Универзитет у Београду
Електротехнички факултет

Дипломски рад

Инфраструктура за сигурну размену података

Ментор:
Проф. др Захарије Радивојевић

Кандидат:
Душан Мршић 2017/0406

Београд, 2021.

Захвалница

Велику захвалност дугујем компанији SYRMIA у Београду што су ми омогућили ово истраживање. Веома сам захвалан менторима из фирме Саши Црнобрњи и Давору Јовановићу на несебичној помоћи, корисним сугестијама и пренетом знању.

Захваљујем се професору Захарију Радивојевићу на чијим сам предавањима научио много нових ствари из рачунарске науке. Хвала Вам на указаном поверењу и на конструктивним сугестијама.

На крају, захваљујем се својој породици и девојци Драгани, на великом разумевању и безусловној подршци свих ових година. Хвала вам што сте ме охрабривали и подстицали да истрајем у својим циљевима.

Садржај

Списак слика

Списак табела

1. Увод

У рачунарској науци сигурна размена података се односи на размену поверљивих или заштићених информација путем заштићених канала. Већина безбедних начина преноса захтева неку врсту шифровања информација. Да би било могуће читање шифрованих информација мора се извршити размена кључева. Многе инфраструктуре, попут банака, ослањају се на сигурне протоколе преноса како би спречиле катастрофално нарушавање безбедности.

End-to-end енкрипција је систем комуникације у којој само корисници који учествују у комуникацији могу читати поруке[1]. Спречава потенцијалне прислушкиваче, у које спадају и интернет провајдери и провајдери комуникационих услуга, да приступе кључевима који служе за дешифровање садржаја. Само прави пошиљалац и прималац имају могућност читања и измене података. Пошиљалац шифрује податке, и шаље примаоцу, он помоћу тајног кључа дешифрује податке, а трећа страна, то јест нападач, нема могућност да дешифрује и чита податке.

Технологија "тренутних порука" (*Instant messaging*) је врста комуникације која омогућава пренос порука на интернету у реалном времену[2]. Комуникација се обавља између две или више страна. Системи за размену порука могу бити самосталне апликације које служе само за комуникацију (*WhatsApp, Viber, Skype*) или могу бити интегрисане у ширу платформу нпр. друштвене мреже (*Facebook*).

Многи системи за размену порука користе end-to-end енкрипцију, укључујући *Facebook, Whatsapp, Zoom*. Ови провајдери су наишли на подељена мишљења око одлуке о усвајању E2EE. *End-to-end* енкрипција отежава провајдерима размену корисничких информација из њихових услуга са властима и потенцијално пружа приватну размену порука људима укљученим у недозвољене активности[3].

Циљ овог пројекта је развој веб апликације за сигурну размену текстуалних порука и датотека. Систем користи *end-to-end* енкрипцију за размену и чување података, што значи да нико сем клијената који учествују у комуникацији нема приступ подацима, чак ни сервер где се подаци чувају. Апликација ради по принципима по којима раде модерне *DRM (Digital Rights Managment)* технологије, а акценат је стављен на криптографију[4].

За имплементацију се користи стандардни скуп алата за развој веб апликација: *MEAN STACK*. За *frontend* се користи Angular, за *backend* Node.js и Express, а за чување података база *MongoDB*[5].

Овај рад се састоји из 4 поглавља где прво поглавље представља увод у сами рад и у кратким цртама даје општи увод у тему која ће бити обрађена у раду.

Друго поглавље се бави историјом шифровања, поделом и алгоритмима шифровања, упоређивањем одређених алгоритама шифровања. Упознаје читаоце са општим особинама end-to-end енкрипције и бави се интеграцијом *end-to-end* енкрипције у апликацијама за дописивање.

Треће поглавље престављa конкретну имплементацију решења. Даје увид у технологије које су коришћене приликом имплементације и инфраструктуру саме апликације. У овом поглављу је и кратко корисничко упутство.

Последње, четврто поглавље је закључак. На почетку имамо кратки осврт на конкретно решење апликације, а након тога предлоге за даље унапређивање апликације.

2. Интеграција апликација за дописивање и шифровања

У овом поглављу ће бити обрађен историјски развој шифровања – од античког доба па све до данас. Биће обрађено симетрично и асиметрично шифровање, биће наведене предности и мане, и разлике оба начина шифровања. Након тога, алгоритми шифровања и предности и мане неких од најзначајнијих алгоритама. И на крају увод у *end-to-end* енкрипцију.

2.1 Шифровање

Када неке податке пошаљемо преко интернета ми немамо скоро никакву контролу над онима ко рукује тим подацима, па самим тим ни приватност над тим подацима. Подаци пролазе кроз различите непознате сервере, рутере и разне уређаје где их може пресрести било ко. Поставља се питање како заштитити своје податке? Приватност се постиже укључивањем шифрирања у цео процес размене података.

Са шифрирањем се сусрећемо још од античких времена, када је шифрирање коришћено у војне сврхе. Широм античке Грчке и Рима коришћена је замена симбола, па је била потребна шифра да би се текст протумачио. Један од најпознатијих начина шифровања из тог времена је "Цезарова шифра", где се свако слово у тексту помера за фиксни број места у абецеди. Временом су људи проналазили начин да дешифрују шифроване текстове, па су тако временом настајали све компликованији системи шифровања.

За време Другог светског рата коришћена је "Енигма машина", где се куцањем по тастатури добијао шифровани текст, уз помоћ ротирајућих дискова. Али и Енигма је дешифрована од стране пољских математичара коришћењем електромеханичих рачунара.

У модерно време шифровање се највише користи за слање различитих података преко интернета. Модерно шифровање је кључ напредне рачунарске и комуникацијске сигурности. Овај ток криптографије у потпуности је заснован на идејама математике, као што су теорија бројева и теорија рачунске сложености, као и концептима вероватноће. Модерни алгоритми шифровања се деле на симетричне и асиметричне[6].

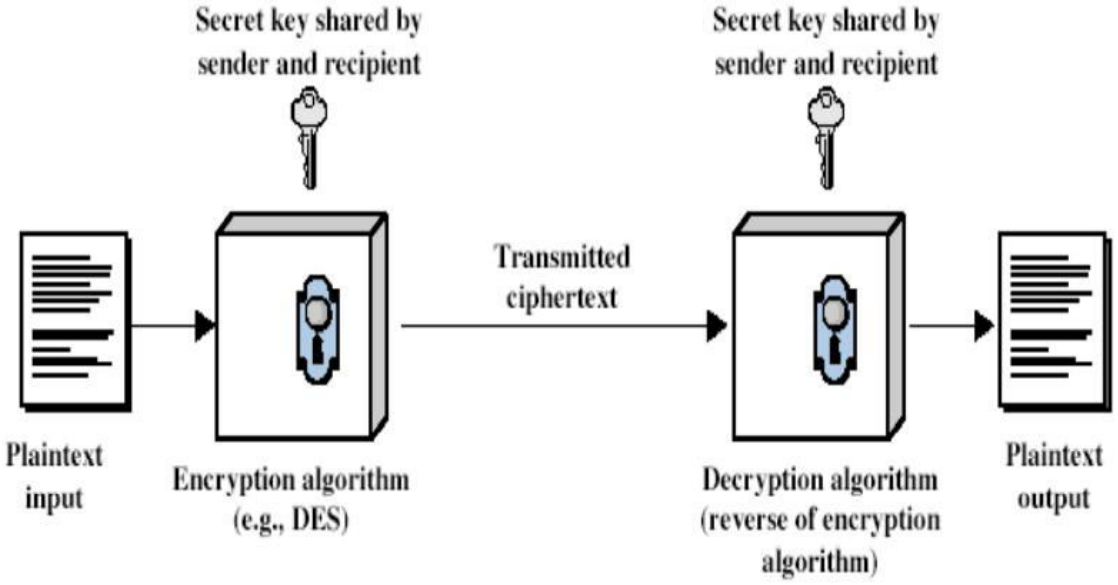
2.1: Енигма машина

2.2 Симетрично и асиметрично шифровање

Симетрично шифровање је широко распрострањена техника шифровања код које се подаци шифрују и дешифрују помоћу једног, тајног криптографског кључа. Симетрично шифровање ради тако што користи шифру тока (*stream cipher*) или блок шифру (*block cipher*) за шифровање и дешифровање података. Шифра тока претвара текст у шифровани текст бајт по бајт, а блоковска шифра претвара читаве јединице или блокове текста користећи унапред одређену дужину кључа, као што су 128, 192 или 256 бита. Пошиљаоци и примаоци који користе симетрично шифровање за међусобну комуникацију морају знати тајни кључ како би, у случају пошиљалаца, шифровали податке, а у случају прималаца дешифровали и прочитали шифроване податке.

Предности коришћења симетричног шифровања пре свега брзина, због кратких кључева и једноставнији су од асиметричних алгоритама. Сигурност у погледу *brute-force* напада, јер је потребно много времена да се на тај начин провали шифра. Симетрични алгоритми су постали стандард шифровања података, због своје брзине и сигурности и годинама су добро прихваћени од стране разних индустрија.

Главна мана симетричног шифровања је коришћење једног кључа, јер он мора бити сигурним путем прослеђен учесницима у комуникацији, а то није увек једноставно, и мора бити добро чуван, у супротном ће кључ бити компромитован[7].

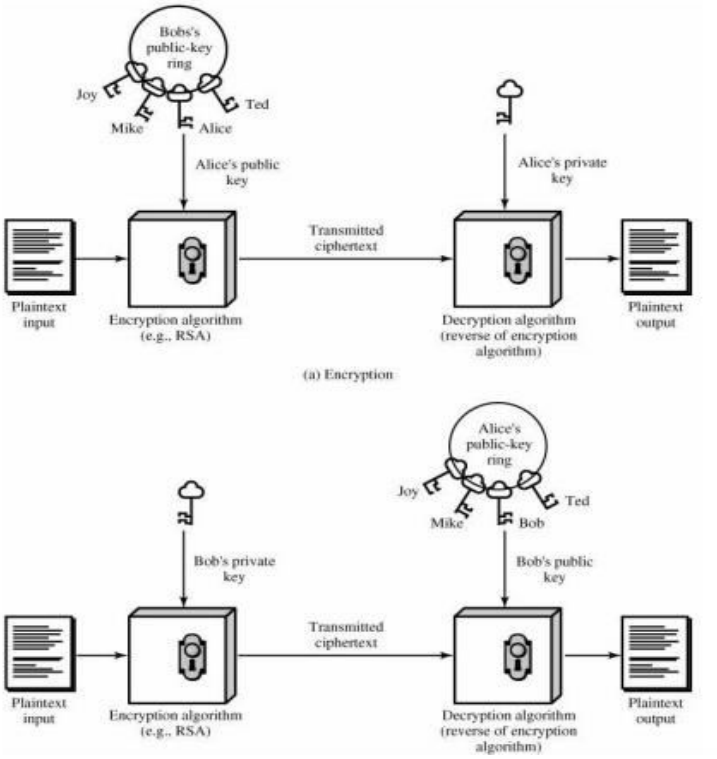


2.2: Модел симетричног шифровања

Асиметрично шифровање за разлику од симетричног, које користи један кључ, користи пар математички повезаних кључева који служе са шифровање и дешифровање података. У комуникацији се користи јавни и приватни кључ. Јавни кључ је свима познат и он служи за шифровање података, док се дешифровање врши само приватним кључем примаоца. Пошиљалац користи јавни кључ примаоца и њиме шифрује податке, док прималац својим приватним кључем дешифрује податке.

