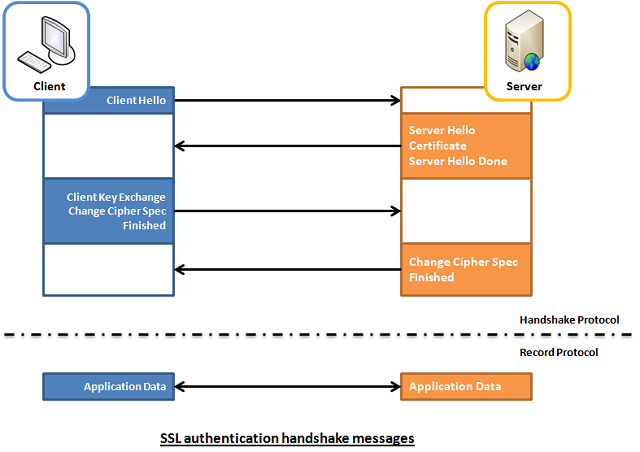
Када клијент код сертификационог тела потврди примљени сертификат он једино зна да има прави сертификат и јавни кључ, али још увек није сигуран да комуницира са правим сервером. Генерише тајну вредност S коју шифрује јавним кључем који је добио од сервера и кључ *К*:

*h* предстаља хеш функцију. На све претходно размењен поруке, које се шљау да би сервер био сигуран да су све порује успешно размењене, клијент дописује договорену константу *CLNT* и *К,* и на крају рачуна хеш вредност свега тога.

Добијена вредснот се шифрује договореним алгоритмом и кључем *К.* Клијент затим шаље шифровану вредност *Ѕ* и

Када сервер прими ове поруке може да израчуна, дешифрује, *Ѕ* и пошто зна *Rа*и *Rb*може да израчуна *К* и дешифрује . Затим формира поруку тако што на све већ размењене поруке допише константу *SRVR* и кључ*К* и клијенту шаљ вредност:

И у овом случају све послате поруке се шаљу ради сигурности да су све успешно размењене. Након овога корисник је сигуран да је комуницира са правим сервером и да су све поруке успешно размењене. Вредност *К* се користи за даљу комуникацију и из ње се рачуна барем шест кључева: два за шифровање, два за интегритет и две иницијалне вредности по једна за пријем и предају.



***Слика 2.3.1.:*** *Поступак аутентификације код SSL протокола*

Читав овај процес је невидљив за корисника и не захтева никакво ангажовање са његове стране. У случају претраживача, у колико је успостављена сигурна комуникација, URI се мења из HTTP у HTTPS. Уколико ипак дође до напада или сертификат није могуће аутентификовати корисник добија упозорење али и даље има могућност да настави комуникацију.

Конекција осигурана SSL протоколом има следећа својства:

* Конекција је приватна, сигурна, јер се симетрична криптографија користи за шифровање података који се шаљу. Кључ који се користи је једниствен за сваку конекцију и заснован на дељеној тајни која се договоара на почетку сесије (). Алгоритам који ће се користит за шифровање је такође договорен пре почетка слања података. Договарање дељене тајне је такође сигурно, нападача не може да дође до ње чак и ако се нађе у сред конекције и комуникација за време преговарања не може да се промени а да то не буде детектовано.
* Обе стране у комуникацији могу да буду аутентификоване коришћењем криптографског система са јавним кључем. Ово може да буде опционо, али се у главном захтева да се барем једна страна, сервер, аутентификује.
* Конекција обезбеђује интегритет јер свака послата порука укључује и кода за аутетнтификацију поруке како би се спречио недетектовани губитак или измене података током преноса.

## Керберос

Керберос је протокол за аутентификацију заснован на „тикетима“ који омогућава корисницим који комуницирају преко несигурне мреже да докажу свој идентитет једни другима на сигуран начин. Назив је добио по Керберосу, троглавом псу чувару улаза у подземни свет из Грче митологије. Заснован је на клијент-сервер моделу и обезбеђује аутентификацију и корисника али и сервера. Комуникација обезбеђена овим протоколом је заштићена од прислушкивања и напада поновног слања.Идеја и концепт апликације израђене у практичном делу рада су осмишљени да донекле симулирају, или опонашају, овај протокол.

Развијен је на MIT-у да заштити мрежне услуге обезбеђене пројектом Атина. Постоји више верзија с' тим да су верзије 1 до 3 познате само унутар MIT-a. Осамдестеих су Стив Милер и Клифорд Нуман, главни дизајнери верзије 4, објавили ту верзију иако је првенствено била намењена за пројеката Атина. Верзија 5, дизајнирана од стране Јона Кола и Клифорда Нумана, се појаваила 1993. са намером да превазиђе ограничења и недостатке верзије 4. 2005. Internet Engineering Task Force је надоградила спецификацију кербероса. Надоградња укључује:

* Спецификацију за енкрипцију и checksum.
* АЕЅ енкрипцију за Керберос 5.
* Нову верзију Керберос В5 спецификације „Керберос Услуга за Аутентификацију преко Мреже (В5)“.
* Нову верзију апликацијоног интерфејса за апликације генеричких сигурносних услуга, „The Kerberos Version 5 Generic Security Service Application Program Interface (GSS-API) Mechanism: Version 2“.

MIT је учинио имплементације Кербероса бесплатним, уз одређена ограничења. 2007. је формиран Керберос конзорцијум ради даљег развоја. Windows 2000 и новије верзије користе Керберос као подразумевани метод аутентификације. Многи UNIX и UNIX-олики оперативни системи, укључујући FreeBSD, Mac OS X, Red Hat Enterprise Linux, Solarais, AIX, OpenVMS и други, садрже софтвер за Керберос аутентификацију корисника и услуга. Embedded импементације Керберос В5 протокола су такође доступне.

Клијент се аутентификује код сервера за аутентификацију који прслеђује корисничко име центру за дистрибуцију кључева (Key Distribution Center- KDC). КDC генерипе ТGT (Ticket-Granting Ticket), који се шифрује корисниковом лозинком и шаље кориснику. TGT има рок трајања, односно истиче у одређеном тренутку, мада може да буде освежен од стране корисничког менаџера сесије, што је за корисника невидљиво. Када корисник жели да комуницира са другим корисником он свој TGT шаље сервису за давање тикета (Ticket Granting Service- TGS), који је најчешће на истом хосту као и KDC. Након што се TGT верификује коринику се омогућава коришћење сервиса. TGS генерише тикет и сесијски кључ и шаље кориснику. Корисник затим шаље тикет серверу сервиса (ЅЅ) заједно са својим захтевом. У даљем тексту ће бити детаљније описани сви кораци.

### Логовање корисника

Корисник уноси корисничко име и лозинку. Други механизми дозвољавају коришћење јавног кључа уместо лозинке. Клијент трансформише лозинку у симетрични кључ. За ово се најчешће користи хеш функција, мада је могуће и генерисање кључа на неки други начин.

### Аутентификација клијента

Клијентска апликација шаље кориснички идентификатор АЅ-у захтевајући сервис за корисника. АЅ генрише тајни кључ рачунајући хеш корисникове лозинке која се налази у некој бази. АЅ проверава да ли се клијент уопште налзи у бази. Ако се налази шаље клијенту поруке:

* Кључ за сесију са ТGS-ом шифрован корисниковим тајним кључем.
* TGT, који се састоји од: корисникове идентификације, адресе клијента (клијент значи апликација која комуницира са сервером), време важења тикета и кључа за сесију са TGS-ом, све то шифровано тајним кључем TGS-a.

