УНИВЕРЗИТЕТ У БЕОГРАДУ
МАТЕМАТИЧКИ ФАКУЛТЕТ
Студентски трг 16
11000 БЕОГРАД

УНИВЕРЗИТЕТ У БЕОГРАДУ
МАТЕМАТИЧКИ ФАКУЛТЕТ
БЕОГРАД
Рачунарство и информатика

МАСТЕР РАД

Протокол TLS за сигуран пренос података кроз Интернет

Студент: Милош Станковић
Број индекса: 1225/2011

Ментор: др Миодраг В. Живковић

Београд, 2013.

Садржај

1. Увод

Корисници услуга на интернету ретко воде рачуна о сигурности информација које размењују преко мреже. Доста порука које се преносе интернет инфраструктуром садржи поверљиве податке (нпр. личне податке) или осетљиве информације (нпр. финансијске извештаје). Такве податке потребно је заштити како би се спречило њихово прегледање, измена или злоупотреба. Један од првих облика заштите био је примена протокола SSL (енг. *Secure Sockets Layer*).

Протокол SSL је пројектовала је корпорација *„Netscape Communications“*, са намером да га искористи у свом веб прегледачу *„Netscape Navigator“*. Прва верзија SSL протокола појавила се 1995 године и то је била верзија 2.0, пошто *„Netscape“* никада није објавио верзију 1.0. Због великог броја сигурносних пропуста већ 1996. је објављена нова верзија протокола 3.0. Верзија 3.0 је донела потпуно преуређен протокол који је коришћен као основа за све новије верзије протокола.

Нови стандардни протокол објављен је 1999. године и назван је TLS (енг. *Transport Layer Security*). Основна намена протокола TLS је заштита података у комуникацији, као и омогућавање аутентификације саговорника. Овај протокол има различите намене (удаљени приступ, поруке електронске поште и сл.), а унапређене су и многе функционалности попут оних везаних за аутентификацију, рачунање кључева и сл.

Иако су после верзије 1.0 протокола TLS објављене и верзије 1.1 и 1.2, верзија 1.0 је и даље најзаступљенија па се у овом документу описује начин рада, предности и недостаци управо протокола TLS 1.0.

2. Протокол TLS за сигуран пренос података

Протокол TLS омогућује сигурну комуникацију две стране, клијента и сервера. Клијент започиње комуникацију, док сервер одговара на захтеве клијента. Пошто се TLS обично користи за сигурну размену података на интернету, TLS клијент је најчешће програм за читање веб страна, док у том случају веб странице преузимају улогу TLS сервера.

Улоге клијента и сервера се највише разликују у процесу размене порука и преговарања око сигурносних опција. Клијент као иницијатор комуникације шаље листу шифарских система које подржава, а сервер бира један од понуђених шифарских система и на тај начин одлучује који ће се шифарски систем користити у комуникацији. Детаљније о шифарским системима говори се у тачки 3.4.2. Након низа размењених порука, и клијент и сервер креирају сигурносне параметре којима се обезбеђује сигуран пренос података. Алгоритми за креирање сигурносних параметара описани су у тачки 3.6.

2.1 Поруке протокола TLS

Клијент и сервер комуницирају разменом порука протокола TLS. Овде се описује само функционалност тих порука, а њихов тачан формат описује се у тачки 3.

У наставку се наводе поруке протокола TLS са кратким описом:

Alert - обавештење да је дошло до сигурносног пропуста или грешке у комуникацији;

ApplicationData - подаци који се размењују између клијента и сервера, а који су фрагментисани, компримовани и шифровани од стране протокола TLS;

Certificate - порука којом се преноси сертификат са јавним кључем;

CertificateRequest - порука којом сервер тражи сертификат од клијента да би га аутентификовао;

CertificateVerify - порука којом клијент јавља серверу да поседује тајни кључ који одговара јавном кључу сертификата који је послао серверу;

ChangeCipherSpec - обавештење примаоцу да договорени сигурносни параметри почињу да се примењују након ове поруке;

ClientHello - порука којом клијент започиње TLS сесију и којом обавештава сервер о листи шифарских система које подржава;

ClientKeyExchange - порука којом се шаље јавни кључ клијента шифрован јавним кључем сервера;

Finished - обавештење да је размена кључева и аутентификација успешно обављена и да је шифрована комуникација успостављена;

HelloRequest - порука коју сервер шаље клијенту и којом му даје сигнал да започне нови процес преговора за обезбеђење шифроване комуникације;

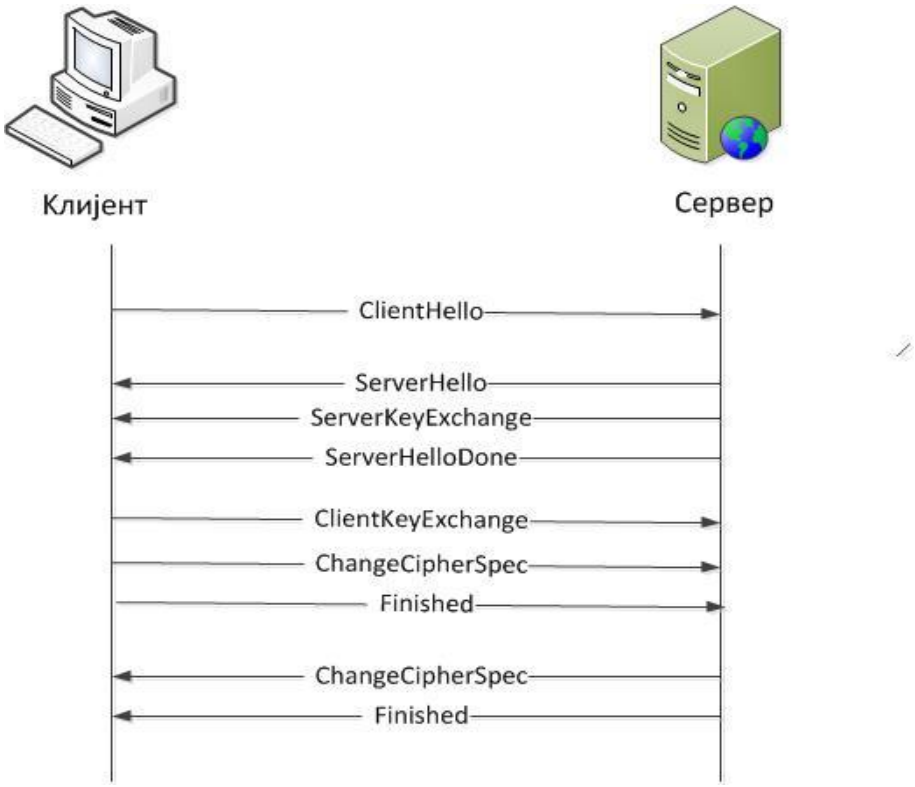
ServerHello - одговор на поруку *ClientHello* којом сервер обавештава клијента који шифарски ситем је изабрао да се користити у преговорима;

ServerHelloDone - порука којом се клијент обавештава да му је сервер послао све поруке потребне за размену кључева;

ServerKeyExchange - порука којом се шаље јавни кључ сервера који ће се користити за комуникацију;

2.2 Успостављање шифроване комуникације

Основни задатак протокола TLS је да клијенту и серверу омогући успостављање канала за шифровану комуникацију. Најједноставнији пример успостављања шифроване комуникације је приказан на слици 2.1.