Главна предност асиметричног у односу на симетрично шифровање је да размена кључева није потребна, јер се јавни кључ прочита се сервера за јавне кључева, док размена приватних није потребна. Такође, код асиметричног шифровања је могуће потписивање садржаја који се шаље, па прималац може да сазна да ли да порука долази он неке треће стране или од правог пошиљаоца.

Мана асиметричног шифровања је брзина. У општем случају кључеви су дугачки и алгоритми шифровања су комплекснији у односу на симетрично шифровање[8].



2.3: Модел асиметричног шифровања

2.3 Алгоритми шифровања

Као што је речено у претходном поглављу, алгоритми се деле на симетричне и асиметричне. У симетричне алгоритме шифровања спадају: *AES*, *DES*, *Triple DES*, итд. А у асиметричне спадају: *RSA*, *Diffie-Hellman*, итд.

DES је симетрични блоковски алгоритам са дужином кључа од 56 бита. Објављен 1977. године, а званично је повучен 2005. године. *DES* је развијен пре више од 35 година како би се обезбедила криптографска сигурност за све владине комуникације. Идеја је била да се осигура да сви владини системи користе исти, сигуран стандард како би се олакшала међусобна повезаност и комуникација.

Да би се показало да *DES* више није сигуран, од 1997. до 1999. године је организовано 3 такмичења у "разбијању" *DES* алгоритма, где је *DES* разбијен за 22 сата и 15 минута и званично показано да ДЕС није сигуран.

Triple DES представља алгоритам где се *DES* користи три пута. Али и поред своје комплексности у односу на *DES*, показано је да и *Triple DES* није отпоран на *brute-force* нападе. *Triple DES* је до 2018. године избачен из употребе.

AES је објављен 2000. године. Главна моћ *AES* алгоритма лежи у томе да је могуће изабрати различите дужине кључева. Могуће је изабрати 128-битни, 192-битни или 256-битни кључ, што га чини експоненцијално јачим од 56-битног кључа *DES* алгоритма. Што се тиче структуре, *DES* користи Фистел структуру која дели блок на две половине пре него што прође кораке шифровања. *AES*, с друге стране, користи пермутацију и супституцију, која укључује низ корака супституције и пермутације за креирање шифрованог блока.

2.1: Упоређивање ДЕС и АЕС алгоритама

	DES	*AES*
Објављен	1977	2000
Дужина кључа	56 бита	128,192 или 256 бита
Тип шифровања	Блоковски алгоритам	Блоковски алгоритам
Величина блока	64 бита	128 бита
Сигурност	Доказано несигуран	Сматра се сигурним

RSA је асиметрични алгоритам шифровања осмишљен на МИТ-у 1977. године. *RSA* користи јавни кључ за шифровање, а приватни за дешифровање. Јавни кључ се добија као производ два огромна проста броја. Да би се добио приватни кључ морају се знати два главна фактора тог производа, с тога само креатор јавног кључа може генерисати приватни кључ.

Diffie-Hellman је објављен 1976. године. Представља алгоритам за јавну размену тајних кључева. Алгоритам се ослања на просте бројеве приликом израчунавања тајног кључа. Постоји јавни кључ који је свима познат, и онда два учесника у комуникацији рачунају тајни кључ помоћу својих приватних[9,10,11,12].

2.4 End-to-End енкрипција (EE2E)

End-to-End енкрипција представља сигуран начин комуникације који спречава сваког ко није учесник у комуникацији да приступи подацима. *End-to-End* енкрипција користи асиметрично шифровање за заштиту података. У онлајн комуникацији готово увек постоји посредник између страна које комуницирају, који прослеђује поруке од једног учесника до другог. Асиметрично шифровање онемогућава посредника да чита и мења садржај порука.

Постоји неколико потенцијалних слабости које могу ометати сигурну размену податка. На првом месту: метаподаци. Помоћу EE2E онемогућен је приступ самој поруци, док су подаци о поруци и даље познати, као што су време и датум слања поруке. То потенцијално може дати информацију где је могуће пресрести поруку када је она декриптована. Компромитоване крајње тачке такође дају приступ нешифрованим порукама. Нападач може да прочита нешифровану поруку пре слања или дешифровану након пријема. Такође, ако нападач компромитовањем крајње тачке дође до кључа може извршити *man in the middle* напад. Још један проблем представља то што неки провајдери тврде да нуде *End-to-End* енкрипцију, а у ствари је то ближе шифровању у транзиту, то јест, до примаоца стижу дешифровани подаци који се чувају на неком серверу, а дешифрује их посредник, а не сам прималац.

Главна предност end-to-end шифровања је висок ниво сигурности података, који је омогућен следећим карактеристикама:

- **Сигурност у транзиту –** *End-to-end* енкрипција користи криптографију јавног кључа, која чува приватне кључеве на уређајима крајњих тачака комуникације. Дешифровање порука се може извршити само помоћу ових кључева, па само људи са приступом уређајима крајњих тачака могу да прочитају поруку.
- **Отпорно на неовлашћену измену података** - Код E2EE кључ за дешифровање не мора да се преноси; прималац ће га већ имати. Ако се порука шифрована јавним кључем промени или преиначи током преноса, прималац неће моћи да је дешифрује, па се фалсификовани садржај неће моћи видети.
- **Сагласност** - Многе индустрије су везане регулаторним законима који захтевају одређену сигурност података. *End-to-end* енкрипција може помоћи организацијама да заштите те податке чинећи их нечитљивим.

Недостаци *End-to-end* енкрипције су следећи:

- **Сложеност у дефинисању крајњих тачака** - Неке имплементације E2EE дозвољавају да се шифровани подаци дешифрују и поново шифрују на одређеним тачкама током преноса. Због тога је важно јасно дефинисати и разликовати крајње тачке комуникационог система.
- **Превише приватности** - Владе и агенције за спровођење закона изражавају забринутост *End-to-end* енкрипција може заштитити људе који деле недозвољени садржај, јер пружаоци услуга нису у могућности да омогуће полицији приступ садржају.
- **Видљиви метаподаци** - Иако су поруке шифроване и немогуће их је прочитати, информације о поруци - датум слања и прималац, на пример - и даље су видљиве, што може пружити корисне информације ометачу.
- **Заштита крајњих тачака** - Ако су крајње тачке компромитоване, могу се открити шифровани подаци.
- **Није поуздана за будућност** - Иако је end-to-end енкрипција сада јака технологија, постоје спекулације да ће је на крају квантно рачунарство учинити застарелом.

2.5 Апликације које користе енкрипцију

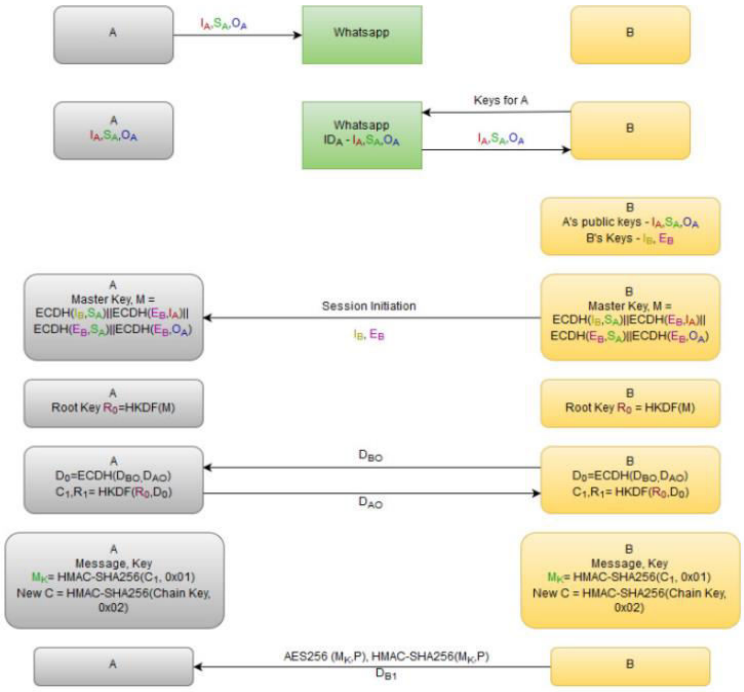
У овом поглављу ће бити говора о апликацијама за дописивање које су интегрисале end-to-end енкриптовање у своју инфраструктуру. О алгоритмима шифровања које те апликације користе, и о инфраструктури за размену порука. На крају ће бити упоређивање апликација на основу битних функционалности које имају савремене апликације за дописивање.

2.5.1 WhatsApp

WhatsApp је бесплатна апликација за размену порука и позива преко интернета у власништву *Facebook* компаније. Објављена је 2009. године, а од 2016. године њена инфраструктура се заснива на *end-to-end* енкрипцији.

Као парове јавних кључева апликација користи Curve25519. За размену кључева се користи *Diffie-Hellman* алгоритам, а као симетричну енкрипцију порука користи AES алгоритам. Сигурна размена се заснива на Сигнал протоколу.

За размену порука између два учесника у комуникацији мора се прво успоставити енкриптована сесија, и када се једном успостави није је потребно опет успостављати за поновну комуникацију. Приликом логовања корисник шаље свој кључ за идентификацију, и "кључ за једнократну употребу". Пошиљалац када жели да отпочне комуникацију од сервера затражи те јавне кључеве. Онда шаље своје јавне кључеве примаоцу. Сложеним алгоритмима се рачуна заједничка тајна која се служити за даљу комуникацију. На слици у прилогу се види детаљна шема комуникације[14].