Када клијент прими ове поруке покушава да дешифрује прву поруку тајним кључем који се генерише коришћењем лозинке коју је корисник унео. У колико је унета погрешна лозинка клијент неће моћи да дешифрује ову поруку. Уз одговарајућу лозинку и тајни кључ клијент дешифрује поруку добија кључ за сесију са TGS-ом, који се користи за даљу комуникацију са TGS-ом, али не може да дешифрује другу поруку, односно TGT. Сада клијент има довољно информација да се се аутентификује код TGS-a.

### Аутентификација клијента на сервис

Када захтева сервис, клијент шаље селедеће поруке TGS-у:

* Поруку која се састоји од TGT-а и од идентификације захтеваног сервиса.
* Аутентифакотор који се састоји од клијентске идентификације и timestamp-а, шифрован сесијским кључем за TGS-a.

Када прими ове поруке TGS из прве поруке узима TGT. Коришћењем свог тајног кључа дешифрује TGT чиме добија клијентов кључ за сесију са TGS-ом. Тим кључем дешифрује другу поруку и шаље клијенту следеће две поруке:

* Client-to-sever тикет, који се састоји од идентификације клијента, клијентове адресе и трајања кључа за сесију са сервером, шифрован коришћењем тајног кључа сервиса.
* Кључ за сесију са сервером шифрован кључем за сесију са ТGS-ом.

### Клијентов захтев за сервисом

По пријему порука од TGS-a, клијент има довољно информација да се аутентификује код ЅЅ-а. Клијент се повезује са ЅЅ-ом и шаље следеће поруке:

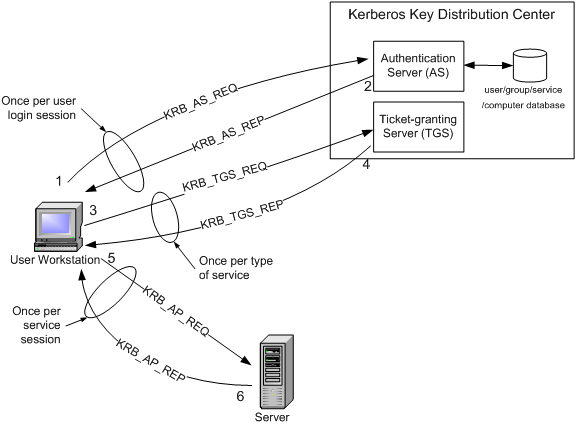
* Client-to-server тикет, који је добио од TGS-a.
* Нови аутентификатор који се састоји од идентификације клијента и timestamp-a шифрован кључем за сесију са сервером.

ЅЅ дешифрује тикет коришћењем свог тајног кључа да би добио клијентов кључ за сесију са сервером. Коришћењем кључа за сесију, ЅЅ дешифрује аутентификатор и шаље клијенту следећу поруку као потврду идентитета и прихавтања да кориснику пружи сервис:

* timestamp из корисниковог аутентификатора, шифрован корисниковим кључем за сесију са сервером.

Клијент дешифрује потвду коришћењем кључа за сесију са сервером и проверава да ли је timestamp добар. Ако јесте онда клијентможе да има поверења у сервер и може да шаље захтеве за сервисима.

Сервер извршава сервисе захтеване од стране клијента.



***Слика 2.4.1.:*** *Цео процес од пријаве до захтева за сервисом*

### Недостаци и ограничења

Највећи недостатак је потреба да централни сервер буде константо активан, у супротном нови корисници не могу да се пријаве. Овај проблем је могуће превазићи употребом више Керберос сервера и механизама за утентификацију у случају квара.

Следећи велики проблем је потреба да сатови буду синхронизовани, унутар неких граница наравно. Тикети имају време трајања и у колико кориснички и сат на серверу нису синхронизовани корисник неће моћи да се аутентификује. По стандардној конфигурацији MIT-а сатови не смеју да се разликују за више од пет минута. Могуће је да сервер шифровано пошаље своје време клијенту, што омогућује да се израчна разлика.

Протокол за администрацију није стандардизован и варира од имплементације до имплементације.

У случају коришћења симетриче криптографије, пошто може да ради и са асиметричном, пошто сву аутентификацију контролише КDC, у колико је он компромитован нападач може да имитира било ког корисника.

Сваки сервис захтева своје име и своје кључеве чиме се компликује виртуелно хостовање.

Сви кориснички профили, клијенти и сервиси морају да буду у истом домену или домену у који се има поверење. Керберос не може да се користи у систему где корисник жели да се повеже на сервис који није познат или није од поверења.

Неки од старијих имплементација Кербероса нису унапређене новим стандардом, што занчи да и даље користе DES алгоритам који није сигуран.

# Апликација

Сама концепт апликације је веома слична принципу рада Керберос протокола, с' тим да су све функционалности обједињене на једном месту. Идеја је да се направи сервер на који би има функционалности сервера за аутентификацију и сервера сервиса, с' да је једини сервис који је доступан сервис генерисања кључа за комуникацију. Корисници се пријављују на сервер и након успешне аутентификације добијају кључ за комуникацију са сервером. Поред тога имају и листу свих корисника који су online и са којим могу да започну комуникацију. При започињања комуникације серверу се шаље захтев за комуникацију са другим корисником, на шта сервер одговара, у колико је наравно све прошлко како треба, кључем за њихову комуникацију. Након тога корисници могу безбено да комуницирају без даље потребе за сервером.

Апликација је писана у С++ програмском језуку коришћењем Qt 5.7 Framework-а, Qt Creator развојног окружења и Crypto++ библиотеке која садржи све потребне класе за шифровање и дешифровање. Сав развој и тестирање су вршени на Ubuntu 16.04 64bit оперативном систему.

Детаљнији опис коришћених алата и функционисање апликације се налазе у даљем тексту.

## Опрема и алати

Апликација је направљена и тестирана на лаптопу Toshiba Satellite L50 B 25M чије је основна спецификација дата у следећој табели.

|  |  |
| --- | --- |
| Процесор: | Intel Core i7-5500U 2.4GHz (Turbo do 3.0GHz, 4MB cache, 2 језгра, 4 thread-a) |
| Меморија: | 8GB 1600MHz DDR3L (2 слота, максимално до 16GB) |
| SSD: | Sandisk Aphrodite II-Z400s (256GB, 33/62k IOPS, 2,5” Cased) |
| Графичка картица: | Intel HD Graphics 5500 + AMD Radeon R7 M260 2GB DD33 |
| Екран: | 15.6 inča LED HD 1366x768 пиксела |
| Мрежа: | WiFi 802.11b/g/n/ac, dual-band, Bluetooth 4.0, Gigabitni LAN 10/100/1000Mbps |
| Батерија: | Li-Ion 4-cell (do 6h15min) |
| Оперативни систем: | Ubuntu 16.04 64bit |

***Табела 3.1.1.:*** *Основна спецификација коришћеног лаптопа*

### Qt

Qt је cross-platform framework за апликације који се корисит за развој апликација које могу да се покрећу на широком спектру софтверских и хардверских платформу без икаквих, или уз врло мало, промена у коду. Развија се од стране The Qt Company и Qt Project, и доступан је у комрецијалној и open source форми.

У главом се користи за развој апликациј са графичким интерфејсом (Graphical User Interface-GUI), али и поргорами без графичког интерфејса могу да се развијају, као што су конзолне апликације и слично. Када се развијају апликације са интерфејсом Qt се класификује као widget toolkit.