Слика 2.1 Успостављање шифроване комуникације

Клијент започиње TLS сесију слањем поруке *ClientHello* којом серверу шаље листу шифарских система које предлаже;

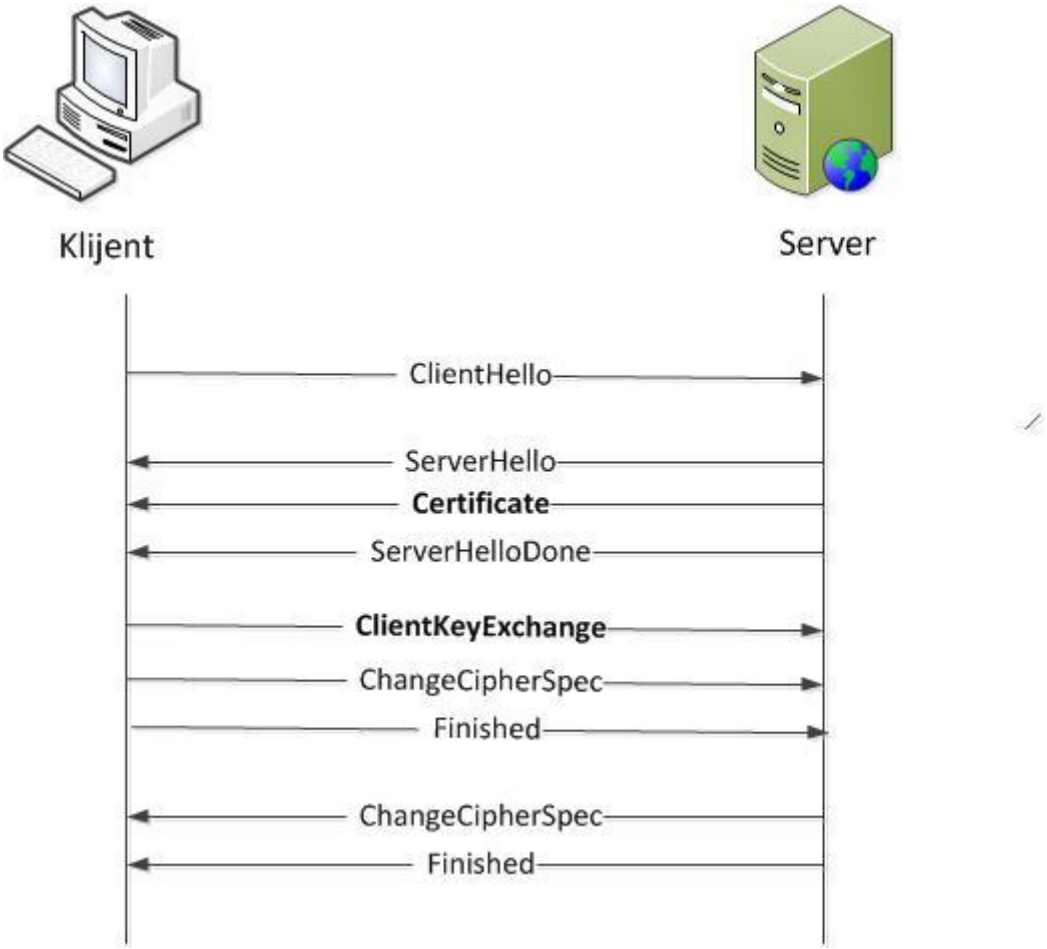
Сервер из листе шифарских система коју је добио од клијента, бира први шифарски систем који и сам подржава, а затим о томе обавештава клијента поруком *ServerHello*; поруком *ServerKeyExchange* сервер затим шаље свој криптографски јавни кључ који ће се користити за комуникацију и завршава ову фазу преговора поруком *ServerHelloDone*.

Клијент свој јавни кључ, шифрован јавним кључем сервера, шаље у *ClientKeyExchange* поруци, а затим користећи поруку *ChangeCipherSpec*, обавештава сервер да ће од наредне поруке почети да примењује договорене сигурносне параметре и завршава ову фазу преговора поруком *Finished*. То је уједно и прва порука коју клијент шаље примењујући договорене сигурносне параметре и коју сервер треба да искористи за проверу договорених параметара.

Сервер на то одговара поруком *ChangeCipherSpec* и њоме обавештава клијента да почиње са применом договорених сигурносних параметара, а затим и поруком *Finished* на основу које клијент може да провери успостављене сигурносне параметре.

2.3 Аутентификација сервера

Постоје случајеви када корисник шаље поверљиве податке, нпр. податке о својој кредитној картици, и тада је потребно да буде сигуран у идентитет сервера са којим комуницира. Из тог разлога у протоколу TLS постоји механизам за аутентификацију сервера. На овај начин клијент је у могућности да провери да ли је сертификат који је добио од сервера издат од тела за издавање сертификата коме верује и на тај начин провери интегритет сервера. Пример успостављања шифроване комуникације где клијент проверава интегритет сервера је приказан на слици 2.2.