2.4: Инфраструктура WhatsApp апликације

2.5.2 Facebook Messenger

Facebook Messenger је апликација за размену порука у оквиру друштвене мреже *Facebook*. Објављена је 2011. године, а 2016. године су увели енкрипцију. Разлика у односу на друге апликације за дописивање је та што се енкрипција мора мануелно активирати опцијом *"Secret Conversation"* из менија за подешавања.

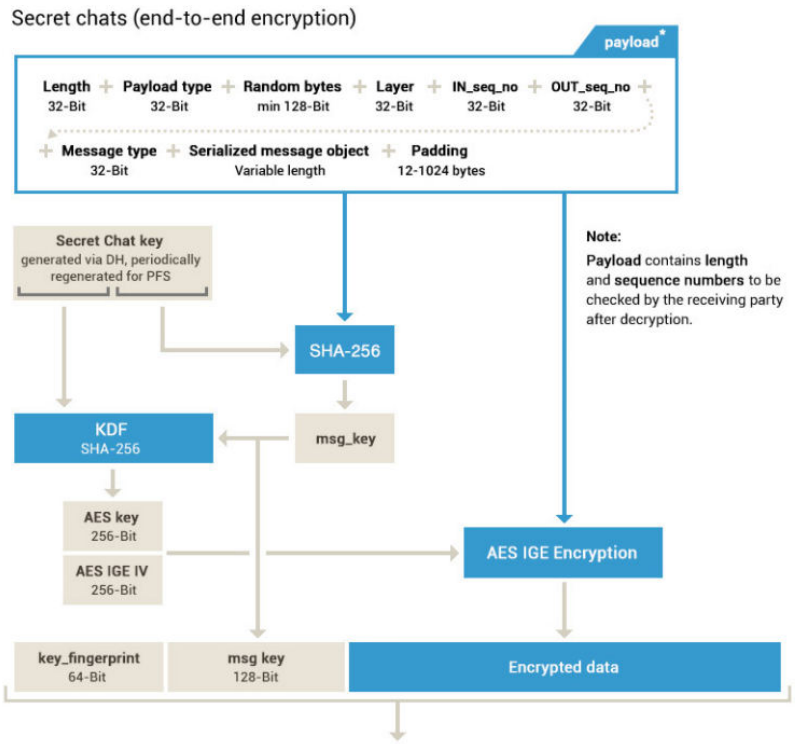
Пошто је *Facebook* власник апликације *WhatsApp*, *Facbook* користи исти протокол сигурне размене као и *WhatsApp*, Сигнал протокол. За парове јавних кључева се користи *Curve25519*. За енкрипцију се користи *AES256* алгоритам. А аутентикација се обавља помоћу *HMAC-SHA256* алгоритма.

2.5.3 Telegram

Апликација *Telegram* је објављена августа 2013. године. Намењена је размени порука и позива између корисника апликације. У свом систему има интегрисану *end-to-end* енкрипцију.

За размену и генерисање кључева се користи *Diffie-Hellman* алгоритам, а за криптовање порука *AES-256* алгоритам.

Када корисник А жели да отпочне разговор са корисником Б, он прво добија *Diffie-Hellman* параметре. Клијент А израчунава 2048-битни број и шаље захтев за комуникацијом клијенту Б. Он може да одбије или прихвати захтев. Након што прихвати захтев, клијент Б такође добија *Diffie-Hellman* параметре. Б добија број израчунат од стране А, и шаље свој број клијенту А, након тога оба клијента могу да израчунају дељени кључ. Дељени кључ даље служи за рачунање кључа за криптовање порука, а за то се користи AES256 алгоритам.

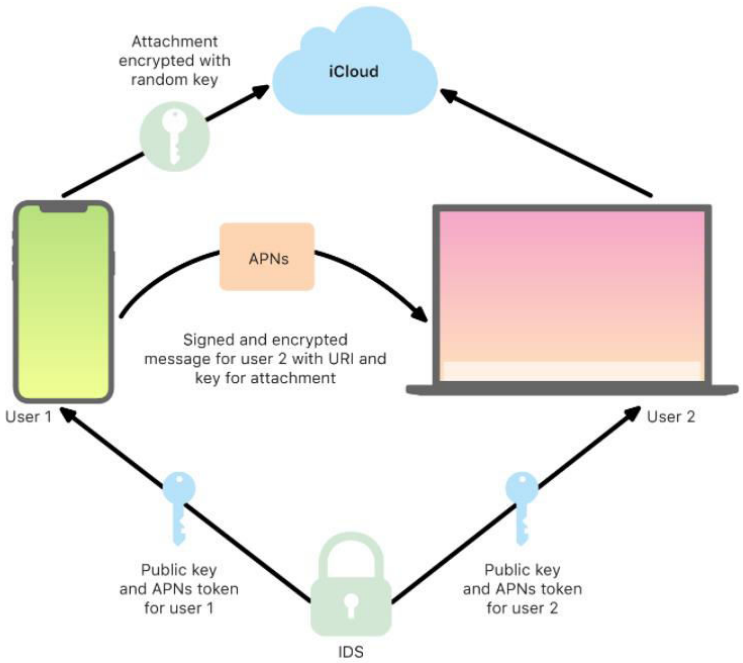


2.5: Инфраструктура Telegram апликације

2.5.4 iMessage

Апликација iMessage је у власништву компаније *Apple*, и на тржиште је постављена 2011. године. Служи за слање порука, слика, видео садржаја и докумената.

Као алгоритме шифровања апликација користи *RSA* и *AES*, а за хеширање *SHA-1*. Корисник започиње нову конверзацију уношењем имена или email адресе. Након тога она од *Apple Identity Service* добија јавне кључеве за енкрипцију и потписивање. Порука се за сваког примаоца шифрује засебно *AES* алгоритмом. Кључ за *AES* алгоритам се шифрује помоћу *RSA* јавног кључа, и након тога се шифрована порука и кључ заједно хешује помоћу *SHA-1* алгоритма. На крају се тај цео садржај за слање дигитално потписује. Мета подаци се не шифрују.



2.6: Инфраструктура iMessage апликације

2.5.5 Signal

Signal је апликација за размену порука и позива објављена 2014. године. Има аутоматску end-to-end енкрипцију. Као алгоритме криптовања користи *Double Ratchet* алгоритам и *Extended Triple Diffie-Hellman*. Користи *Curve25519, AES256* и *HMAC-SHA256*. Многи сматрају Signal сигурнијом од осталих апликација за размену порука и позива зато што су отишли корак даље, па поред криптовања порука и позива, они криптују и метаподатке, то јест податке везане за поруку, датум и време, број пошиљаоца и сл.

Апликације као што су *Facebook Messenger* и *WhatsApp* користе Сигнал протокол за сигурну размену.

У наставку је дата табела упоређења функционалности датих апликација[15]:

2.2: Подржане функционалности апликација за четовање

	Facebook	*iMessage*	*Telegram*	*WhatsApp*	*Signal*
Сакупља корисничке податке	Да	Да	Да	Да	Не
Подразумевана енкрипција	Не	Да	Не	Да	Да
Open-source	Не	Не	Не	Не	Да
Енкриптовани метаподаци	Не	Не	Не	Не	Да
Чува IP адресе	Да	Да	Да	Да	Не
Одбио је да обавјештајним агенцијама достави податке корисника	Не	Да	Да	Не	Да

2.6 Аутентикационе апликације

Поред криптовања самог садржаја порука, велику улогу има и аутентикација корисника, која представља обавезни део *end-to-end* енкрипције. Аутентикација служи да након идентификације и верификације одређеном кориснику да одређена права за коришћење неке апликације. За идентификацију се углавном користи неки вид креденцијала (корисничко име, емаил адреса и лозинка, итд...). Након идентификације и верификације, корисник добија неку врсту кључа или токена који се користи за приступ одређеном садржају апликације.

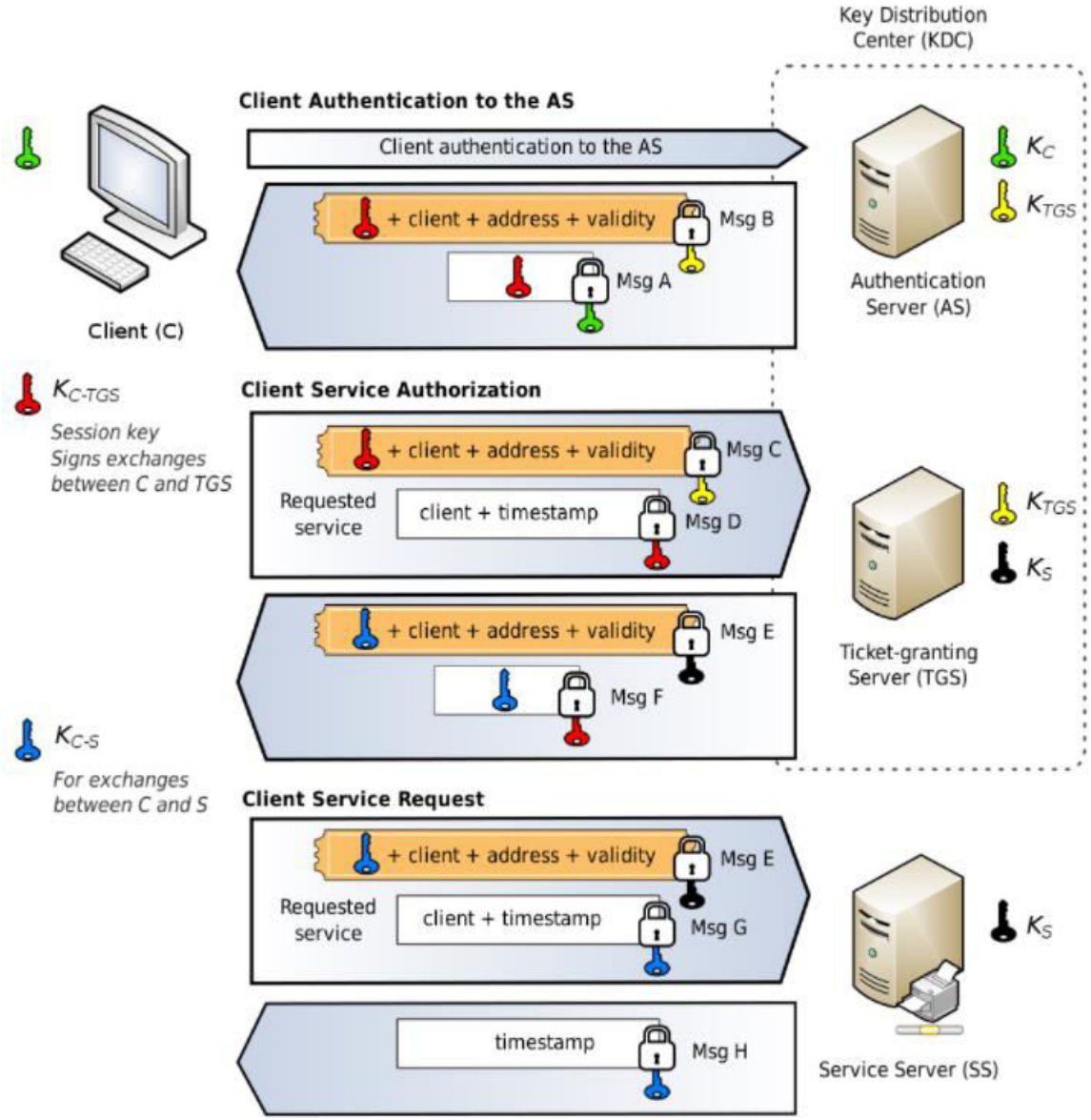
Аутентикациони протоколи (апликације) служе за утврђивање идентитета и размену кључева или токена. Оне морају обезбедити тајност и временску компоненту, то јест да се спрче replay напади. Један од најпознатијих протокола аутентикације је *Kerberos*[16].