Qt користи стандардни С++ са неким додацијма као што су сигнали и слотови. Сигнали и слотови у великој мери олакшавају обраду догађаја (event), што помаже у развоју како GUI апликација тако и серверских апликација, од којих сваке имају свој специфичан скуп догађаја који је потребно на прави начин обрадити. Подржани су многи комплајлери као што су GCC C++ или компајлер Visual Studio развојног окружења. Такође је обезбеђен Qt Quick који укључује декларатвни скриптни језик QML који омогућује коришћење JavaScript логике. Уз Qt Quick могућ је брз развој мобилних апликација. Поред С++-а могуће је коришћење и других програмских језика. Функционише на свим већим desktop платформама и неким од мобилних платфоми. Поред GUI могућности, омогућује и повезивање на SQL базу, парсирање XML-а и JSON-а, рад са нитима и комуникацију преко мреже.

Три кључне концепта Qt-a су:

* Потпуна апстракција GUI-a. Када је први пут објављен Qt је користио свој engine за сликање и своје контроле, чиме је емулиран изглед различитих платформи кад су сликани widget-и. Овим је олакшана преносивост јер библиотке ниси зависиле од платформе, али је понекад долазило да мало лошијег приказа. Новије верзије користе домаћи API разчичитих платформи.
* Сиглани и слотови. Ово је механизам који омогућује комуникацију међу објектима. Коришћењем овога код је много једноставнији. Концепт је да GUI компоненте могу да шаљу сигнале са информацијама о догађају који могу да приме друге компоненте коришћењем посебне методе која се назива слот.
* Metaobject компајлер. Ово је алата који се покреће на изворном коду Qt програма. Интерпретира макрое из C++ кода и користи их да генерише додатни С++ код са метаподацима о каласма које се користе у програму. Ове метаподатке Qt користи да обезбеди оно шта иначе није доступно у С++-у: сигнале, слотове, интерспекцију и асинхроно позивање функција.

Подржава многе платформе од којих су неке: X11, Wayland, Embedded Linux, Android, OS X, iOS, Windows, Windows CE, Windwos RT, Integrity, QNX, VxWorks.

Qt долази са својим алатима који олакшавају развој апликација развијених у Qt-у. Qt Creator је cross-platform IDE за С++ и QML. Qt Designer је интегрисан у IDE, иако је могуће да се одвојено покрене. Поред овога обезбеђен је и qmake, алат за генерисање build скрипти, који аутоматски генерише Make фајлове за развој пројекта на различитим платформама. Без овог алата би морао да се напише по Make фајл за сваку платформу. Остали алати који су доступни су Qt Designer, Qt Assisatant, Qt Linguist, uic (user interface compiler) и moc (Meta-Object Compiler).

Овај framework је изабран за израду пројкета, пре свега зато што својим већ готовим библиотекама у великој мери олакшава програмирање у С++ програмском језику и својим концептима олакшава креирање GUI-a и обраду догађаја. Оно шта је такође веома битна особина јесте то што је cross platform, односно могућност покретања истог кода на различитм платформама.

### Crypto++

Crypto++ је библиотека написан од стране Веи Даи и укључује алгоритме за шифровиање, хеш функције, криптосистеме са јавним кључем, шеме за договарање кључа и друго. Сама билиотека је прилично обимна и биће укратко описани само они делови који су коришћени за израду апликације. Детљнији опис целе бибилотеке, упуство за коришћење као и линкови за преузимање могу се пронаћи на сајту: <https://www.cryptopp.com/>.

Заглавље cryptlib.h је вероватно најбитиније заглавље из бибилотеке. Оно укључује: Exception, BufferedTransformation, BlockTransformation, StreamCipher, Random-AccessStreamCipher, RandomNumberGenerator, HashTransformation, MessageAuthenticationCode. Скоро све остале каласе наслеђују неку од класа из овог заглавља. Најбољи начин да се научи како функционише нека од изведених класа јесте да се прво види како функционише класа из овог заглавља из које је изведена та класа.

Заглавље filters.h дефинише класе за унос и приказ података, класе су изведене из BufferedTransformation, и класе које су овде дефинисане су Filter, Sink, Source и класе изведене од њих. Ово је друга најбитније заглавље у библиотеци чије класе такође многе друге класе наслеђују. Ова класа је врло битна за елегантан рад са подацима.

Заглавље hex.h обезбеђје класе HexEncoder и HexDecoder, обе наслеђују класу Filter.

Загвлавље aes.h дефинише AESEncryption и AESDecryption који су у ствари само други називи за класе RijndaelEncryption и RijndaelDecryption. Из назива је очигледно да се користи Rijndael алгоритам за шифровање и дешифровање.

Заглавље osrng.h омогућује генерисање квалитетних насумичних врености које су специфициране платформом. Обезбеђује AutoSeedRandomPool, класу изведену из RandomPool која сама себи поставља почетну вредност користећи BlockingRng или NonblockingRng, у зависности од тога шта је доступно.

Заглавље modes.h обезбеђује интерфејсе за коришћење блоковских алгоритама у CFB, OFB, CBC или неком другом моду.

Ова библиотека обезбеђује све потребне класе и функционалности неопходне за израду апликације.

### Остало

Поред већ наведених алата коришћени су и OpenSSL и AQemu.

OpenSSL је библиотека која се корити у апликацијама које захтевају сигурну комуникацију, безбедну од прислушкивања или имају потребу да идентификују другу страну. Пронашла је широку примену у web серверима. Садржи open-source имплементације SSL и TLS протокола. Библотека је написана у С програмском језику и имплементира основне криптографске функције. Постоје „обмотачи“ (wrappers) који омогућују коришћење OpenSSL библиотеке у другим програмским језицима. Разне верзије су доступне за различите Unix-онлике оперативне системе као: Solaris, Linux, Mac OS X и разне open-source BSD оперативне системе, OpenVMS и MicrosoftWindows.

Протоколи дефинисане у овој библиотеци су коришћени код SSL сокета ради безбедне комуникације. Такође ова библиотека је коришћена за генерисање сертификата и RSA кључа за сервер.

Aqemu је графички интерфејс за QEMU и KVM емилаторе. QEMU је генерични open-source емулатор и алата за виртуелизацију. У главном се користи у комбинацији са неким убрзивачем као што је KVM или Xen. Коришћење QUEM-а самостално је врло неефикасно и споро.

Овај алата је коришћен за покретање виртуелене машине и тестирање рада апликације преко wireless мреже.

## Дијаграми коришћења

|  |  |
| --- | --- |
| **Корисник** | **Сервер** |
| 1. Корисник покреће апликацију која се повезује на сервер преко SSL сокета што обезбеђује сигурну комуникацију. |  |
| 1. Корисник серверу шаље своје идентификационе податке, односно корисничко име и лозинку. | 1. Сервер прима податке. Проверава да ли корисник постоји у бази корисника. У колико не постоји или је већ пријављен на систем шаље се порука о грешци. |
|  | 1. У колико су подаци исправни генерише се кључ за комуникацијом са сервером који се шаље кориснику заједно са потврдом о успешном пријављивању. |
| 1. Корисник прима одговор од сервера. У колико је потвдан поставља се кључ који се користи за комуникацију са сервером. У колико није приказује се порука о грешци. |  |