Слика 2.2 Аутентификација сервера

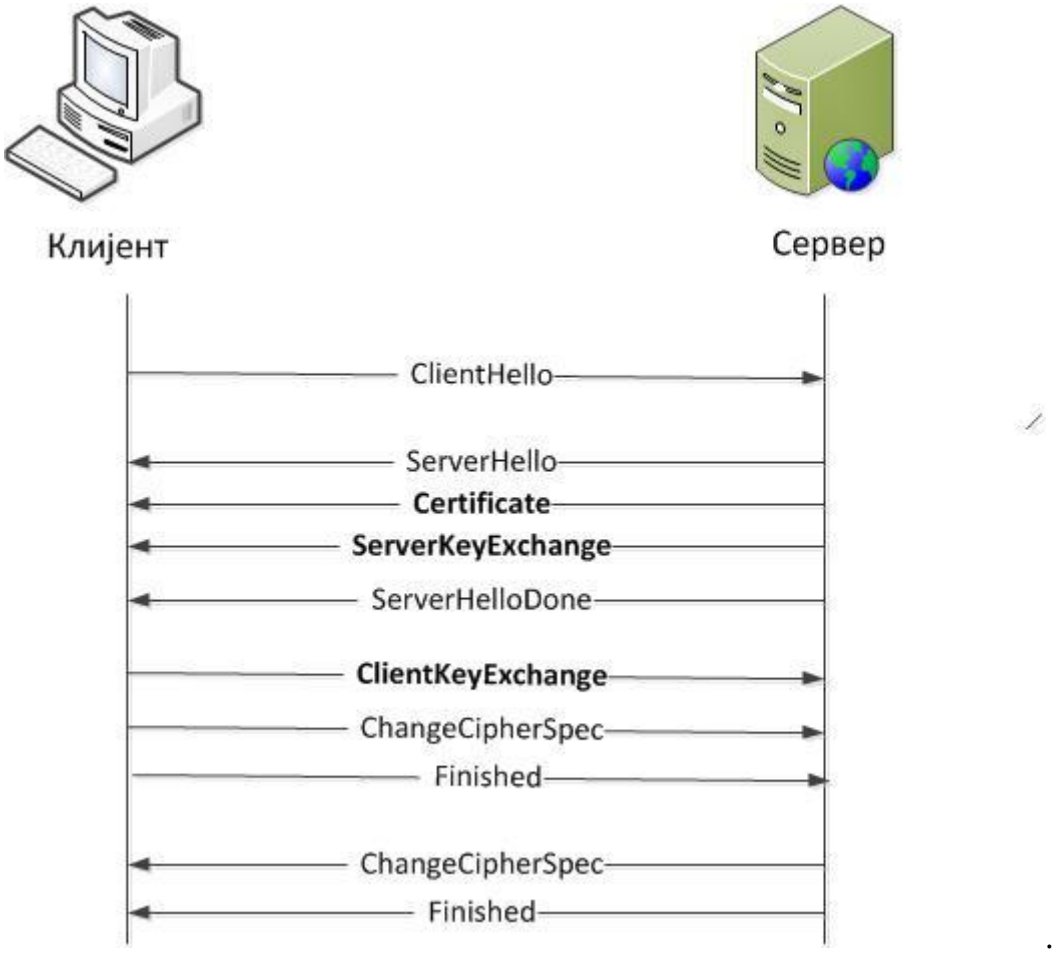
У односу на основни процес успостављања шифроване комуникације описан у тачки 2.2, у овом случају уместо да пошаље само јавни кључ поруком *ServerKeyExchange*, сервер у поруци *Certificate* шаље сертификат који садржи његов јавни кључ.

Пре него што пошаље свој јавни кључ, клијент проверава да ли може да верује сертификату који је добио од сервера, а затим свој кључ шифрује јавним кључем сервера и шаље га у поруци *ClientKeyExchange*. На тај начин, клијент се уверава да сервер са којим комуницира поседује одговарајући приватни кључ.

2.4 Аутентификација и шифровање као посебни кораци

У процесу описаном у тачки 2.3, јавни кључ који је клијент добио од сервера се користио и за аутентификацију сервера и за шифровање јавног кључа клијента. Постоје сигурносни алгоритми (нпр. DSA–*Digital Signature Algorithm*) који се могу користити само за потписивање, али не и за шифровање. Уколико је серверски сертификат потписан једним таквим алгоритмом, јасно је да клијент у том случају свој јавни кључ не може шифровати јавним кључем који је добијен у сертификату сервера.

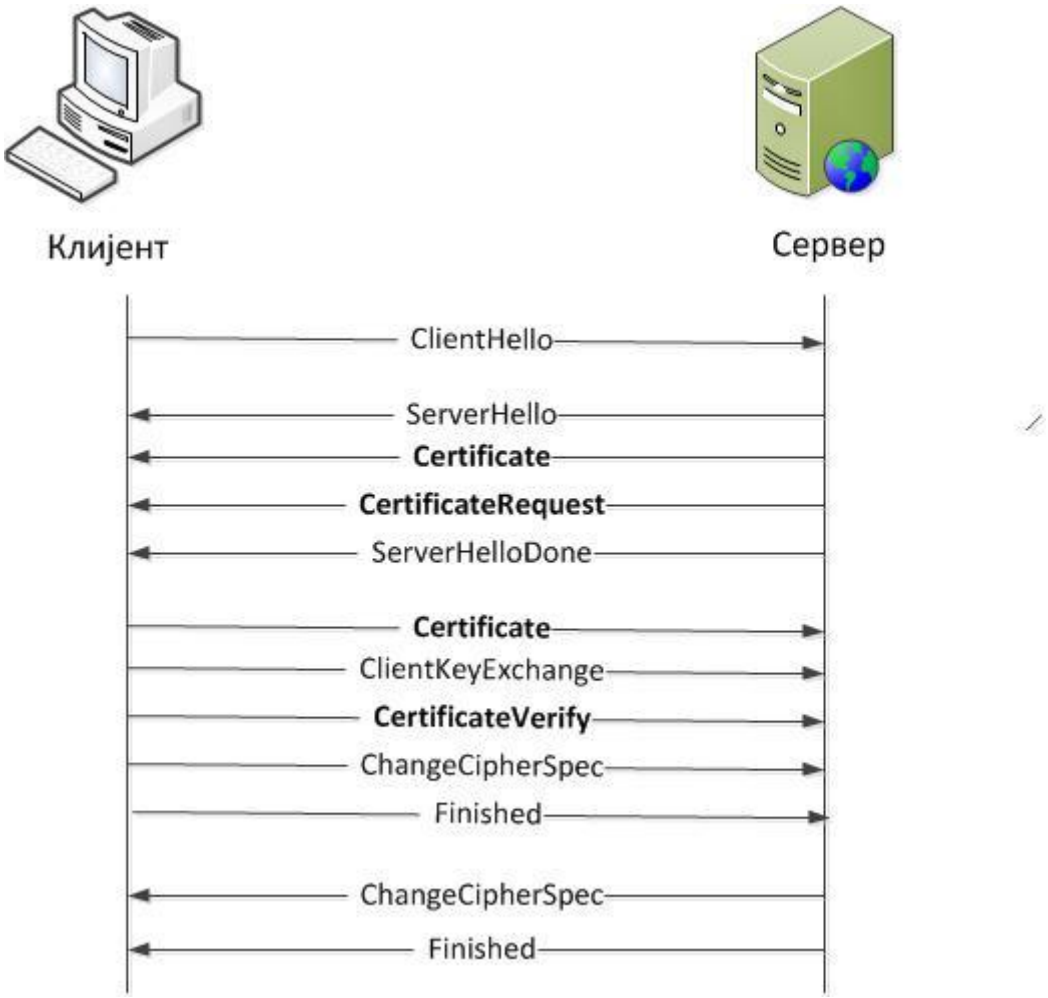
У том случају се јавни кључ добијен од сервера у поруци *Certificate*, користи само за потврду интегритета сервера, односно његову аутентификацију, а кључ који се користи за шифровање података, сервер шаље у посебној *ServerKeyExchange* поруци, одмах након слања поруке *Certificate*. Порука *ServerKeyExchange* се потписује приватним кључем који одговара јавном кључу који се налази у серверском сертификату, и на тај начин клијент може да утврди да сервер заиста поседује приватни кључ који одговара јавном кључу из његовог сертификата. На слици 2.3 је приказана размена порука у случају када су аутентификација и шифровање одвојени у посебне кораке.



Слика 2.3 Аутентификација и енкрипција у посебним корацима

2.5 Узајамна аутентификација сервера и клијента

У неким случајевима постоји потреба да и сервер потврди интегритет клијента. Један пример успостављања шифроване комуникације при коме се врши и аутентификација клијента и аутентификација сервера је приказан на слици 2.4.