2.6.1 Kerberos

Керберос је аутентикациони протокол који омогућава учесницима у комуникацији да утврде свој идентитет на небезбедној мрежи, и обезбеди приступ одређеним сервисима. Ради помоћу тикета и ослања се на симетричну криптографију.

Клијент се прво обраћа серверу за аутентикацију и након потврде идентитета добија тикет који користи за приступ центру за доделу тикета за приступ одређеним сервисима. Када жели да приступи одређеном сервису, клијент се обраћа центру за доделу тикета за тај сервис и касније са тим тикетом приступа одређеном сервису[17,18,19].

У наставку је дата детаљна шема Керберос система.



2.7: Керберос протокол

2.7 Предлог решења на глобалном нивоу

Проблем којим се бави овај рад је обезбедити сигурну размену порука између два корисника, и избећи нападе хакера који ће угрозити приватност корисника. То се постиже интеграцијом *end-to-end* енкрипције у апликацију која ће служити за размену порука. Конкретно у овом раду, апликација ће бити коришћена у претраживачу, то јест веб апликација.

Решење овог проблема је имплементација апликације која ће се састојати од клијената, који размењују поруке, и сервера, посредника који ће обезбедити сигурну размену, и обезбеђивања *end-to-end* енкрипције. То ће бити обезбеђено инфраструктуром која ће се бавити аутентикацијом корисника, и енкрипцијом конкретних порука. Замисао је да инфраструктура буде налик Керберос протоколу, који је за аутентикацију, али да се уз низ измена обезбеди и сигурна размена порука.

3. Опис система

У овом поглављу ће бити описана сама апликација која је и тема овог рада. Циљ је имплементација система са *end-to-end* енкрипцијом који ће по функционалности бити што ближи горе описаним системима. Биће описане коришћене технологије и инфраструктура по којој ће функционисати размена порука између клијената. Након тога следи конкретна имплементација апликације.

3.1 Коришћене технологије

Апликација ће бити развијена као веб апликација која ће се користити у интернет претраживачу. За имплементацију се користи стандардни скуп алата *MEAN Stack*. *MEAN Stack* даје могућност развоја *full stack* апликације (*frontend* и *backend*) и могућност складиштења података.

3.1.1 Angular

За *frontend* се користи *Angular*. То је *framework* за развој веб апликација предвођен *Angular* тимом у Гуглу. Базиран на *TypeScript* програмском језику. Прва верзија је објављена септембра 2016. године. Користи се код прављења *single-page* апликација. Главни елемент *Angular* језика су компоненте, од којих се касније праве блокови, и добија се комплетна страница. За коришћење *Angular* потребно је предзнање из *HTML*, *CSS* и *JavaScript*.

Angular је добар избор за развој јер сам по себи спречава *XSS* (*Cross-site Scripting*) нападе. *XSS* представља напад код кога нападач подмеће скрипту која може компромитовати корисника.

Пре коришћења мора се инсталирати *AngularCLI*. Инсталација се врши у терминалу помоћу команде: npm install -g @angular/cli.

Креирање новог пројекта се врши помоћу команде: ng new my-app. А покретање пројекта: ng serve –open.

Након ових команда добија се веб страница која се спремна за надограђивање.

Верзија *AngularCLI* која је коришћена у овом пројекту је 11.2.6, а верзија *TypeScript* 4.1.5

3.1.2 Node.js и Express

За *backend* се користе *Node.js* и *Express*. *Node.js* представља *JavaScript* окружење за развој серверске стране апликације. Архитектура *Node.js* је заснована на догађајима који омогућавају асинхроне улазе и излазе, што представља неопходну особину за развој апликације у реалном времену, какве су апликације за дописивање. Објављен је маја 2009. године. Прво је подржавао само *Linux* і *Mac OS X*, а касније и остале оперативне системе.

Инсталација *Node.js* се налази на званичном сајту: https://nodejs.org/en/download/.

Express је *framework* за *Node.js* под MIT лиценцом. Објављен је новембра 2010. године. Постао стандард за писање сервера, јер омогућава лакше и брже писање кода за сервер.

Инсталација *Express framework* се врши помоћу команде: npm install express.

Верзија *Node.js* која је коришћена у овом пројекту је 15.12.0, верзија *Express* 4.17.1.

3.1.3 MongoDB

Као база података се користи *MongoDB*. *MongoDB* је нерелациона база података за разлику од *SQL* базе. Објављена је фебруара 2009. године. Код базе је написан у *C++*, *JavaScript* и *Python*. Формат докумената је типа *JSON*. *MongoDB* заједно за претходно наведеним технологијама (*Angular*, *Node.js*, *Express*) представља један од најпопуларнијих скупа алата који је довољан за развој једне веб апликације од почетка до краја и који је коришћен за развој многих популарних веб апликација. Постоји онлајн верзија базе коју је могуће користити у претраживачу, и постоји десктоп апликација за коришћење ван интернета. У овом пројекту је коришћена онлајн верзија. Потребна је само регистрације преко сајта базе: www.mongodb.com и након тога се из апликације повезујемо преко линка који дат на самом сајту. Линк изгледа овако:
mongodb+srv://dusan:UZO9H2pl6CCH5I13@cluster0.l13rl.mongodb.net/MessagesDB?retryWrites=true&w=majority, где се у самом линку наводи корисник, и име базе података.

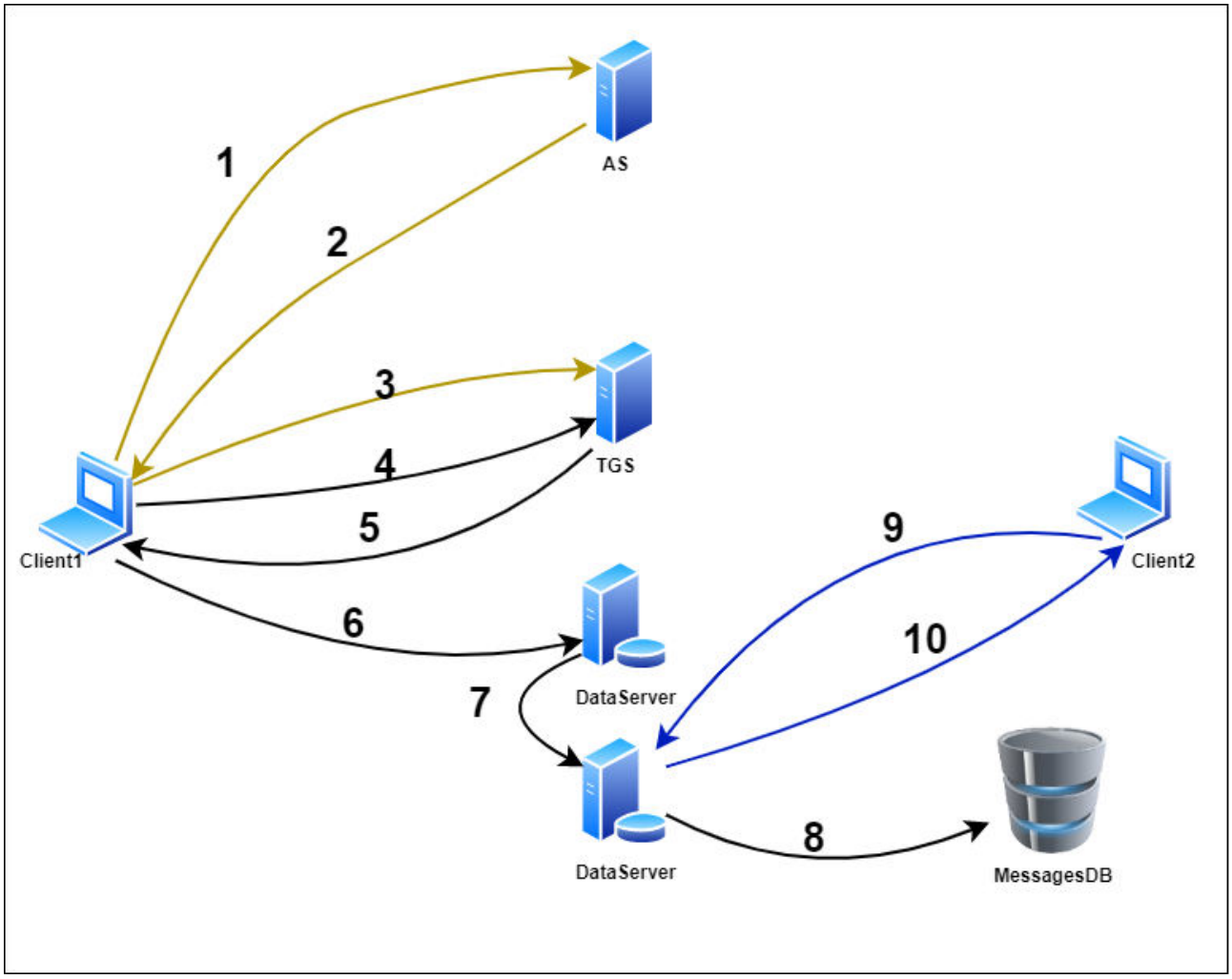
3.2 Инфраструктура апликације

Инфраструктура апликације мора да обезбеди сигурну размену порука и датотека између корисника апликације. Систем се састоји од клијената, који учествују у комуникацији, сервера који обезбеђују сигурну размену, и базе података за чување података регистрованих корисника, и чување порука.