***Табела 3.2.1.:*** *Приајва корисника на систем*

Untitled Diagram.png

***Дијаграм 3.2.1.:*** *Пријава корисника на систем*

У табели и на дијаграму су представљени поступак пријављивања корисника на сервер и добијање кључа за комуникацију са сервером. Пре свега се преко SSL сокета остварује сигруна конекција. Након што се корисник успешно повеже са сервером шаље своје податке, односно корисничко име и лозинку. Сервер проверава примљене податке и у колико корисник са тим подацима не постоји у бази корисника или је већ пријављен на систем кориснику се враћа порука о грешци. У колико су подаци исправни и корисник није већ пријављен на систем генерише се кључ за корисника и шаље заједно са поруком којом се потврђује успешно повезивање на север. Након тога се све поруке које корисник и сервер размењују додатно шифрују генерисаним кључем као додатни ниво заштите, иако би SSL сокет требало да обезбеди сигурну комуникацију.

|  |  |  |
| --- | --- | --- |
| **Корисник2** | **Корисник1** | **Сервер** |
|  | 1. Корисник шаље серверу захтев за комуникацију са другим корисником. | 1. Сервер прима захтев и проверава податке. У колико подаци нису исправни шаље се порука о грешци. |
|  |  | 1. Генерише се кључ за комуникацију корисника и шаље кориснику који је захтевао комуникацију. |
|  | 1. Корисник прима поруку од сервера. У колико је порука о грешци она се исписује. |  |
| 1. Прима и поставља кључ за комуникацију са корисником. | 1. У колико је примљен кључ он се проселеђује другом кориснику и поставља за комуникацију са тим корисником. |  |

***Табела 3.2.2.:*** *Захтев за комуникацијом са другим корисником*

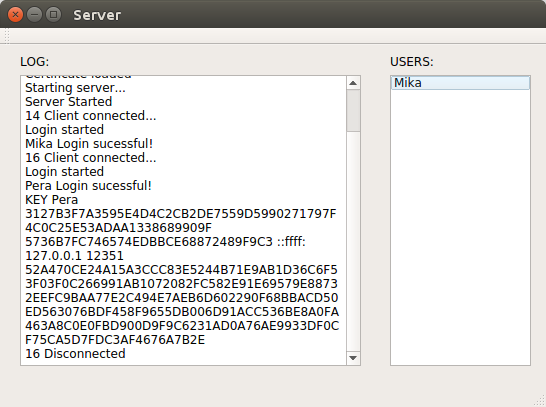
Untitled Diagram (1).png

***Дијаграм 3.2.2.:*** *Захтев за комуникацијом са другим корисником*

У другој табели и на другом дијаграму су приказани поступак слаља захтева за комуникацијом са другим корисником. Корисник прво шаље захтев у којем се наводи са ким жели да комуницира. Услов за успостављање комуникације јесте да су оба корисника у тренутку прихватања захтева пријављена на систем, како може да буде много корисника и много захтева ову проверу је неопходно обавити јер захтев неће увек бити обрађен чим се прими. У колико су оба корисника пријављена на систем шаље се потврдна порука. Ова порука се састоји од корисничког имена корисника са којим је захтевана комуникација, кључа, адресе и порта корисника, а затим други део поруке који је намењен другом кориснику се састоји од корисничког имена корисника који је захтевао комуникацију и кључа. Други део поруке је шифрован кључем кругог корисника тако да само он може да га дешифрује а цела порука је шифрована кључем корисника који је захтевао комуникацију, тако да само он може да је прочита. Када корисник прими поруко он је дешифрује, поставља кључ за комуникацију са другим корисником и, пошто зна адресу и порт, прослеђује му други део поруке. Други корисник прима поруку дешифрује је и из ње узим податке о томе са ким комуницира и кључу који се користи за ту комуникацију. Након што су оба корисника све поставила могу да започну сигрурно да комуницирају. Корисници за међусобну комуникацију не користе SSL сокете, већ обичне, због чега је битно да све поруке буду шифроване.

Овде је приказано уопштено како изгледа процес пријављивања на сервер и започињања комуникације са другим корисником. Даље ће детаљније бити описан цео процес као и конкретан изглед порука.

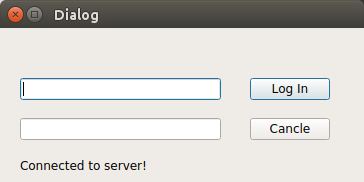
## Кориснички интерфејс



***Слика 3.3.1.:*** *Сервер*

На слици 3.3.1. је приказан изглед графичког интерфејса сервера. Као што се види нема никавих команди и служи сикључиво за приказивање информација преко два текстуална поља LOG и USERS. LOG приказује статусне информације о покретању сервера, пријављивању и одјављивању корисника, порукама које су генерисане и евентуалним грешкама.

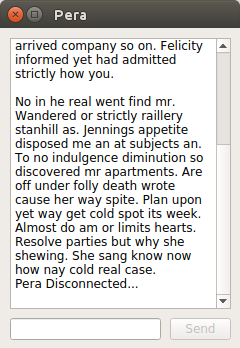
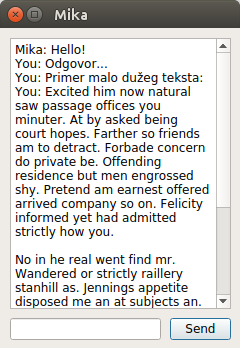
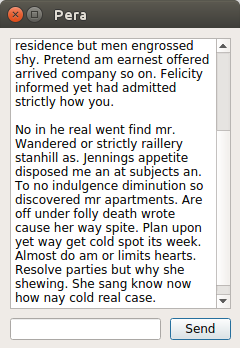
Даље ће бити приказан изглед клијентског интерфејса. Прво ће бити приказан прозор за пријаву, након тога изглед прозора из којег се започиње комуникација и на крају сами прозори за слаље порука.



***Слика 3.3.2.:*** *Клијентски прозор за пријаву*



***Слике 3.3.3. и 3.3.4.:*** *Кориснички прозори*



***Слике 3.3.5.-3.3.7:*** *Изглед прозора за слање порука*

На сликама је приказан кориснички интерфејс. Прво је приказан прозор за пријаву на систем. У дну порозора се налази порука о статусу повезаности са сервером. У колико сервер из неког разлога није покренут или апликација није могла да се повеже, неопходно је искључити апликацију у покушати поново касније. У два текстуална поља је потребно унети корисничко име и лозинку у притиснути дугме Log in.

У колико су унети исптавни подаци добија се кориснички прозор, приказан на сликама 3.3.3. и 3.3.4., у супротном се на прозору за пријаву исписује порука о грешци. Кориснички прозор је врло једноставан, састоји се само од листе корисника који су тренутно пријављени и дугмета за започињање комуникације. У заглављу је написано име пријављеног корисника.

По започињању комуникације се добија прозор за комуникацију, приказан на сликама 3.3.5. до 3.3.7. У заглављу прозора исписано је име особе са којом се комуницира, затим велико текстуално поље за приказ свих размењених порука, текстуално поље за унос поруке и дугме за слање поруке. У колико се корисник са којим се комуницира одјави или икључи прозор за комуникацију о томе се добија порука о онеспособљава се дугме за слање порука.

На сликама 3.3.3. до 3.3.7. је приказан сценаријо где су на систем пријављена два корисника Мика и Пера. Мика започиње комуникацију са Пером и шаље му поруку „Hello!“. Пера одговара са неколико порука, од којих је једна мало дужа, а затим искључује свој прозор за комуникацију. Мика прима све те поруке и на карају поруку „Pera Disconnected…“, чиме се сигнализира крај њихове комуникације.

## Код

Комплетна апликација се сатроји из два дела сервера и клијента. Улога сервера јесте да аутентификује клијенте и генерише све потребне кључеве. Корисници могу да се пријаве и комуницирају једни са другима. Иако су њихове улоге потпуно различите постоје неки делови који су идентични. Код апликације ће бити разматран у три целине: заједнички делови, серверски и клијентски део. Делови који се врло мало разликују биће у заједничком делу уз напомену на то где се разликују.