Слика 2.4 Узајамна аутентификација клијента и сервера

Клијент започиње TLS сесију слањем поруке *ClientHello* којом шаље серверу листу шифарских систеам које предлаже.

Сервер из листе шифарских система коју је добио од клијента, бира први шифарски систем који и сам подржава, а затим о томе обавештава клијента поруком *ServerHello*; у поруци *Certificate* сервер клијенту шаље свој сертификат, а затим од клијента тражи клијентски сертификат поруком *CertificateRequest*. На крају сервер завршава ову фазу преговора поруком *ServerHelloDone.*

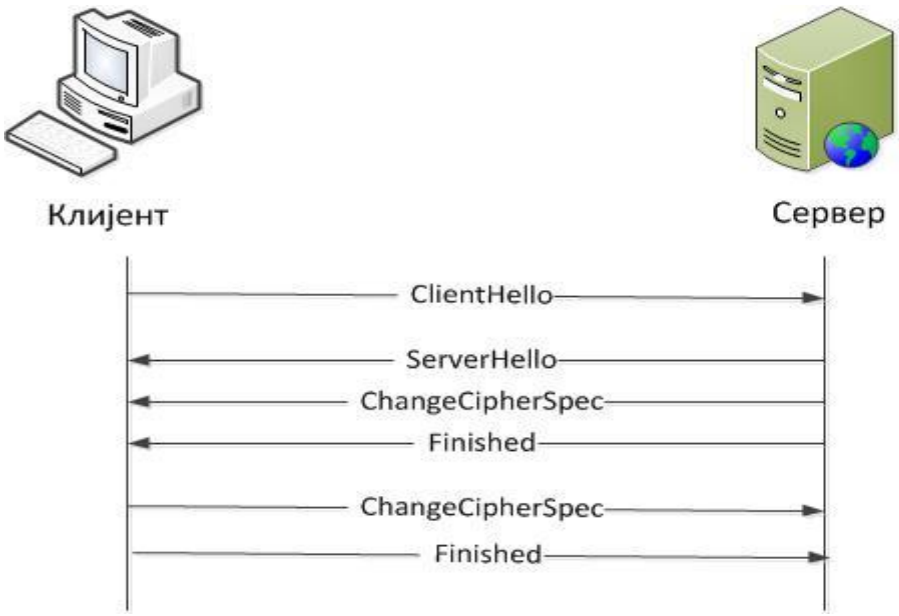
У поруци *Certificate* клијент шаље свој сертификат, а затим у поруци *ClientKeyExchange* и свој јавни кључ који претходно шифрује јавним кључем сервера. Након тога у поруци *CertificateVerify* клијент серверу шаље податке који служе за верификацију клијентског сертификата. На крају клијент, користећи поруку *ChangeCipherSpec*, обавештава сервер да ће од наредне

поруке почети да примењује договорене сигурносне параметре и завршава ову фазу преговора поруком *Finished*. То је уједно и прва порука коју клијент шаље примењујући договорене сигурносне параметре и коју сервер треба да искористи за проверу договорених параметара.

Сервер на то одговара поруком *ChangeCipherSpec* и њоме обавештава клијента да почиње са применом договорених сигурносних параметара, а затим и поруком *Finished* на основу које клијент може да провери успостављене сигурносне параметре.

2.6 Наставак претходне сесије

У претходним примерима се могло видети да успостављање једне TLS сесије може бити компликован процес са много криптографских израчунавања и великим бројем размењених порука. Уместо договарања нових сигурносних параметара, клијент и сервер могу да одлуче да искористе параметре који су били договорени у једној од претходних сесија. Такав пример приказан је на слици 2.5.



Слика 2.5 Наставак претходне сесије

Уколико жели да предложи наставак неке од претходних сесија, клијент у поруку *ClientHello* ставља идентификатор претходне сесије. По пријему такве поруке од клијента, сервер проверава да ли поседује одговарајући идентификатор сесије. Уколико сервер пронађе одговарајући идентификатор и уколико жели да настави сесију, у поруци *ServerHello* враћа исти тај идентификатор. У случају да сервер не поседује одговарајући идентификатор сесије, генерисаће нови идентификатор и започеће се потпуни преговори око сигурносних параметара.

Иако се на овај начин значајно може убрзати успостављање везе, сигурност се у тим ситуацијама смањује и на тај начин се нападачима отварају врата за нове врсте напада.