У овом поглављу ће бити представљена инфраструктура целокупне апликације. Како тече комуникација од тренутка пријављивања неког клијента, па све до размене поруке са неким другим клијентом, криптовања те поруке и смештања у базу података.

Систем функционише по принципу Керберос апликације, уз низ измена које ће омогућити да постане систем са *end-to-end* енкрипцијом, што је и циљ пројекта. У систему има неограничен број клијената, то су регистровани корисници који користе апликацију за међусобну комуникацију. Постоји *Autehtication Server* (AS) коме се обраћају корисници приликом логовања, и који идентификоване кориснике усмерава даље. Следећи је *Ticket Granting Server* (*TGS*). Он служи за размену кључева између клијената. И на крају постоје Дата сервери који служе за размену порука. У систему има и две базе, у једној се чувају корисници, а у другој поруке.

У наставку је комплетна инфраструктура апликације и детаљно објашњење сваког корака у комуникацији.



3.1: Инфраструктура за сигурну размену порука

Објашњене сваког корака у комуникацији:

1. *Client1* се улогује на апликацију и генерише свој приватни и јавни кључ за *RSA* алгоритам. Шаље поруку *AuthenticationServer* (AS) у којој шаље свој јавни кључ.
2. *AS* генерише кључ који ће *Client1* користити за приступ *TicketGrantingServer* (*TGS*). То је *RSA* јавни кључ. Тим кључем ће *Client1* криптовати поруку за *TGS*. *AS* сада мора да пошаље приватни кључ *TGS* да би он могао да декриптује ту поруку. *AS* и *TGS* имају заједнички тајни кључ за *AES* алгоритам, па ће *AS RSA* приватни кључ криптовати тим тајним кључем. На крају, *AS* криптује поруку која се састоји из 2 блока (први је *RSA* јавни кључ за приступ TGS, а други *RSA* приватни кључ криптован *AES* алгоритмом) јавним кључем Клијента1 и шаље Клијенту1.
3. *Client1* генерише кључ који служи за криптовање порука. Он је примио поруку од *AS* и декриптује је својим приватним кључем (јер је криптована јавним кључем Клијента1). Из те поруке узима блок који је за *TGS* (блок који је *AS* криптовао *AES* алгоритмом, и *Client1* нема приступ том блоку, само *AS* и *TGS*), и кључ који је генерисао криптује јавним кључем који је добио од *AS*. Поруку шаље *TGS* серверу.

Након пријема поруке *TGS* декриптује блок који је криптован *AES* алгоритмом, и добија кључ за декриптовање поруке од *Client1*. Након што декриптује поруку од *Client1* добија кључ који ће служити за криптовање порука.
На овај начин сви клијенти приликом логовања шаљу своје кључеве за криптовање порука, који се чувају на *TGS* серверу.

4. Сада *Client1* хоће да комуницира са *Client2*. Шаље поруку *TGS* серверу у којој шаље свој јавни кључ.
5. *TGS* након што прими поруку од *Client1*, и види да он хоће комуникацију са Клијентом2, узима кључ за криптовање порука *Client2* и криптује га јавним кључем *Client1*. Ту поруку шаље *Client1*.
6. *Client1* декриптује поруку од *TGS* својим приватним кључем и тако добије кључ за криптовање *Client2*. Поруку криптује тим кључем *AES* алгоритмом и шаље серверу *DataServer*.
7. *DataServer* прима поруку и нема информације за кога је та порука, он само зна од кога је. Он ту поруку шаље другом серверу *DataServer*.
8. Други *DataServer* нема информацију од кога је, али зна за кога је та порука, па смешта поруку у базу.
9. *Client2* тражи своје поруке из базе поруком *DataServer*.
10. DataServer шаље поруке *Client1*.

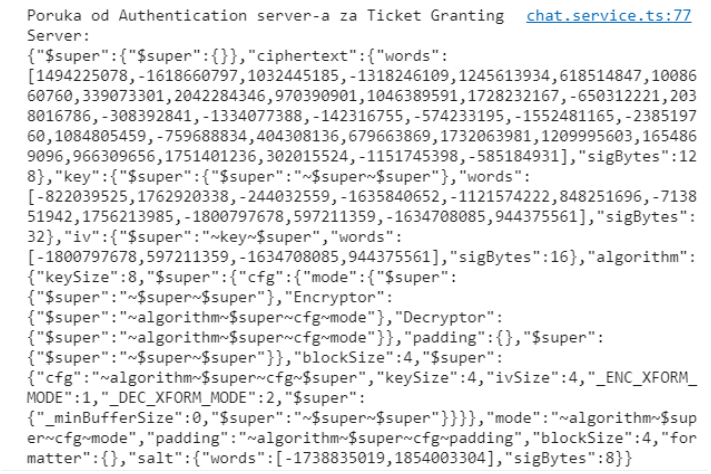
Ако је *Client1* онлајн у тренутну слања поруке, тј. кораку 6, он од *DataServer* директно добија поруку.

Кораком 7, и додавањем још једног сервера *DataServer* смо успорили проток информација, али смо добили на сигурности сакривши један метаподатак, а то је да први *DataServer* зна од кога је порука стигла, али не зна за кога, док други *DataServer* не зна од кога је порука, али зна за кога је, што крије информацију која би потенцијалном нападачу била од користи.

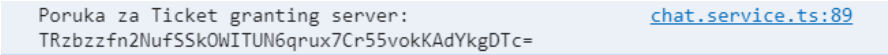
У наставку ће бити приказани примери конкретних порука између клијената и сервера:

```
Poruka za Authentication server: undefined-----BEGIN  chat.service.ts:72
RSA PRIVATE KEY-----
MIIFwwIBAAKCAUEAjHycwl/JxkKBAMAXlyp6cZv8pWDIyN0AA43kzFz3+tcDU3FZzhlEPzPq
vhBqKq/fpnOMzUu7oXVj6Zkeu0K6q2N5hjb5ocKYCoTwzDVlf31kxNu4GfEyu7nETuKZVFD6
ID1aeqZmLmjNd1Z4JcfJcblF/uPJEMW+m+CqHQ/AyzgTKKMWqXR8UNNGnLfLnSNPSqGjrr8k
/UqGQIyqA2ejC7JCJxASUchquFLoBW4gSXkJ9URv7KdnYff08qSCbAr2KIH5/o6lKlIziIl8
ABwHC3ID56pf4y1QJQptuJ9Xyn5w9x1qmuzBRo35noKdehvNu1kYLcDvh+AB2dhGC1D5y2tz
tpoIJ10UQQjiyK61l28nxvw6bxcsrScYuIKyGF2EGfJ1pfmo/FQWTWHADtKZurdqQhPLs8PV
S8dcceNbDhECAwEAAQKCAUBX5+Fu/uES3ao6/ffE7x+O1zxPfzAj/6Vyk+cGuWWx37iFDfhT
rhu2MJZCN6M28rXPQZ5pF2pgrUPHd7hzFF4ZwDTadqvdPBZG/sht/ztzetMDGzkngnVsYeEk
O0YWLFVWtw1Wkah+gbWPZtvnSAwUzQmvZHj7tcB/aSUI6kKI5p2WX/ZeMjVXndwhVCBkQyPq
Xxo8zf4nP+qhmWMInmC2kmuj2mcv6R+rEmNKamWEeApbDhhY25Thh1JV+07V3Cp6Tcky9X7W
X+mm9O6E90JirIzRXTkN+5KTKMEjla7ai0s4gwDLFxLnzArSLlcDkMhtoNH81YcBrMRw+grJ
t2DBRWaAhLunBEFHNk4sPSgBu+4mzF+EkUDt641Y3G+8321N7Ix/+3EejQ7GYZNzqpgEM0xL
ltjXEo+UoQK64OgNAQKBoQDRJ4zVeXuKRScySVs/HJLapFEkFABIqH/dcvVNJDthfZR1R1qT
GjRXuLemPtP6ILtw9uPsA372a36z4tyrOkTDp/FnFDUd3ZGc4EOPkRc7LNSn5wCzyG0DxYho
Fllvim0+OWxW7NfEheC7BS7AZwaQsu/ko39iY07j5pmBq/kaKFWkOl2nkTUG6tFvXaouY7hr
fjEs4kkB2N+1AlWKFw5VAoGhAKvzzwd857hn9n7rZRQ/DQhhU6r5sen/IEEjatPkU1pDACvW
5LcDC8K+sYiC1DMInDiwVhWX854RmuFz6LItZwCRQ4kHGj9exQeObd62+XPH9x13tT6rtFtZ
Msib8crLzjT4VJRvcFus9/qhJ/5cjnJ70KJJnv73Hqi7O6Jhjom8xRFwCtfUS47PRU8Dq97E
8vGf9lBCubGp37uB/Vy+RM0CgaEAsfzBOa1e94Tv3ThV4ThUtKGet2AZ+bsCzRYWMCaELeur
Wb52Erb7EieMiXnv9GzEd1Z0EZXMBrHlqfmGJmS4vfP+V+T3asnyL5kdhWBIb8Av3A6WJ2iJ
dOs1sLZMN8KKQqilTLfSylfPrfe0GGHUUXEk+uIz9itiqEjGG3aYzqzCbLaKmp3UjWVpAyen
OkD8YOWLvZF35Y2Emko5NwF13QKBoAD/POgSNVXAZJF/0+o2I2E1D0Mih4GP1SQLjIcF/+W6
FborWNAm2F6ctHLSL2DMX/uEKjud0e/MslU0UtkVok9CiF23CNPtPL86eDCeZvvg4PHuy8I9
RKJy9fRr0w8V93AfMH5b1w7xBc8SOCUlbD4FscP/IUjhLfBUhZf/cNAjpBDh0W5G8Nk3InO+
1bRI+A+mmcsFWAZLsdxoeIudHtECgaBoN5BmjWlEg+lKSdo/BNXN1t3FFqeSXGYu5KyBaPdH
dia6NJE8xRqDkvKCcVxU+hxrLoioQFK+kTpoA+M5aZOSZ2ewAu1z3zx3sKsfLxA8Rk+kIp+i
SSDwuBkMNbqIi27fJZpl9wDx0KXbGWNcDOXZyhzHF37i5Bd0/RVOldNHhEZwE0p+Vdg4fqvc
NfP0K9wvN11jTGVAYxbkkBUiqGdd
-----END RSA PRIVATE KEY-----
```