### Заједнички делови

Како је акценат овог пројекта постављен на шифровању порука прво ће бити објашњена класа MyCrypt која је задужена за шифровање и дешифровање порука, као и генерисање кључева и иницијалних вектора потребних за шифровање. Сами алгоритми и класе неопходни за испуњавање свих функционалносит класе су већ дефинисани у библиотеци Crypto++ и овде су само имплементирани. Заглавље mycryp.h изгледа овако:

#include<crypto++/aes.h>

#include<crypto++/cryptlib.h>

#include<crypto++/filters.h>

#include<crypto++/osrng.h>

#include<cryptopp/modes.h>

#include<crypto++/hex.h>

#include<QByteArray>

#include<QDebug>

usingnamespaceCryptoPP;

classMyCrypt

{

private:

bytekey[AES::MAX\_KEYLENGTH];

byteiv[AES::BLOCKSIZE];

public:

MyCrypt();

QByteArrayencrypt(QByteArraydata);

QByteArraydecrypt(QByteArraydata);

staticQByteArraymakeKey();

staticQByteArraymakeIV();

voidsetKeyIv(std::stringkey,std::stringiv);

byte\*getKey(){returnkey;}

};

На самом почетку укључења сва неопходна заглавља, која су већ објашњена у ранијем тексту. Класа има два атрибута типа byte то су key и iv величине256 и 128, те вдености су дефинисане у заглављу aes.h, који предтављају кључ и иницијализациони вектор. Даље постоји празан конструктор, методе encrypt и decrypt које служе за шифровање и дешифровање, статичне методе makeKey и makeIv за генерисање кључа и иницијализационог вектора, метода setKeyIv која поставља кључ и иницијализациони вектор.

Прво ће бити објашњена метода за шифровање текста.

QByteArrayMyCrypt::encrypt(QByteArraydata){

try{

CBC\_Mode<AES>::Encryption\*e=newCBC\_Mode<AES>::Encryption;

e->SetKeyWithIV(key,sizeof(key),iv);

std::stringchiper1,chiper2;

chiper1.clear();

chiper2.clear();

StringSource(data.data(),true,

newStreamTransformationFilter(\*e,new StringSink(chiper1),StreamTransformationFilter::PKCS\_PADDING));

chiper2.clear();

StringSource(chiper1,true,newHexEncoder(newStringSink(chiper2)));

data.clear();

data.append(chiper2.c\_str());

deletee;

returndata;

}catch(constCryptoPP::Exception&e)

{

qDebug()<<e.what();

}

returnNULL;

}

Пво се преко показивача инстанцира нови објекат класе Encryption, који представља AES алгоритам у CBC моду, и поствљају му се кључ и иницијализациони вектор. Потребна су нам два стринга због тога што све функције из библиотеке Crypto++ раде са објектима типа std::string а Qt framework користи своје QString и QByteArray , због чека је често неопходна конверзија између ових типова. Прво се преко StringSouce, StringTransformatinFileter-a и StringSink прослеђени текст шифрује. Затим се шифрат преставља у hex формату. Овај други корак је неопходан због размене продатака. Током шифровања могуће је да се негде у тексту појави карактер '\0' који представља крај стринга, што може довести до лошег читања података а онда и до проблема са дешифровањем. Представљањем шифрата у hex формату овај проблем се елиминише и за то се користи HexEncoder. Затим се тако добијени string враћа. Све ово се налази унутар try-catch блока који ће ухватити било коју грешку. Битно је напоменути да ће StringSink и сви његови параметри аутоматски бити избрисани због тога што Crypto++ користи свој pipeline механизам, док је Encription неопходно ручно обрисати како не би дошло до цурења меморије.

Метода за дешифровање текста:

QByteArrayMyCrypt::decrypt(QByteArraydata){

try{

CBC\_Mode<AES>::Decryption\*d=newCBC\_Mode<AES>::Decryption;

d->SetKeyWithIV(key,sizeof(key),iv);

std::stringchiper1,chiper2;

chiper1.clear();

chiper2.clear();

chiper2=data.data();

StringSource(chiper2,true,newHexDecoder(newStringSink(chiper1)));

chiper2.clear();

StringSource(chiper1,true,

newStreamTransformationFilter(\*d,newStringSink(chiper2)));

data.clear();

data.append(chiper2.c\_str());

deleted;

returndata;

}catch(constCryptoPP::Exception&e)

{

qDebug()<<e.what();

}

returnNULL;

}

Ова метода функционше скори идентично као метода за шифровање текста. Главна разлика је у томе што се корисити Decryption и што се прво текст из hex формата преводи у стандардни а онда дешифрује. Процес је у суштини супротан од процеса шифровања.

Методе за генерисање кључа и иницијализационог вектора:

QByteArrayMyCrypt::makeKey(){

AutoSeededRandomPool\*prng=newAutoSeededRandomPool;

bytecKey[AES::MAX\_KEYLENGTH];

prng->GenerateBlock(cKey,sizeof(key));

std::stringstrKey;

StringSource(cKey,AES::MAX\_KEYLENGTH,true,newHexEncoder(newStringSink(strKey)));

deleteprng;

returnQByteArray(strKey.c\_str());

}

QByteArrayMyCrypt::makeIV(){

AutoSeededRandomPool\*prng=newAutoSeededRandomPool;

bytecIV[AES::BLOCKSIZE];

prng->GenerateBlock(cIV,sizeof(cIV));

std::stringstrIV;

StringSource(cIV,AES::BLOCKSIZE,true,newHexEncoder(newStringSink(strIV)));

deleteprng;

returnQByteArray(strIV.c\_str());

}

Објекат класе AutoSeededRandomPool се користи за генерисање насумичних ведности помоћу методе GenerateBlok, којој се прослеђује величина блока који желимо да буде генерисан. И кључ и иницијализациони вектор се представљају у hex формату из истог разлога као и шифрат.

Метода за постављање кључа и иницијализационог вектора:

voidMyCrypt::setKeyIv(std::stringstrKey,std::stringstrIv){

qDebug()<<strKey.c\_str();

qDebug()<<strIv.c\_str();

std::stringsKey,sIv;

StringSource(strKey,true,newHexDecoder(newStringSink(sKey)));

StringSource(strIv,true,newHexDecoder(newStringSink(sIv)));

memcpy(key,sKey.c\_str(),AES::MAX\_KEYLENGTH);

memcpy(iv,sIv.c\_str(),AES::BLOCKSIZE);

}

Све линије које почињу са qDebug служе само за дибаговање програма, односно исписивање информације у конзоли, које су корисне теком развоја апликације и требале би да буду уклоњене када апликација крене да се користи. Функцијом memcpy се прослеђене вредмости уписују у променљиве key и iv, након што се конвертују из hex у обичан формат.

Следећа заједничка класа јесте MyServer. Сервер прихвата долазеће конекције и за сваку инстанцира нови сокет преко којег се врши даља комуникација. Ипак постоје неке кључне разлике код серверске и клијентске класе. Прва и највађнија јесте у томе што серверска класа користи QSslSocket-e и има много више функционалности, док клијентска класа котисти QTcpSocket-e и само прихвата нове конекције.