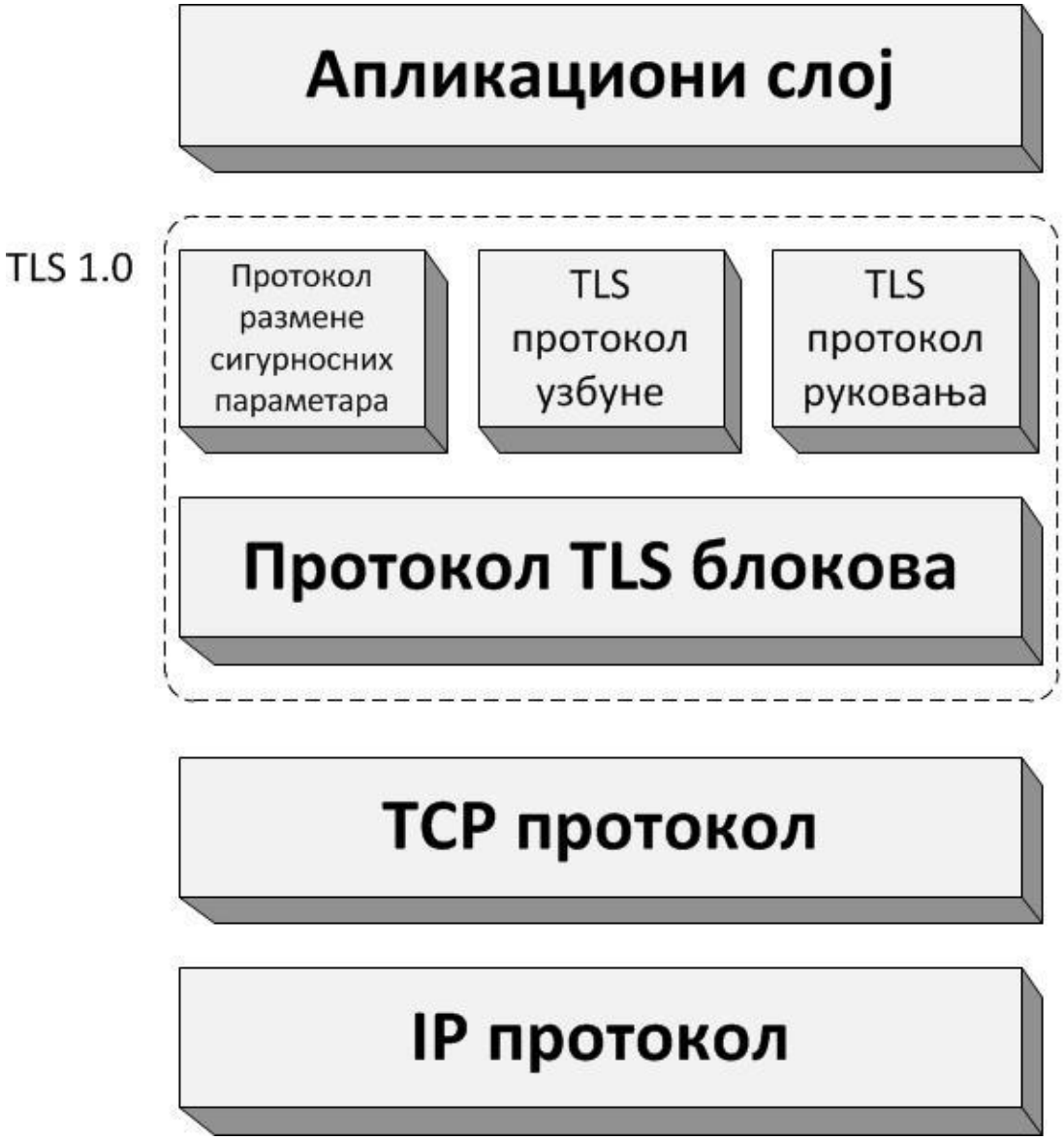
3. Формати порука

Протокол TLS се састоји од четири потпротокола:

1. Протокол TLS блокова (енг. *TLS Record Protocol*)
2. TLS протокол руковања (енг. *TLS Handshake Protocol*)
3. TLS протокол размене сигурносних параметара (енг. *ChangeCipgerSpec Protocol*)
4. TLS протокол узбуне (енг. *TLS Alert Protocol*)

Користећи ове протоколе две стране су у могућности да се договарају око сигурносних параметара, међусобно се аутентификују, примењују договорене сигурносне параметре и сигнализирају уколико дође до неке грешке.

На слици 3.1 се може видети међусобна веза између ових протокола.

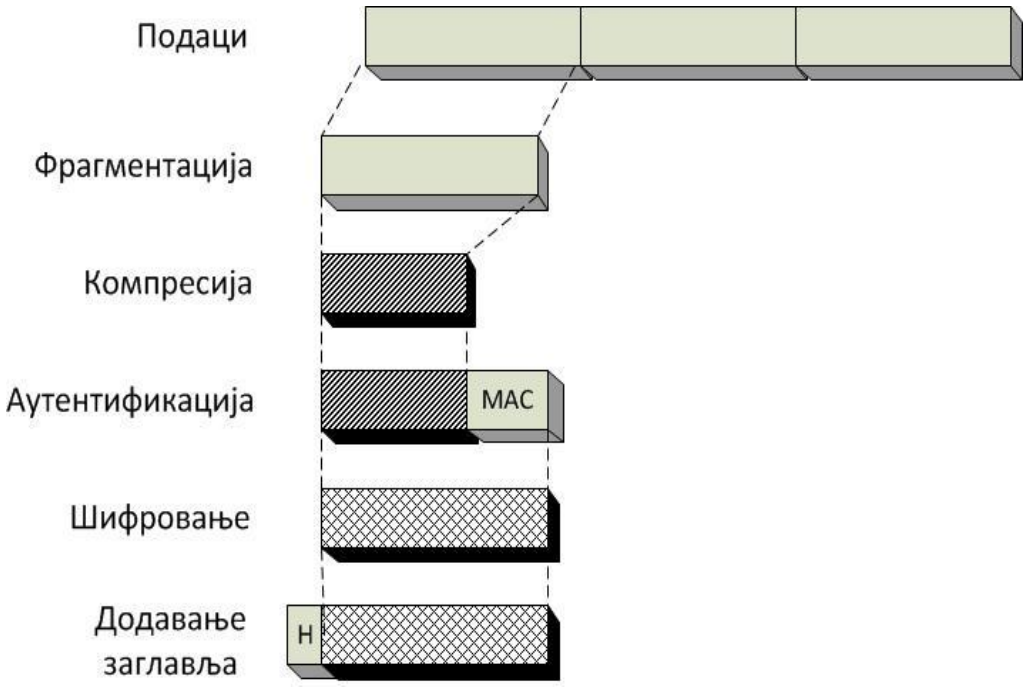


Слика 3.1 Међусобна веза протокола који чине протокол TLS

TLS протокол руковања је одговоран за идентификацију субјеката у комуникацији, посредовање шифарских система и контролу и размену *premaster_secret* вредности (видети тачку 3.6.2) која се користи за генерисање *master_secret* вредности (видети тачку 3.6.3), а која служи за генерисање кључева за шифровање и аутентификацију. Функција протокола измене сигурносних параметара је да обавести слој TLS блокова о променама у сигурносним параметрима. Протокол за обавештења служи да другој страни јави уколико се јаве неке грешке у току проверавања примљене поруке, као и о свим неслагањима која се могу десити током процеса руковања.

3.1 Протокол TLS блокова

Протокол TLS блокова (енг. *TLS Record Protocol*) је одговоран за пренос блокова информација између клијента и сервера. При слању, протокол TLS блокова прихвата податке од апликационог слоја, врши њихову фрагментацију у блокове, опционо те податке компримује, додаје аутентификацију (*MAC – message authentication code*), шифрује податке, додаје заглавље и предаје слоју TCP ради слања. На пријемној страни подаци се преузимају од слоја TCP, дешифрују се, израчунава се и пореди MAC ради провере веродостојности, врше се декомпресија и дефрагментација података и на крају се подаци предају апликационом нивоу, видети слику 3.2.



Слика 3.2 Обрада података приликом слања

Апликациони блокови се деле на блокове не веће од 2^{14}=16384 бајтова. Компресија се изводи опционо и мора бити без губитака. На конкатенацију компримованих података, параметрима компресије и симетричног MAC кључа, израчунава се хеш поруке по алгоритму MD5 ili SHA1 и тако осигурава аутентичност поруке. Последњи корак је додавање заглавља које садржи следећа четири поља:

- Тип садржаја (енг. *Content type*): поље величине 1 бајт које идентификује један од четири протокола вишег слоја, видети табелу 3.1 (вредности у тавели су представљене декадним бројевима).

Вредност	Протокол
20	ChangeCipherSpec
21	Alert
22	Hanshake
23	Application

Табела 3.1 Типови протокола

- Редни број верзије протокола (енг. *Major Version*): Поље величине 1 бајт које дефинише верзију протокола

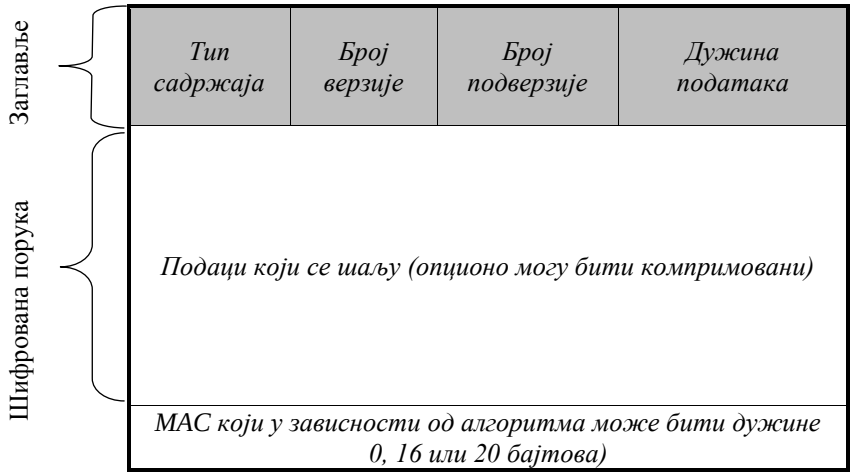
- Број подверзије или ревизије (енг. *Minor Version*): Поље величине 1 бајт које дефинише подверзију протокола, видети табелу 3.2.