3.2: Порука Клијента ка АС (корак 1)



3.3: Приватни кључ криптоваан AEC алгоритмом



3.4: Порука Клијента за ТГС (корак 3)


```
_id: ObjectId("60b2aeb287a8a52ecc215e26")
uid: "test1c7F021K+UK20Ikw+asKTMWt1XZCcWPE/hUbbKKnp1OcVlpQfUs/XDMHosfhg+uN06..."
fromUser: "test1"
toUser: "c7F021K+UK20Ikw+asKTMWt1XZCcWPE/hUbbKKnp1OcVlpQfUs/XDMHosfhg+uN069Zamr..."
datetime: 2021-05-29T21:14:26.246+00:00
message: "bOqbWC0/uU1cyi2qBZyzA9einb3GZbKpMtaP3K+bjJG5pb2VNFDlfeA58Q4aA5oC5DuRx8..."
__v: 0
```


3.5: Порука коју Клијент1 шаље првом ДатаСерверу


```
_id: ObjectId("60b2aeb287a8a52ecc215e26")
uid: "dJuiPBiDShxFvvJu7K0hdCxhYxFIQeOqbga3Pov6TNT4R+88j3i3bP88NHFTy9IcZ/YQCJ..."
fromUser: "dJuiPBiDShxFvvJu7K0hdCxhYxFIQeOqbga3Pov6TNT4R+88j3i3bP88NHFTy9IcZ/YQCJ..."
toUser: "test2"
datetime: 2021-05-29T21:14:26.246+00:00
message: "bOqbWC0/uU1cyi2qBZyzA9einb3GZbKpMtaP3K+bjJG5pb2VNFDlfeA58Q4aA5oC5DuRx8..."
__v: 0
```


3.6: Порука коју први ДатаСервер шаље другом ДатаСерверу


```
_id: ObjectId("60b2aeb287a8a52ecc215e26")
uid: "R1611L3cnvuP6VHhwxkPDfXnNJq0q4GAk5D7pOiHgZCtMwus6Y0kvoLs+7N1Ns9NMD5cpX..."
fromUser: "R1611L3cnvuP6VHhwxkPDfXnNJq0q4GAk5D7pOiHgZCtMwus6Y0kvoLs+7N1Ns9NMD5cpX..."
toUser: "c7F021K+UK20Ikw+asKTMWt1XZCcWPE/hUbbKKnp1OcVlpQfUs/XDMHosfhg+uN069Zamr..."
datetime: 2021-05-29T21:14:26.246+00:00
message: "bOqbWC0/uU1cyi2qBZyzA9einb3GZbKpMtaP3K+bjJG5pb2VNFDlfeA58Q4aA5oC5DuRx8..."
__v: 0
```


3.7: Изглед поруке смештене у бази


```
_id: ObjectId("60b2aeb287a8a52ecc215e26")
uid: "test1test2"
fromUser: "test1"
toUser: "test2"
datetime: 2021-05-29T21:14:26.246+00:00
message: "Zdravo test2 :)"
__v: 0
```


3.8: Изглед некриптоване поруке

3.3 Потенцијални проблеми у раду апликације

У претходном поглављу (3.1) приказан је рад апликације у идеалним условима, то јест када се све изврши како је и очекивано, међутим у доста случајева се дешава да дође до непланираних проблема приликом рада апликације.

Главни проблем представља отказ неког од сервера који посредује између клијената. Постоји четири сервера у апликацији. Први је *LoginServer*, па ако дође до његовог отказа, корисник неће моћи да се улогује на апликацију, и добиће одговарајуће обавештење о томе. Овај проблем се може решити додавањем резервног сервера LoginServer. Такође, улогу сервера за логовање може да преузме и сервер за аутентикацију (AutehticationServer). Ово је добро решење јер се не троше средства за нови сервер. Ако се успешно улогује, прелази се на инфраструктуру која обезбеђује сигурну размену (поглавље 3.1).

После логовања, клијент шаље поруку серверу за аутентикацију (*AS*). Његово отказивање би довело до тога да клијент не може да потврди свој идентитет, и не може да добије кључ за приступ серверу *TGS*. Клијент добија одговарајућу поруку, и нема могућност приступа апликацији док се не омогући рад сервера за аутентикацију. Додавање помоћног сервера, који би се користио само у случају да откаже сервер за аутентикацију би решило проблем. Такође, његову улогу би могао да замени LoginServer, коме би се слале поруке приликом логовања. Следећи у низу је сервер за размену кључева (*TGS*).

Ако дође до његовог отказивања, клијент неће моћи да свој кључ проследи серверу, и не може да добије кључ других клијената за шифровање порука. Такође добија поруку, и не може да настави са радом док се не омогући рад сервера. Његову улогу би могао да замени још један идентични сервер, с тим што би морало да се чува и копија кључева на тим серверима, јер TGS чува све кључеве клијената који служе за шифровање порука.

У случају отказа једног од *DataServer*, клијент добија обавештење након слања поруке, пошто прослеђивање поруке неће бити могуће, та порука се одбацује и клијент добија обавештење да опет пошаље поруку. Пошто има два DataServera, један би могао да замени други у случају отказа.

Проблем који настаје када један сервер преузима улогу другог у свим горе наведеним случајевима је ако дође до отказа оба сервера, тако се намеће као решење да сваки сервер има резервни сервер са истом улогом, али то решење је скупље, у смислу да се троши више ресурса.

Још један могућ проблем је отказивање базе података. Ако дође до тога, порука не може бити сачувана, па самим тим није могућа ни комуникација. Отказивањем базе, сервер нема могућност конектовања на исту, па добија поруку. Након пријема поруке, обавештава се клијент да покуша поновно слање.

3.4 Имплементација апликације и корисничко упутство

У овом поглављу ће бити представљени детаљи имплементације саме апликације, коришћене библиотеке, и начин на који је нешто имплементирано. Такође, биће представљене и функционалности самог система.

3.4.1 Клијентска страна апликације

За имплементацију клијентске стране коришћен је *Angular*. Клијентска страна се састоји од више компоненти које заједно чине *frontend* апликације. Свака компонента се састоји од .html, .css, .ts фајлова. За изглед апликације су задужени .html и .css фајлови. У *Typescript* фајлу се налазе све функције које престављају везу између *frontend* и *backend* делова. Велику улогу имају и сервиси. Њихово коришћење је ствар имплементације самог програмера, али рад са њима у великој мери олакшава имплементацију. Сервис представља обичну класу која има *@Injectable()* декоратор. У конструктору компоненте се мора нагласити да зависи од сервиса да би било могуће коришћење његових функционалности. Сервис омогућава чување неких битних података и након брисања компоненте, читање података при креирању компоненте, и пошто је могуће сервис уградити у више компоненти, смањује потребу за имплементацијом истих функција више пута.

У сервису су имплементиране функције за шифровање на клијентској страни. За шифровање је искоришћена библиотека *cypto-browserify*. Она пружа функције за генерисање кључева за *RSA* алгоритам, као и функције за шифровање помоћу *RSA* и *AES* алгоритма.

За генерисање кључева се користи функција *keypair(512)*, чији је аргумент дужина кључа, а повратна вредност приватни и јавни кључ.

У наставку је дата функција која представља слање иницијалних порука *AS* и *TGS* серверу. У поруци за *AS* се шаље корисничко име и јавни кључ клијента. Захтев се шаље као *http* захтев. Након одговора у којем клијент декриптовањем поруке добија кључ за приступ *TGS* серверу. Помоћу функције *crypto.publicEncrypt*() клијент шифрује свој кључ за шифрирање порука и заједно са блоком који иде од *AS* до *TGS* шаље *TGS* серверу. Након тога добија одговор да ли је порука успешно послата или не.,

```
sendLoginMessageToAS(username: String, password : String){
    const messageAS : MessageToAS = {username: username, publicKey: this.rsakey.public};
    this.http.post<{publicSessionKey: string, messageForTGS:
string}>("http://localhost:8000/loginMessageToAS", messageAS).subscribe(response => {
      let buffer = Buffer.from(response.publicSessionKey, 'base64');
      this.sessionKeyTGS = crypto.privateDecrypt(this.rsakey.private, buffer1);
      this.messageFromAStoTGS = response.messageForTGS;
      let buffer1 = new Buffer(this.username.concat(this.password.toString()));
               let encrypted = crypto.publicEncrypt(this.sessionKeyTGS, buffer1);
               encrypted = encrypted.toString('base64');
               const messageTGS : MessageToTGS = {key: encrypted as string, messageFromAStoTGS:
this.messageFromAStoTGS};
               this.http.post<{message: string}>("http://localhost:8080/messageToTGS",
messageTGS).subscribe(response => {
                    console.log(response.message);
               });
    });
```

3.9: Исечак кода који приказује иницијалне поруке за АС и ТГС сервер

За изглед апликације је коришћена библиотека *Angular Material*. *Angular Material* помаже у прављењу атрактивних и функционалних веб страна и апликација модерног изгледа[20].

3.4.2 Регистрација и логовање

На самом почетку, уласком на сајт апликације приказује се прозор за логовање. Нерегистривани корисници имају могућност регистровања на систем, и не могу користити апликацију без регистрације.

Приликом регистрације корисник уноси: име, презиме, корисничко име (мора бити јединствено на нивоу система), лозинку и потврду лозинке. Сва поља су обавезна. Подаци регистрованог корисника се чувају у бази података. Шифра корисника се криптује и као таква чува у бази. За криптовање шифре се користи библиотека *"bcrypt"*. Конкретно, из библиотеке је искоришћена функција за хеширање података.