Заглавље класе MyServer на серверској страни изгледа овако:

classMyServer:publicQTcpServer

{

Q\_OBJECT

public:

MyServer(QTextEdit\*,QListWidget\*,QObject\*parent=nullptr);

voidStartServer();

~MyServer();

protected:

voidincomingConnection(qintptrsocketDescriptor);

private:

QSslKey\*sslKey;

QSslCertificate\*sslCertificate;

ClientArray\*clients;

QTextEdit\*log;

QListWidget\*userList;

QStringListonlineUsers;

publicslots:

voidsendOnlineUsers(MyClient\*);

voidmessageReceived(QByteArray,MyClient\*);

voidloggedIn(QByteArray,MyClient\*,QString,QString,QByteArray,QByteArray);

voidkeyGenerated(QByteArray,MyClient\*,QString);

voidclientDisconected(QString);

voidlogMessage(QString);

signals:

voiduserAddRem(bool,QString);

voidlogData(QString);

};

Сва неопходна заглавља су наравно укључена. Клас наслеђује класу QTcpServer. Сви неопходни атрибути класе су декларисани, атрибуту QTextEdit и QListWidget су неопходни за манипулисање графичким окружењем. Такође сви неопходни signali и slotovi су декларисани.

Констурктор:

MyServer::MyServer(QTextEdit\*txt,QListWidget\*lst,QObject\*parent):QTcpServer(parent)

{

log=txt;

userList=lst;

boolok=true;

log->setReadOnly(true);

clients=newClientArray;

QByteArraykey;

QByteArraycert;

log->append("Loadingkey...");

QFilefile\_key("cert.key");

if(file\_key.open(QIODevice::ReadOnly))

{

key=file\_key.readAll();

file\_key.close();

}

else

{

ok=false;

log->append(file\_key.errorString());

}

sslKey=newQSslKey(key,QSsl::Rsa);

log->append("RSAkeyloaded");

log->append("Loadingcertificate...");

QFilefile\_cert("cert.crt");

if(file\_cert.open(QIODevice::ReadOnly))

{

cert=file\_cert.readAll();

file\_cert.close();

}

else

{

ok=false;

log->append(file\_cert.errorString());

}

sslCertificate=newQSslCertificate(cert);

log->append("Certificateloaded");

if(ok)

StartServer();

}

Конструктор на серверу је много конпликованији него код клијента. Прво поазивачи добију прослеђене вредности. Дефинише се једна променљива типа bool која служи као флег да ли је све прошло како треба. Затим се из фајлова учитавају сертификат и RSA кључ неопходни за SSL сокете. Цео процес је пропраћен порукама о статусу извршавања. На крају, у колико је све прошло како треба, се позива метода StartServer.

voidMyServer::StartServer(){

log->append("Startingserver...");

if(this->listen(QHostAddress::Any,1234)){

log->append("ServerStarted");

}else{

log->append("Servercouldnotstart");

}

}

Ова метода покреће сам сервер, односно поставља сервер да ослушкује на порту 1234. Исписује се порука о успешности покретања сервера.

Код клијента конструкор је празана, пошто нема потребе за учитавањем кључа и сертификата, а метода StartServer исто функционише, с' тим да јој се прослеђује број порта на који сервер треба да ослушкује.

Метода incomingConnection је дефинисан у класи QTcpServer и служи са прихватање долазећих конекција.

voidMyServer::incomingConnection(qintptrsocketDescriptor){

MyClient\*client=newMyClient(this);

client->setSocket(socketDescriptor,\*sslKey,\*sslCertificate);

connect(client,SIGNAL(msgRcv(QByteArray,MyClient\*)),

this,SLOT(messageReceived(QByteArray,MyClient\*)),Qt::QueuedConnection);

connect(client,SIGNAL(cliDisconeccted(QString)),

this,SLOT(clientDisconected(QString)),Qt::DirectConnection);

connect(client,SIGNAL(logMessage(QString)),

this,SLOT(logMessage(QString)),Qt::AutoConnection);

}

Прво се инстанцира објекта класе MyClient, која ће у дљем тексту бити објашњена а затим се сви одговарајући сигнали повезују са одговарајућим слотовима. На овај начин када стигне нека порука сервер може да је обради.

voidMyServer::messageReceived(QByteArraystr,MyClient\*cli){

MyTask\*task=newMyTask(str,clients,cli);

task->setAutoDelete(true);

connect(task,SIGNAL(loggedIn(QByteArray,MyClient\*,QString,QString,QByteArray,QByteArray)),

this,SLOT(loggedIn(QByteArray,MyClient\*,QString,QString,QByteArray,QByteArray)),Qt::QueuedConnection);

connect(task,SIGNAL(keyGenerated(QByteArray,MyClient\*,QString)),

this,SLOT(keyGenerated(QByteArray,MyClient\*,QString)),Qt::QueuedConnection);

connect(task,SIGNAL(disconnected(QString)),

this,SLOT(clientDisconected(QString)),Qt::QueuedConnection);

connect(task,SIGNAL(sendOnlineUsers(MyClient\*)),

this,SLOT(sendOnlineUsers(MyClient\*)),Qt::QueuedConnection);

QThreadPool::globalInstance()->start(task);

}

Ово је слот који обрађује догађај, сигнал, када стигне нека порука. Инстацира се објакт класе MyTask, чији се сигнали повезују са одговарајућим слотовима а затим се покреће у глобалном QThreadPool-у. Оно шта је овде битно јесте да свака GUI апликација има глобални ThreadPool у којем можемо да покрећемо наше Task-ове, чиме се постиже то да ће се сваки извршавати на посебној нити. То значи да није гарантовано да ће Task-ови бити извршавани оним редоследом којим су пристигли, нити када ће који бити завршен, али се знатно добије на брзини извршавања програма. Такође је могуће одредити максималан број нити у ThreadPool-у на којима ће се извршавати таскови, тако да никад не дође до преоптерећивања сиситем, али може да дође до мало дужег чекања у случају великог броја порука.

Код клијента се на овај сигнал само емитује други сигнал који се даље обрађује.

На серверу су такође дефинисани сви слотови који обрађују сигнале које емитује MyTask, који у главном шаљу одговоре кориснику који су генерисани у Task-у.

Последња заједничка класа јесте MyClient.

classMyClient:publicQObject

{

Q\_OBJECT

public:

explicitMyClient(QObject\*parent=0);

voidsetSocket(qintptrDescriptor,QSslKey,QSslCertificate);

voidsendMessage(QStringmessage);

voidsendMessage(constchar\*msg);

voidsendMessage(QByteArraydata);

QByteArrayencrypt(QByteArraydata);

voidsetCrypt(std::stringkey,std::stringiv);

QStringgetName(){returnname;}

voidsetName(QStringname){this->name=name;}

QStringgetAddrAndPort();

voidsetPort(qintptr);

~MyClient();

signals:

voidmsgRcv(QByteArray,MyClient\*);

voidcliDisconeccted(QString);

voidlogMessage(QString);

publicslots:

voidconnected();

voiddisconnected();

voidreadyRead();

voidready();

private:

QStringname;

qintptrsocketDescriptor;

QSslSocket\*socket;

qintptrport;

MyCrypt\*crypt;

boolisEncripted;

};

Ова класа служи за саму комуникацију. Садржи све неопходен податке, који се користе за идентификацију, објекта класе MyCrypt који обезбеђује да сва комуникација буде шифрована и сокет преко којег иде сама комуникација. Има сигнале потребен за комуникацију са сервером и слотове за обраду сигнала од сокета. Класа наслеђује класу QObject.