Протокол	Редни број верзије	Број подверзије
SSLv3.0	3	0
TLSv1.0	3	1
TLSv1.1	3	2
TLSv1.2	3	3

Табела 3.2 Верзије SSL протокола

- Дужина компримованих података (енг. Compressed Length): Поље величине 2 бајта које садржи дужину података који се шаљу.

Коначни изглед пакета протокола TLS блокова приказан је на слици 3.3:



Слика 3.3 Формат пакета протокола TLS блокова

3.2 Протокол размене сигурносних параметара

Од четири протокола која чине протокол TLS, протокол размене сигурносних параметара (енг. *ChangeCipherSpec Protocol*) је најједноставнији и састоји се од само једне поруке која је шифрована и компримована користећи тренутно стање конекције. Порука протокола размене сигурносних параметара је дужине 1 бајт и њена вредност је увек 1. По пријему ове поруке прималац прелази на договорене сигурносне поруке. Порука размене сигурносних параметара која је упакована у поруку протокола TLS блокова је приказана на слици 3.4.

Тип садржаја	***Бр. верзије***	***Бр. подверзије***	***Дужина***	
20	*3*	*1*	*0*	*1*
CCS:1				

Слика 3.4 Порука *ChangeCipherSpec*

3.3 Протокол узбуне

Протокол узбуне (енг. *TLS Alert Protocol*) је механизам унутар протокола TLS који се користи да се друга страна која учествује у комуникацји упозори да је дошло до одређене нерегуларности или грешке у процесу успостављања шифроване комуникације. Порука протокола узбуне се састоји од два поља, при чему је свако дужине 1 бајт. Прво поље служи да дефинише ниво опасности, док се друго користи за опис узбуне. Изглед поруке протокола узбуне којој је додато заглавље протокола TLS блокова је приказан на слици 3.5

Тип садржаја	***Бр. верзије***	***Бр. подверзије***	***Дужина***	
21	*3*	*1*	*0*	*2*
Ниво опасности	*Опис*			

Слика 3.5 Порука протокола узбуне

Порука узбуне се, у зависности од ситуације, може послати из различитих слојева протокола TLS: апликативни слој шаље поруку узбуне да би иницирао крај сигурне везе, протокол руковања је шаље уколико дође до неке грешке у овој фази преговора, а протокол TLS блокова када интегритет поруке није у реду.

Поље за ниво опасности служи да се прималац обавести о озбиљности проблема и може имати две вредности, 1 или 2. Уколико је вредност овог поља 1, порука узбуне представља само упозорење, а уколико ово поље има вредност 2 порука представља фаталну грешку. Порука узбуне која представља фаталну грешку, изазива тренутни прекид шифроване комуникације.

С обзиром да фатална грешка изазива крај комуникације, TLS је осетљив на нападе типа „отказ сервиса“ (енг. *denial of service*). У случају да нападач успе да замени оригиналну поруку, поруком која означава фаталну грешку, сигурна комуникација се прекида.

Друго поље поруке узбуне даје опис грешке до које је дошло и може да има једну од вредности приказаних у табели 3.3.

Вред.	Назив	Фатална грешка	Опис
0	close_notify	Да	Порука којом се иницира крај сигурне комуникације.
10	unexpected_message	Да	Упозорење да је примљена неправилна порука.
20	bad_record_mac	Да	Упозорење да је примљена порука са неправилним MAC-ом.
21	decryption_failed	Да	Упозорење да је дошло до грешке приликом дешифровања примљене поруке.
22	record_overflow	Да	Упозорење којим се јавља да је дужина поруке након дешифровања већа од 2^{14}+2048 бајтова.
30	decompression_failure	Да	Пошиљалац јавља да није успео да декомпримује примљену поруку.
40	handshake_failure	Да	Упозорење да је дошло до грешке у фази преговарања о сигурносним параметрима.
42	bad_certificate	Да	Обавештење да је примљен неисправан сертификат.
43	unsupported_certificate	Да	Формат примљеног серфификата није подржан.
44	certificate_revoked	Да	Примљени сертификат је опозван од стране тела које га је издало.
45	certificate_expired	Да	Примљеном сертификату истекла је важност.
46	certificate_unknown	Да	Пошиљалац јавља да има одређених проблема са примљеним сертификатом.
47	illegal_parameter	Да	Порука којом пошиљалац обавештава да је приликом преговора око сигурносних параметара примио поруку са параметрима који су неправилни или неконзистентни са осталим примљеним параметрима.
48	unknown_ca	Да	Грешка којом се обавештава да пошиљалац не може идентификовати или да не верује телу које је потписало примљени сертификат.
49	access_denied	Да	По пријему исправног сертификата, пошиљалац овог упозорења одлучује да не жели да настави преговоре и о томе обавештава другу страну.

50	decode_error	Да	Пошиљалац обавештава да због неисправног садржаја или дужине, порука не може да се декодира.
51	decrypt_error	Да	Обавештење којим се јавља да дешифровање примљене поруке није било успешно.
60	export_restriction	Да	Грешка којом се јавља да је дошло до кршења америчких правила о извозу криптографских материјала.
70	protocol_version	Да	Грешка којом пошиљалац јавља да не подржава тражену верзију протокола.
71	insufficient_security	Да	Упозорење којим сервер прекида преговоре јер захтева сигурносни пакет већег степена сигурности од оног који је клијент предложио.
80	internal_error	Да	Пошиљалац јавља да је код њега дошло до грешке која нема везе са самим протоколом и која му онемогућава даљи наставак преговора. Ова порука је увек фатална.
90	user_canceled	Да	Обавештење да је пошиљалац прекинуо преговоре из разлога који није везан за грешку у протоколу. Ова порука представља упозорење и увек би требало да је следи *CloseNotify* порука.
100	no_renegotiation	Не	Пошиљалац јавља да одбија захтев за поновне преговоре око сигурносних параметара.

Табела 3.3 Опис порука узбуне

3.4 Протокол руковања

Протокол руковања (енг. *Handshake protocol*) је најважнији део у процесу преговора око TLS параметара, јер је задужен за генерисање свих параметара за шифровање у једној сесији шифроване комуникације. Протокол руковања се налази изнад протокола TLS блокова и у њега пакује поруке које генерише. Као што је већ напоменуто, једна порука протокола TLS блокова може садржати и више од једне поруке руковања.