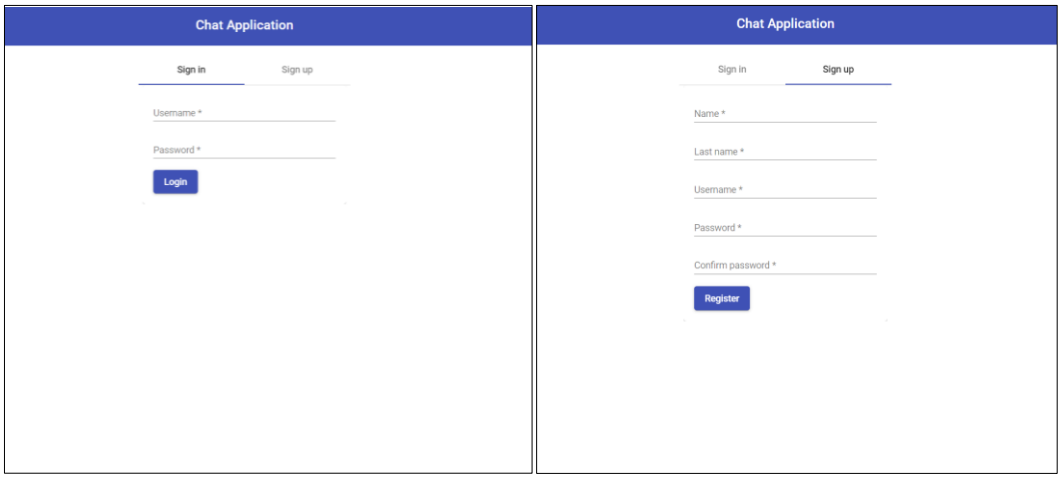
Хеширање представља алгоритам којим се дати подаци преслакиавју у низ података фиксне величине. То је *one-way* функција, што знаци да је готово немогуће добити почетни облик података. Једини начин је *brute-force* напад на податке, и провера подударања података[21].

Приложени део кода представља захтев за регистрацијом корисника. Прво се функцијом *findOne()* проверава да ли корисник са истим корисничким именом постоји и у случају да постоји враћа се грешка. У супротном се чувају подаци у бази, с тим сто је пре тога шифра криптована функцијом *hash* библиотеке *bcrypt*.

```
appLogin.post('/register', (req, res, next) => {
  bcrypt.hash(req.body.password, 10).then( hash => {
    console.log(req.body.password);
    const user =  new User({
      name: req.body.name,
      lastName: req.body.lastName,
      username: req.body.username,
      password: hash
    })

    User.findOne({ username:req.body.username }).then(user1 => {
      if(user1){
        return res.status(401).json({
          message: "User Already Exist!"
        })
      }

      user.save().then(result => {
        if(!result){
          return res.status(500).json({
            message: "Error Creating User!"
          })
        }
        res.status(201).json({
          message: "User created!",
          result: result
        });
      })
    })
    .catch(err => {
      console.log(err);
      res.status(500).json({
        error: err
      });
    });
  });
});
```

3.10: Исечак кода који приказује захтев за регистрацијом корисника

Приликом логовања на систем од корисника се тражи корисничко име и шифра. Када се унесу тражени подаци, ако постоји корисник са датим корисничким именом у бази, проверава се шифра. Шифра се проверава помоћу функције compare из *bcrypt* библиотеке. Та функција проверава хеширану шифру из базе. Ако се шифре подударају, омогућује се наставак коришћења апликације. Том приликом добија токен за ауторизацију (поглавље 3.2.4).



3.11: Екран за логовање (лево) и регистрацију (десно)


```
_id: ObjectId("6082b84b93c9bf14fc35291e")
name: "test1"
lastName: "test1"
username: "test1"
password: "$2b$10$UuUesydOokkkonnDAKwyOOIy8XkujqbU3nAYe1awpNK10JLXT50fG"
__v: 0
```


3.12: Изглед сачуваних података корисника у бази

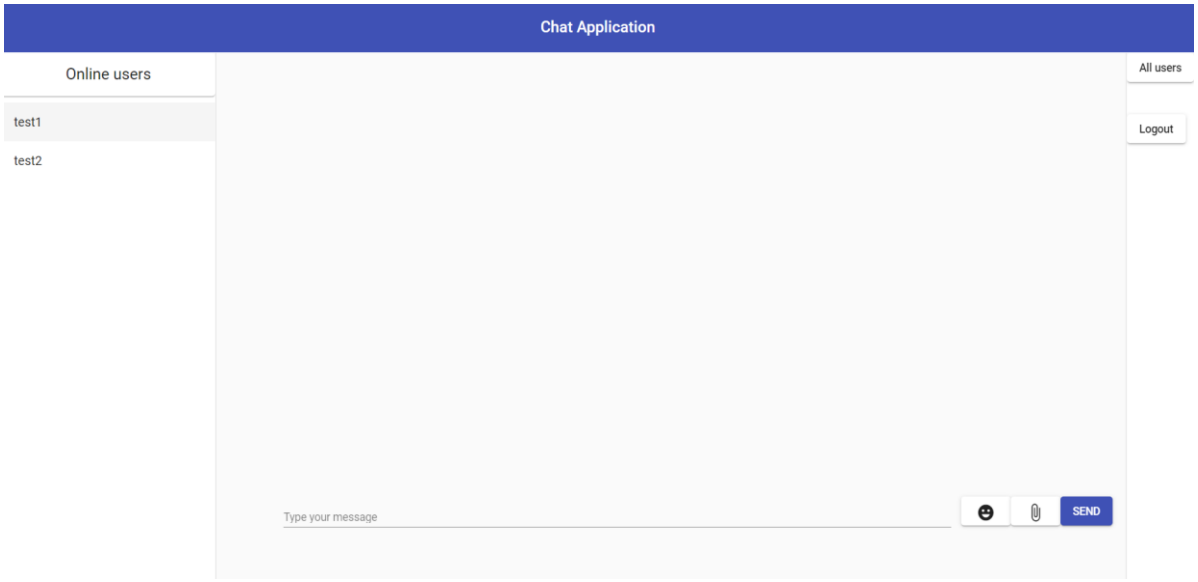
3.4.3 Почетни екран апликације

Након логовања кориснику се приказује почетни екран аликације. У горњем десном углу су приказани онлајн корисници (корисници који су тренутно на мрежи). Избором одређеног корисника отвара се историја порука са тим корисником.

На десној страни се налазе два дугмета. Прво је за преглед свих корисника, чијим избором се отвара падајући мени са свим корисницима, а друго је дугме за излоговње са система.

На средини екрана се налази поље за унос поруке. Са десне стране поља се налазе три дугмета. Прво служи за додавање емотикона у поруку. Друго је за избор фајла за слање (тренутно је могуће слати само .txt фајлове), и треће је за слање поруке.

Имплементација *frontend* дела почетног екрана је извршена помоћу *HTML* и *CSS*. Састоји се из више компоненти које заједно дају целину. Функционалности свих дугмета су преко функција из клијентског дела повезане са backend делом.



3.13: Почетни екран апликације

3.4.4 Серверска страна апликације

Имплементација сервера у овој апликацији је извршена помоћу Node.js и *Express framework*. Сервери представљају посреднике између клијената и они су ти који обезбеђују сигуран пренос података. Компромитовањем само једног сервера нарушава се приватност целе конверзације, па цела имплементација мора бити добро испланирана.

Сервери су задужени за скоро све што корисник "не види", то јест оно што се дешава у позадини. У наредним поглављима ће бити објашњена појединачна улога и имплементација сервера.

3.4.5 Ауторизација корисника

На самом почетку, приликом логовања, потребно је кориснику доделити одређена права, то јест дозволити му да приступа одређеним рутама, сервисима и ресурсима у некој апликацији. Постоји више начина који то омогућавају, у овој апликацији је имплементиран *JSONWebToken*[22,23].

JSONWebToken (*JWT*) је стандард који дефинише компактан и независан начин за безбедан пренос информација између страна као *JSON* објекат. Овим информацијама се може веровати јер су дигитално потписане. Токени се могу потписати помоћу тајног (са *HMAC* алгоритмом) или пара јавних/приватних кључева користећи *RSA* или *ECDSA* алгоритам.

Токен се састоји из три дела: хедер, тело и потпис токена. Хедер садржи два дела: тип токена, и тип алгоритма за потписивање. У телу токена се обично преносе информације о кориснику, и о његовим правима. Потпис служи да обезбеди да порука није мењана на путу до одредишта или да потврди идентитет пошиљаоца. За потписивање се користи алгоритам наведе у заглављу.

У овој апликацији за ауторизацију је задужен *LoginServer*. Његова улога је да поред регистрације и логовања обезбеди и токен улогованом кориснику. Корисник приликом

логовања, када се потврди корисничко име и шифра, добија јединствени токен. Имплементација је извршена помоћу *jsonwebtoken* библиотеке. Токен добијамо помоћу функције *sign*. Тај токен се шаље клијенту и он користи тај токен за даљи приступ апликацији. Токен се шаље у хедеру http захтева. Веома корисна функционалност је та да приликом креирања токена можемо навести време за које ће тај токен истећи, што омогућава да се корисник аутоматски излогује.

У наставку је део кода функције која служи за логовање и додељивање токена одређеном клијенту. На почетку се проверава да ли је унета исправна лозинка корисника. У случају да није враћа се порука и омогућава поновни покушај. У супротном корисник добија јединствени токен помоћу функције *jwt.sign()*. Токен траје 1 сат, а након тога се мора ресетовати јер истиче сесија.

```
return bcrypt.compare(req.body.password, user.password).then( result => {
        if(!result){
          return res.status(401).json({
            message: "Authentication failed! Inccorect password!"
          })
        }
        const token = jwt.sign(
          { username: fetchedUser.username, userId: fetchedUser._id },
          "secret_this_should_be_longer",
          { expiresIn: "1h" }
        );
        res.status(200).json({
          token: token,
          expiresIn: 3600,
          userId: fetchedUser._id
        });
      })
```

3.14: Исечак кода који приказује функцију за логовање корисника

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

3.15: JSONWebToken – хедер, тело и потпис

3.4.6 Размена порука

Најбитнија функционалност савремених апликација за дописивање је размена порука у реалном времену. У овој апликацији смо то постигли коришћењем библиотеке *Socket.io*.

Socket.io омогућава двосмерну комуникацију између клијента и сервера. Та комуникација је омогућена ако клијент има *Socket.io* у претраживачу, а сервер такође има интегрисан *Socket.io*. Подаци се могу слати у више облика, али је *JSON* најједноставнији. Мана овог решења је што ће апликација бити лимитирана на одређени број корисника. *Socket.io* у великој мери оптерећује сервер, па када сервер достигне свој максимум, улога сервера се мора поделити на два дела. То није могуће решити само додавањем једног сервера, јер Socket.io зна само за клијенте повезане на тај сервер, не и на други. Творци *Socket.io* су решили овај проблем стварањем Socket.io-adapter, који служи за дељење информација између више сервера.

У овој апликацији интеграција *Socket.io* је извршена на оба *DataServer* сервера и клијентима. Приликом логовања на апликацију, отвара се конекција између клијента и сервера, и размена порука је могућа. Наравно, пре размене су извршени сви кораци како би обезбедили сигурну размену.