voidMyClient::setSocket(qintptrDescriptor,QSslKeykey,QSslCertificatecert){

socketDescriptor=Descriptor;

socket=newQSslSocket(this);

socket->setSocketDescriptor(socketDescriptor);

socket->setPrivateKey(key);

socket->setLocalCertificate(cert);

socket->setPeerVerifyMode(QSslSocket::VerifyNone);

connect(socket,SIGNAL(connected()),this,SLOT(connected()));

connect(socket,SIGNAL(disconnected()),this,SLOT(disconnected()));

connect(socket,SIGNAL(readyRead()),this,SLOT(readyRead()));

connect(socket,SIGNAL(encrypted()),this,SLOT(ready()));

socket->startServerEncryption();

}

Метода setSocket поставља све потребне подтке за сокет и повезује све потребне сигнале са слотовима. На крају се позива метода startServerEncryprion, која обезбеђује сигурну комуникацију преко SSL сокета. Ова метода се позива из методе incomingConnection код објекта MyServer класе.

voidMyClient::ready(){

emitlogMessage(QString::number(socketDescriptor)+"Clientconnected...");

}

Ово је сигнал који се емитује када се неко успешно повеже, односно након успешног руковања. Слично сигнал disconnected означава да се неко „откачио“.

voidMyClient::readyRead(){

QByteArraymsg=socket->readAll();

if(isEncripted)

msg=crypt->decrypt(msg);

emitmsgRcv(msg,this);

}

Када сокет емитује сигнал readyRead овај слот то обрађује. Прочита се долазна порука и, у колико је комуникација шифрована MyCrypt објектом, дешифрује се, и на крају се емитује сигнал који се обрађује на серверу.

voidMyClient::sendMessage(QByteArraydata){

if(isEncripted)

data=crypt->encrypt(data);

socket->write(data);

}

Када се поруке шаљу такође се проверава да ли је комуникација шифрована и, у колико јесте, пре слања се порука шифрује.

voidMyClient::setCrypt(std::stringkey,std::stringiv){

crypt=newMyCrypt;

crypt->setKeyIv(key,iv);

isEncripted=true;

}

Овом методом се комуникација додатно осигурава шифровањем MyCrypt објектом.

Код клијента ова класа је много једноставнија. Користи се QTcpSocket сокет и класа само поставља сокет, повезује сигнале, којих је мање, и када прими поруку прослеђује је даље на обраду.

### Серверски део

Сервер, поред тога што су код њега заједничке класе много сложеније, има и своју класу MyTask. Суштина ове класе јесте да се сва обрада обавља асинхроно, на одвојеним нитима, чиме се добија на перформансама. То се постиже у комбинацији са ThreadPool-ом. Пошто се Task-ови извршавају на одвојеним нитима они не могу да приступе сокету и пошаљу податке, то мора сервер који је на главној нити, али зато Task-ови могу да изврше сву потребну обраду и серверу проследе готове поруке за слање. Најбитнија метода јесте метода run од које почиње извршавање Task-a.

classMyTask:publicQObject,publicQRunnable

{

Q\_OBJECT

private:

MyClient\*cli;

QByteArraystr;

ClientArray\*clients;

QStringListlist;

voidLogIn();

voidstartConversation();

voidgenerateKey();

voidLogOff();

public:

MyTask(QByteArray,ClientArray\*,MyClient\*);

~MyTask();

protected:

voidrun();

signals:

voidloggedIn(QByteArray,MyClient\*,QString,QString,QByteArray,QByteArray);

voidkeyGenerated(QByteArray,MyClient\*,QString);

voiddisconnected(QString);

voidsendOnlineUsers(MyClient\*);

};

Класа има све потребне променљиве, које се просеђују преко конструктора, методе које обрађују поруке, различите методе обрађују различите поруке, и сигнале који се обрађују у серверу.

voidMyTask::run(){

QStringmessage=str.data();

list=message.split('');

if(list[0]=="LOGIN")LogIn();

elseif(list[0]=="LOGOFF")LogOff();

elseif(list[0]=="CONN")startConversation();

elseif(list[0]=="USRLIST"){

cli->setPort(list[1].toLongLong());

emitsendOnlineUsers(cli);

}

}

Метода run од које почиње извршавање Task-a само дели поруку на делове и у завриснити од поруке позива одговарајућу методу за обраду поруке.

voidMyTask::LogIn(){

QStringlogstr="Loginstarted\n"+list[1]+'';

QStringname="NO";

QByteArrayconnKey,connIv;

QByteArrayresponse;

if(ClientList::usernames.contains(list[1])&&

ClientList::passwords.at(ClientList::usernames.indexOf(list[1]))==list[2]){

if(clients->contains(list[1])){

response.append("FAIL1");

logstr+="Loginfailed:Useralreadyloggedin!";

}else{

cli->setName(list[1]);

clients->add(cli);

name=list[1];

connKey=MyCrypt::makeKey();

connIv=MyCrypt::makeIV();

logstr+="Loginsucessful!";

response.append("SUCESS"+connKey+''+connIv);

}

}else{

logstr+="Loginfailed:Wrongusername/password!";

response.append("FAIL2");

}

emitloggedIn(response,cli,name,logstr,connKey,connIv);

}

Ова метода се позива у колико се корисник пријављује. У колко је корисник унео исправне податке, шаље му се потврдна порука заједно са кључем и иницијализационим вектором. Исте те вредности се постављају и на серверској страни.

voidMyTask::startConversation(){

QByteArrayresponse;

QStringlogstr;

if(clients->contains(list[1])&&clients->contains(list[2])){

QByteArrayconnKey=MyCrypt::makeKey();

QByteArrayconnIv=MyCrypt::makeIV();

response.append("KEY"+list[2]+'');

response.append(connKey+'');

response.append(connIv+'');

response.append(clients->operator[](list[2])->getAddrAndPort()+'');

QByteArrayresponse2;

response2.append("KEY"+cli->getName()+''+connKey+''+connIv);

response2=clients->operator[](list[2])->encrypt(response2);

response.append(response2);

logstr=response.data();

}else{

response.append("FAIL3");

logstr=response.data();

}

emitkeyGenerated(response,cli,logstr);

}

Метода која се позива када корисници започињу комуникацију. У колико су оба корисника пријављена, ово мора да се провери због асинхроног извршавања, и генерише се порука која се састоји из два дела. Први део се сатоји од речи KEY, имена корисника са којим је захтевана комуникација, кључа, иницијализационог вектора, адресе и порта корисника са којим се захтева комуникација. Други део се састоји из речи KEY, имена корисника који захтева комуникацију, кључа и иницијализационог вектора. Овај део је намењен другом кориснику и шифрован је његовим кључем.

### Клијентски део

Клијентски део апликације је доста једноставнији од серверског, с' обзиром на то да сва обрада на серверском делу. Клијент, као што је већ речено, има класу MyCrip која је иста као на серверу, MyServer и MyClient које су веома сличне по својој функционалности. Главна улога ове две класе јесте у прихватању нових конекција, а размена и обрада порука се прослеђује другим деловима. Битна разлика је у томе што код клијента оне користе QTcpSocket класу за сокете, што значи да комуникација преко њих није сигурна. Поред ових класа корисник има класу SslClient, која служи само за комуникацију са сервером. Ова класа кориси QSslSocket, као класа MyClient на серверу.

Највећа разлика јесте у томе што клијентски део има Chat класу, која служи за размену порука између корисника.

classChat:publicQDockWidget

{

Q\_OBJECT

public:

explicitChat(QTcpSocket\*,QString,QByteArray,QByteArray,QWidget\*parent=0);

~Chat();

private:

QTcpSocket\*socket;

MyCrypt\*crypt;

Ui::Chat\*ui;

publicslots:

voidread();

voidsend();

};

Као што се види класа је прилично једноставана. Има само сокет и објекат класе MyCrypt, конструктор и слотове за читање и слање порука.