Протокол руковања једне TLS сесије се састоји од следећих корака:

- Размена *hello* порука како би се направио договор око алгоритама који ће се користити, размениле случајне вредности клијента и сервера и евентуално врши провера да ли се наставља нека од претходних сесија.

- Размена неопходних криптографских параметара који су потребни да би се клијент и сервер договорили око *premaster_secret* вредности (видети тачку 3.6.2).

- Размена сертификата који су потребни да би се сервер и клијент међусобно аутентификовали.

- Генерисање *master_secret* вредности (видети тачку 3.6.3) користећи случајне вредности и *premaster_secret* вредност (видети тачку 3.6.2).

- Снабдевање протокола TLS блокова сигурносним параметрима.

- Провера да су обе стране у комуникацији израчунале исте сигурносне параметре и да је размена порука протекла без ометања од стране нападача.

Свака порука руковања почиње бајтом којим се дефинише врста поруке, након чега следе три бајта којима се дефинише дужина тела поруке протокола руковања. Дужина се изражава у бајтовима и у дужину поруке се не рачунају поља за тип поруке и поље које означава дужину поруке.

3.4.1 Порука *HelloRequest*

Помоћу поруке *HelloRequest*, сервер тражи од клијента да поново започну преговоре око TLS параметара. Иако се ова порука не користи често, она серверу даје неке додатне могућности. Ако се на пример нека конекција користи толико дуго да је сигурност података који се размењују нарушена, сервер поруком *HelloRequest* од клијента тражи да договоре нове сигурносне параметре или нове кључеве. Формат поруке *HelloRequest* је веома једноставан и може се видети на слици 3.6. Тип поруке *HelloRequest* је 0 и с обзиром да је тело поруке празно и дужина поруке је 0.

Тип садржаја	*Бр. верзије*	*Бр. подверзије*	*Дужина*
22	*3*	*1*	*0*
4	*Тип: 0*	*Дужина:0*	*0*
0			

Слика 3.6 Порука HelloRequest

3.4.2 Порука *ClientHello*

Клијент започиње успостављање шифровање комуникације слањем поруке *ClientHello,* чији је формат приказан на слици 3.7. Тип поруке *ClientHello* је 1, док њена дужина може да варира и налази се одмах после типа поруке. Следећа два бајта су резервисана за верзију протокола TLS. Иако се податак о верзији протокола налази и унутар протокола TLS блокова, у теорији постоји могућност да се протокол TLS блокова и протокол руковања развијају независно један од другог.

Након верзије протокола налази се случајна вредност клијента која је дужине 32 бајта. По спецификацији TLS протокола, предложено је да се за прва четири бајта користе тренутни датум (број секунди који је протекао од 1.

Јануара 1970 године) и да се на тај начин смањи вероватноћа појаве истих случајних бројева што би могло угрозити сигурност.

<table>
<tr><th>Тип садржаја</th><th>Бр. верзије</th><th>Бр. подверзије</th><th>Дужина</th></tr>
<tr><td>22</td><td>3</td><td>1</td><td>...</td></tr>
<tr><td>...</td><td>Тип: 1</td><td>Дужина:</td><td>Дужина:</td></tr>
<tr><td>Дужина:</td><td>Бр. верзије</td><td>Бр. подверзије</td><td></td></tr>
<tr><td colspan="3">Случајна вредност клијента
(32 бајта)</td><td>Дужина ИД сесије</td></tr>
<tr><td colspan="4">Идентификатор сесије</td></tr>
<tr><td colspan="2">Дужина листе која садржи шифарске системе (1 бајт)</td><td colspan="2">Шифарски систем 1</td></tr>
<tr><td colspan="2">Шифарски систем 2</td><td colspan="2"></td></tr>
<tr><td colspan="4">...</td></tr>
<tr><td colspan="2">Шифарски систем n</td><td colspan="2">Дужина листе метода за компресију</td></tr>
<tr><td>Метод за комп. 1</td><td colspan="2">...</td><td>Метод за комп. n</td></tr>
</table>

Слика 3.7 Порука ClientHello

Наредни бајт садржи дужину идентификатора сесије изражену у бајтовима, а након тога и сам идентификатор сесије. Уколико клијент не наставља неку од претходних сесија, идентификатор сесије се изоставља, а његова дужина се поставља на 0. Дужина идентификатора сесије је протоколом TLS ограничена на 32 бајта (може бити и мања), док нема ограничења по питању садржаја идентификатора. Како се идентификатор сесије преноси у поруци *ClientHello*, дакле пре него што се употреби било какво шифровање, посебна пажња треба да се обрати да се у идентификатор не стављају важне информације или информације које могу да угрозе сигурност.

Порука *ClientHello* даље садржи листу која је састављена од шифарских система које клијент подржава. Та листа почиње бајтом који садржи дужину листе у бајтовима,а за сваки предложени шифарски систем у листи, резервисана су 2 бајта, тако би листа која садржи 5 предложених шифарских система имала дужину од 10 бајтова. У табели 3.4 приказани су шифарски системи подржани протоколом TLSv1.0.

Последња поља у поруци *ClientHello* односе се на листу метода за компресију које предлаже клијент. Као и листа шифарских система, и ова листа почиње бајтом који садржи дужину листе. За разлику од листе шифарsких система, овде је сваки метод за компресију представљен једим бајтом. Уколико је дужина листе 1, а следећи бајт је 0, значи да није предложена ни једна метода за компресију.

Вредност	Пакет за шифровање
0,0	SSL_NULL_WITH_NULL_NULL
0,1	SSL_RSA_WITH_NULL_MD5
0,2	SSL_RSA_WITH_NULL_SHA
0,3	SSL_RSA_EXPORT_WITH_RC4_40_MD5
0,4	SSL_RSA_WITH_RC4_128_MD5
0,5	SSL_RSA_WITH_RC4_128_SHA
0,6	SSL_RSA_EXPORT_WITH_RC3_CBC_40_MD5
0,7	SSL_RSA_WITH_IDEA_CBC_SHA
0,8	SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
0,9	SSL_RSA_WITH_DES_CBC_SHA
0,10	SSL_RSA_WITH_3DES_EDE_CBC_SHA
0,11	SSL_DH_DSS_EXPORT_WITH_DES40_CBC_SHA
0,12	SSL_DH_DSS_WITH_DES_CBC_SHA
0,13	SSL_DH_DSS_WITH_3DES_EDE_CBC_SHA
0,14	SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA
0,15	SSL_DH_RSA_WITH_DES_CBC_SHA
0,16	SSL_DH_RSA_WITH_3DES_EDE_CBC_SHA
0,17	SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
0,18	SSL_DHE_DSS_WITH_DES_CBC_SHA
0,19	SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
0,20	SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
0,21	SSL_DHE_RSA_WITH_DES_CBC_SHA
0,22	SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
0,23	SSL_DH_anon_EXPORT_WITH_RC4_40_MD5
0,24	SSL_DH_anon_WITH_RC4_128_MD5
0,25	SSL_DH_annon_EXPORT_WITH_DES40_CBC_SHA
0,26	SSL_DH_anon_WITH_DES_CBC_SHA
0,27	SSL_DH_anon_WITH_3DES_EDE_CBC_SHA

Табела 3.4 Шифарски системи подржани протоколом TLS

3.4.3 Порука *ServerHello*

Поруку *ServerHello* сервер шаље као одговор на поруку *ClientHello* примљену од клијента. Тело поруке *ServerHello* почиње навођењем типа поруке чија је вредност у овом случају 2, за којим следи дужина саме поруке.