Ово је функција за слање поруке помоћу сокета, која је претходно шифрована:

```
this.socket.emit('send-chat-message', message, username, fromUsername);
```

Ово је функција за пријем поруке на серверској страни:

```
this.socket.on('send-chat-message', function data(message, username, fromUsername)){...}
```

3.4.7 Шифровање порука

У овом поглављу ће бити описани детаљи имплементације везани за конкретно шифровање порука на серверској страни. Од алгоритама за шифровање су коришћени *AES* и *RSA*. Библиотеке које су коришћене су *cryptoJS* и *crypto-browserify*. У наставку је дат део кода који се односи на криптовање и декриптовање садржаја на серверској страни.

```
cryptoJS.AES.encrypt(rsakeys.public, 'secret key for AS and TGS')

cryptoJS.AES.decrypt(JSON.parse(req.body.messageFromAStoTGS), 'secret key for AS and TGS');
```

Прва функција представља функцију библиотеке *cryptoJS* помоћу које криптујемо садржај, а помоћу друге декриптујемо.

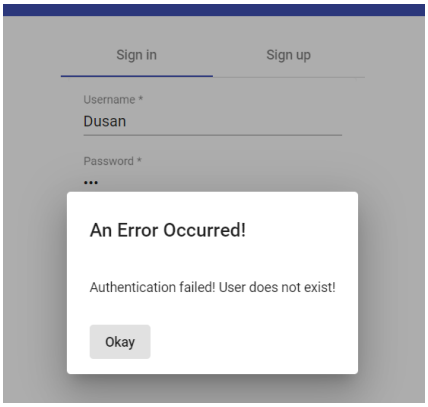
```
crypto.publicEncrypt(this.publicKey, buffer);

crypto.privateDencrypt(this.privateKey, buffer);
```

Прва функција представља функцију за криптовање помоћу *RSA* алгоритма из библиотеке *crypto-browserify*, а друга за декриптовање садржаја. Кључеви су генерисани помоћу функције *keypair()*.

Ове функције су примењиване за све врсте криптовања у овом пројекту, и за криптовање порука између самих сервера, и за криптовање порука које шаљу клијенти.

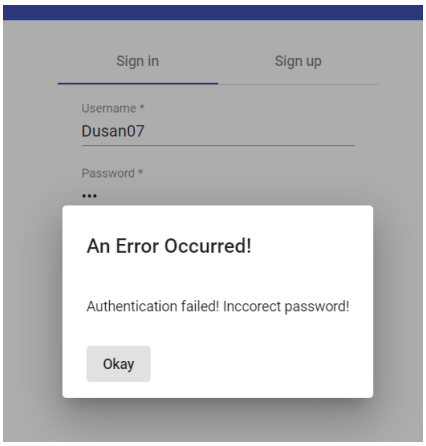
3.5 Понашање система у различитим ситуацијама

Приликом логовања клијената ако се унесе корисничко име које се не налази у бази података систем ће избацити следеће обавештење:



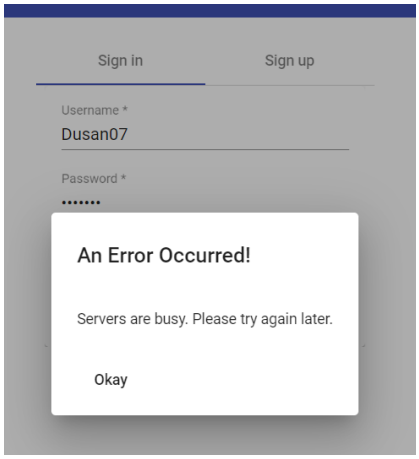
3.16: Грешка приликом тражења корисника

Уколико клијент унесе тачно корисничко име, али погрешну лозинку систем приказује обавештење:



3.17: Грешка приликом уноса погрешне шифре

Ако неки од сервера (*AS, TGS*) тренутно не ради, или не може да прихвати захтев корисника, добија се следеће обавештење:



3.18: Грешка на неком од сервера

Понашање апликације варира од ситуације до ситуације и апликација покрива основне сценарије који се могу догодити. Тема апликације је таква да је могуће константно унапређивање и додавање нових функционалности.

Једна од највећих мана је ограничен број корисника због коришћења *Socket.io* библиотеке. Овакве апликације се углавном конструишу за велики број корисника, па је то једна од мана коју би прво требало решити у даљем унапређивању апликације.

Добра страна је што је избором технологија у старту спречен *XSS* (*Cross Site Scripting*) напад, али нису покривени сви напади. Додавањем дигиталних потписа порукама између сервера би се отпорност апликације доста побољшала.

Чување лозинке корисника у шифрованом формату је неизбежно за овакав тип апликације, где је приватност најбитнија па је то имплементирано, али би требало обезбедити и криптовање осталих информација о кориснику.

Поред порука, које се у бази чувају у шифрованом формату, ова апликација пружа чување и података о пошиљаоцу и примаоцу у шифрованом формату, што у великој мери смањује могућност потенцијалном нападачу да угрози приватност корисника.

4. Закључак

У овом раду смо се бавили сигурном разменом података на интернету, што у данашњем времену представља свакодневницу, јер све више и више људи користи интернет, па и је и сигурност један он најбитнијих фактора. Сигурну размену смо имплементирали у виду веб апликације – *chat* апликација у реалном времену. Скуп технологија коришћен за развој апликације је *MEAN Stack*.

Имплементирали смо инфраструктуру која представља *end-to-end* енкрипцију, и представља основу модерних система комуникације. Инфраструктура обухвата све од регистровања корисника, до крајње размене порука. За аутентикацију корисника смо користили *JSONWebToken*, који је регистрованим клијентима омогућио приступ апликацији. Такође, *JSONWebToken* нам омогућује да видимо како функционише потписивање садржаја на интернету. Размену порука у реалном времену смо имплементирали помоћу библиотеке *Socket.io*. У овом раду смо показали и како функционише сигурна размена кључева између клијената и сервера. За симетричну енкрипцију смо користили *AES* алгоритам, а за асиметричну *RSA*.

За изглед апликације је коришћена библиотека *Angular Material* која даје савремен изглед апликацији и коју користи много популарних интернет сервиса данас. Од функционалности имамо слање текстуалних порука, слање емотикона, и текстуалних фајлова онлајн и офлајн корисницима апликације.

Апликација подржава бројна проширења која се могу једноставно имплементирати, као што су измена личних информација, блокирање/одблокирање других корисника, брисање порука, архивирање конверзација, слање слика и видео клипова, итд. Ове функционалности нису имплементиране у овом раду јер је акценат био на инфраструктури за сигурну размену података.

Битне функционалности апликације које треба додати у будућности су криптовање свих метаподатака, чиме би се смањиле шансе нападачу да компромитује систем и наруши приватност корисника. Следећа битна функционалност је интеграција *Socket.io-adapter*. Ово би подржало "неограничен" број корисника у апликацији. За корисничке функционалности битно је додати групни *chat*, да би могло више корисника да учествује у једној конверзацији. Аудио и видео позиви су у данашњим апликацијама неизбежни, па је битно и њих интегрисати у апликацију.

Литература

[1] "Šta je end-to-end šifrovanje?", https://bs.eyewated.com/sta-je-end-to-end-sifrovanje/, приступано август 2021.

[2] "Instant messaging", https://en.wikipedia.org/wiki/Instant_messaging, приступано август 2021.

[3] "Secure transmission", https://en.wikipedia.org/wiki/Secure_transmission, приступано август 2021.

[4] Daniel Kim, "PlayReady DRM – 5 Things to Know About DRM Technology " . PlayReady DRM - 5 Things to Know About DRM Technology | PallyCon, приступано август 2021.

[5] "What is MEAN Stack?". What is the MEAN Stack? Introduction & Examples | MongoDB, приступано август 2021.

[6] "Kriptografija", Kriptografija - Wikipedia, приступано август 2021.

[7] "Симетрично шифорвање", Microsoft PowerPoint - 02 Simetricno sifrovanje (bg.ac.rs), приступано август 2021.

[8] Brett Daniel, "Symmetric vs. Asymmetric Encryption: What's the Difference?", https://www.trentonsystems.com/blog/symmetric-vs-asymmetric-encryption, приступано август 2021.

[9] "Advanced Encryption Standard", Advanced Encryption Standard - Wikipedia, приступано август 2021.

[10] Data Encryption Standard", Data Encryption Standard - Wikipedia, приступано август 2021.

[11] "RSA(cryptosystem)", RSA (cryptosystem) - Wikipedia, приступано август 2021.

[12] "Diffie-Hellman key excange", Diffie–Hellman key exchange - Wikipedia, приступано август 2021.

[13] "About end-to-end encryption", WhatsApp Help Center - About end-to-end encryption, приступано август 2021.

[14] "Whatsapp infrastructure", https://scontent.whatsapp.net/v/t39.8562-34/122249142_469857720642275_2152527586907531259_n.pdf/WA_Security_WhitePaper.pdf?ccb=1-5&_nc_sid=2fbf2a&_nc_ohc=OoNQIjAa6kQAX-xoD7M&_nc_ht=scontent.whatsapp.net&oh=ee1515179a09b7f932422d15706611c9&oe=611E1B59, приступано август 2021.

[15] Signal application blog, https://signal.org/blog, приступано август 2021.

[16] Autentikacione aplikacije, Microsoft PowerPoint - 07 autentikacione aplikacije [Compatibility Mode] (bg.ac.rs), приступано август 2021.

[17] "Kerberos – The Network Authentication Protocol", Kerberos: The Network Authentication Protocol (mit.edu), приступано август 2021.

[18] Jason Garman, Kerberos: The Definitive Guide. August, 2003

[19] Simplilearn, "What Is Kerberos, How Does It Work, and What Is It Used For?", What Is Kerberos, How Does It Work, and What Is It Used For? (simplilearn.com), приступано август 2021.

[20] "Angular material", Angular Material UI component library, приступано август 2021.

[21] "Cryptographic hash function" Cryptographic hash function - Wikipedia, приступано август 2021.

[22] "jsonwebtoken", jsonwebtoken - npm (npmjs.com), приступано август 2021.

[23] "Introduction to JSONWebTokens", JSON Web Token Introduction - jwt.io, приступано август 2021.

[24] "Socket.io". Socket.IO, приступано август 2021.

[25] "Socket.io - Introduction", Introduction | Socket.IO, приступано август 2021.