Chat::Chat(QTcpSocket\*socket,QStringname,QByteArraykey,QByteArrayiv,QWidget\*parent):

QDockWidget(parent),

ui(newUi::Chat)

{

ui->setupUi(this);

ui->txtDisplayMessages->setReadOnly(true);

this->socket=socket;

this->setWindowTitle(name);

crypt=newMyCrypt(key.data(),iv.data());

connect(this->socket,SIGNAL(readyRead()),

this,SLOT(read()),Qt::DirectConnection);

connect(this->ui->btnSend,SIGNAL(released()),

this,SLOT(send()),Qt::DirectConnection);

}

Chat::~Chat()

{

deleteui;

}

voidChat::read(){

ui->txtDisplayMessages->appendPlainText(crypt->decrypt(socket->readAll()));

}

voidChat::send(){

QByteArraydata;

data.append(ui->txtMessage->text());

data=crypt->encrypt(data);

socket->write(data);

ui->txtMessage->setText("");

}

Као параметри констуктора јој се прослеђују сокет, име корисника, кључ и иницијализациони вектор, чиме се одмах у конструктору све иницијализује и повезују потребни сигнали и слотови.

## Недостаци

Ова апликација је замишљена као демонстрација коришћења АЕЅ алгоритма за остваривање шифроване комуникације и као таква пуна је пропуста и недостатака па није намењена за комерцијалну употребу.

Први, и нјавећи, сигурносни ризик јесте то што сервер има лог у којем се приказују све поруке које он генерише. Идеја је да се генерисани кључеви нигде не чувају, како нико не би могао да дође до њих и злоупотреби их. Јавља се и питање поверења, да ли корисник може да верује серверу да види његове тајне кључеве. Како је ово демонстративана апликација, лог постоји ради лакшег праћења догађаја на серверу. Следећа ствар код серверске стране је то што у овом делу у опште не постоји потреба за графичким интерфејсом. Како не постоји потреба за приказом свих порука сервер је боље реализовати као конзолну апликацију која само приказује статусне податке и евентуално има опцију за излиставање пријављених корисника. Графика, такође, додатно оптерећује систем и, мада у овом случају не претерано, може да доведе до смањења перформанси. Код сервера који треба да обрађује огроман број захтева треба максимално смањити непотребне процесе како би сервер максимално користио ресурсе за битне ствари.

Следећи сигурносни пропуст јесте у томе што се лозинке чувају и шаљу о облику отвореног текста. Уместо тога требале би да се користе вредности хеш функција. Још један недостатак је то што се листа корисника чува унутар саме апликације чиме се онемогућује додавање нових корисника, ондносно свака промена би траја само до искључивања сервера, при поновном покретању све би се вратило на старо. Ови подаци би због тога могли да се чувају у некој бази податада на коју би сервер био повезан. На тај начин не само да би се решио проблем чувања података о корисницима већ би се убрзало и претраживање у случају великог броја корисника.

У случају да клијентска апликација не може да се повеже на сервер кориснику се не даја никаква прецизнија порука о томе и једни начин да се поново покуша повезивање јесете да се апликација искључи и поново укључи.

# Закључак

Популаризација интернета је са собом донела велике предности и у великој мери олакшала наше свакодневне животе. Данас је врло лако ступити у контакт са неком удаљеном особом, било преко мобилне мреже или преко интернета, коришћењем е-маил порука или нечег другог. Овај вид комуникације иако је врло лак и брз носи са собом и одређене ризике, од којих су неки присутни у свакој врсти комуникације (прислушкивање), и већина корисника о томе и не размишља.

У овом раду је приказана реализација апликације која која би требала да обезбеди сигруну комуникацију између корисника. Цео процес од генерисања кључа до шифровања и дешифровања порука треба да буде транспарентан за корисника и он треба да има утисак да нормално комуницира са другим корисником. Једино о чему корисник треба да води рачуна јесте да не заборави своје корисничко име и лозинку, које може једино администратор лично да промени. Цео процес прављења налога и добијања копије клијентске апликације би такође требало да се врши директно и лично а не преко интернта или телефона.

Иако је пуна недостатака, сам концепт је сличан концепту Керберос протокола који се користи у пракси, и апликација би могла уз довољно развоја и тестирања могла да се доведе на комерцијално употребљив ниво.

# Индекс појмова

backdoor

задња врата, 7, 8

DES, 6, 7, 8, 9, 10, 16

IETF, 12

KDC, 14

MIT, 14, 16

Rijndael, 9, 10, 18

RSA, 11, 12, 19, 27

SSL, 4, 11, 12, 13, 19, 20, 21, 27, 30, 37

TGS, 14, 15

TLS, 4, 12, 19, 37

UNIX, 14

Windows, 14, 18

АЕЅ, 9, 10, 14, 33

алгоритама

алгоритам, 3, 4, 6, 12, 19

Апликација, 4, 17

Асиметрични криптосистеми, 11

Блоковске шифре

блоковске шифре, 4, 6

Дафи-Хелманов алгоритам, 11

дешифровање, 3, 4, 6, 10, 11, 17, 18, 23, 24, 25

Дигитални потпис, 12

дифузије

дифузија, 6, 7, 10

Енигма, 4

иницијални вектор

Иницијални вектор, 11

Интегритет

интегритет, 5

Информације, 1, 4

једносмерне функције са замком, 11

Керберос, 4, 14, 16, 17, 35

клијент, 12, 13, 14, 15

кључ, 3, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15, 17, 19, 20, 21, 24, 25, 27, 32, 33

комплетности

комплетност, 6

комуникације, 2, 6, 17, 19, 21, 23, 33, 35

комуникационим каналом

комуникациони канал, 4, 11

конфузије

конфузија, 6, 7

Корисник

корисник, 14, 19, 20, 21

Крипто-анализа

крипто-анализа, 5

Криптографија

криптографија, 4, 5

Ојлерова функција, 12

Отворени текст

отворени текст, 4, 6

прислушкивања, 4, 14, 19

Простор кључева

простора кључева, 5

рачунари

рачунар, 2, 4

Режим рада блоковских шифара, 10

сервер, 3, 12, 13, 14, 15, 16, 17, 19, 20, 21, 23, 28, 30, 34

сертификат, 12, 13, 27

сигуран, 2, 3, 5, 6, 11, 12, 13, 14, 16

Симетрични крипотсистеми

симетрични крипотсистеми, 4, 6

систем, 2, 3, 5, 12, 17, 19, 20, 21, 23, 34

ТGT, 14

тајности

тајност, 5

тикетима

тикет, 14

Фејстел шифра

фејстел шифра, 6, 7, 9

Цезарова шифра, 4

Шифрат

шифрат, 4, 6

шифровање, 1, 2, 3, 4, 6, 11, 12, 13, 17, 18, 23, 24, 25

# Литература

[1] Ferguson, Niels; Schneier, Bruce (2003) *Practical Cryptography*

[2]Bryant, Bill (1988) *Designing an Authentication System: A Dialogue in Four Scenes*

[3]Rivest, Ronald L. (1990) *Cryptography* из J. Van Leeuwen *Handbook of Theoretical Computer Science*

[4]Bellare, Mihir; Rogaway, Phillip (2005) *Introduction to Modern Cryptography*

[5] Delfs, Hans & Knebl, Helmut (2007). "Symmetric-key encryption"*Introduction to cryptography: principles and applications*

[6] Pelzl & Paar (2010) *Understanding Cryptography*

[7]Eric Rescorla (2001) *SSL and TLS: Designing and Building Secure Systems*

[8]

[9]