У поруку *ServerHello* сервер може да укључи и идентификатор сесије и на тај начин омогући клијенту да у будућности покуша да успостави исту сесију. Уколико сервер не жели клијенту да омогући успостављање неке од претходних сесија, једноставно изоставља идентификатор сесије тако што његову дужину постави на 0.

Из сваке листе коју је добио од клијента, сервер бира један шифарски систем и један метод за компресију и то се у поруци *ServerHello* налази одмах након идентификатора сесије.

Формат поруке *ServerHello* је приказан на слици 3.8

Тип садржаја	***Бр. верзије***	***Бр. подверзије***	***Дужина***
22	*3*	*1*	*0*
...	*Тип: 2*	*Дужина:*	*Дужина:*
Дужина:	*Бр. верзије*	*Бр. подверзије*	
Случајна вредност сервера (32 бајта)			*Дужина ИД сесије*
Идентификатор сесије			
Шифарски систем		*Метод за компресију*	

Слика 3.8 Порука *ServerHello*

3.4.4 Порука *Certificate*

Поруку *Certificate* могу слати и клијент и сервер да би се другој страни омогућило да аутентификује саговорника. Порука *Certificate* почиње пољем које садржи тип поруке који је овом случају има вредност 11. Након тога следи дужина тела поруке које садржи низ сертификата. На почетку низа сертификата налазе се три бајта која означавају дужину низа, а та дужина је увек за три мања од дужине тела поруке. Пре сваког сертификата налазе се три бајта у која се уписује дужина сертификата који следи.

Низ сертификата мора да буде хијерархијски сложен, при чему је први сертификат у низу увек сертификат пошиљаоца. Сваки следећи сертификат у низу је сертификат тела које је издало претходни сертификат из низа. Низ се тако наставља све док се не дође до сертификата који је издало тело за издавање сертификата.

Тип садржаја	***Бр. верзије***	***Бр. подверзије***	***Дужина***
22	*3*	*1*	*0*
...	*Тип: 11*	*Дужина:*	*Дужина:*
Дужина:	*Дужина првог сертификата*		
Сертификат 1			
...			
Дужина н-тог сертификата			
Сертификат н			

Слика 3.9 Порука Certificate

3.4.5 Порука *ServerKeyExchange*

Поруком *ServerKeyExchange* сервер шаље клијенту свој јавни кључ. Формат ове поруке зависи од алгоритма за размену кључева који се користи. На сликама 3.10 и 3.11 приказани су формати порука које за размену кључева користе алгоритам Дифи-Хелман (енг. *Diffie-Hellman*) , односно алгоритам RSA. Без обзира на то који се начин размене користи, тип серверске поруке размене кључева је 12.

На слици 3.10 се још види да се осим типа и дужине поруке преносе и Дифи-Хелман параметри (p, q, y_s), а пре саме вредности сваког параметра налази се поље које означава његову дужину. Параметар *p* представља велики прост број који представља групу и који се користи у Дифи-Хелман операцијама, параметар *q* представља генератор, а параметар y_s је јавна Дифи-Хелман вредност сервера.

Тип садржаја	***Бр. верзије***	***Бр. подверзије***	***Дужина***
22	*3*	*1*	*...*
...	*Тип: 12*	*Дужина:*	*Дужина:*
Дужина:	*Дужина p вредности*		*p ...*
… вредност		*Дужина q вредности*	
q вредност		*Дужина* y_s *вредности*	
y_s *вредност*			
Дигитално потписани MD5 хеш вредност (16 бајтова)			
Дигитално потписани SHA1 хеш вредност (20 бајтова)			

Слика 3.10 Порука ServerKeyExchange са Дифи-Хелман параметрима

Слика 3.11 приказује како изгледа порука када се користи алгоритам RSA за размену кључева. Порука се тада састоји од RSA модула и јавног експонента, а слично као у претходном случају, пре саме вредности сваког од параметара, наводи се његова дужина.

Тип садржаја	***Бр. верзије***	***Бр. подверзије***	***Дужина***
22	*3*	*1*	*...*
...	*Тип: 12*	*Дужина:*	*Дужина:*
Дужина:	*RSA мод- дужина*		*RSA ...*
… мод вредност		*RSA експонент - дужина*	
RSA експонент - вредност			
Дигитално потписани MD5 хеш вредност (16 бајтова)			
Дигитално потписани SHA1 хеш вредност (20 бајтова)			

Слика 3.11 Порука ServerKeyExchange са RSA параметрима

3.4.6 Порука *CertificateRequest*

Како би га аутентификовао, сервер од клијента прво тражи сертификат. То чини слањем поруке *CertificateRequest*. Поред тога што том поруком сервер захтева да му клијент пошаље свој сертификат, на тај начин наводи и врсту сертификата који су му прихватљиви.

Тип поруке којом се захтева клијентски сертификат је 13. Након типа поруке налази се дужина саме поруке. У наставку поруке следи листа прихватљивих типова сертификата. Листа почиње са једним бајтом који представља дужину листе, након чега следе типови прихватљивих сертификата. За сваки тип сертификата резервисан је по један бајт.

Тип садржаја	***Бр. верзије***	***Бр. Подверзије***	***Дужина***
22	*3*	*1*	*0*
...	*Тип: 13*	*Дужина:*	*Дужина:*
Дужина:	*Дужина листе типова сертификата*	*Тип сертификата 1*	*Тип сертификата 2*
...	*Тип сертификата н*	*Дужина листе CA сертификата*	*Дужина…*
... CA1	*CA1*		
...			
Дужина CAn	*CAn*		

Слика 3.12 Порука CertificateRequest

Типови сертификата подржани протоколом TLS приказани су у табели 3.5.

Вредност	Тип сертификата
1	RSA потпис и размена кључева
2	DSA потпис
3	RSA потпис са Дифи-Хелман разменом кључева
4	DSA потпис са Дифи-Хелман разменом кључева
5	RSA потпис са краткотрајном Дифи-Хелман разменом кључева
6	DSA потпис са краткотрајном Дифи-Хелман разменом кључева

Табела 3.5 Типови сертификата подржани протоколом TLS

Након типова сертификата наводи се листа тела које издају сертификате, а које сервер сматра прихватљивим. Листа почиње са два бајта која представљају дужину листе, за којима следе имена тела која издају сертификате. Свако име има и поље које представља његову дужину.

3.4.7 Порука *ServerHelloDone*

Поруком *ServerHelloDone* завршава се део преговора на серверској страни и она не садржи никакве додатне информације. Тип завршне поруке сервера је 14, а дужина тела те поруке је 0.

Тип садржаја	***Бр. верзије***	***Бр. Подверзије***	***Дужина***
22	*3*	*1*	*0*
4	*Тип: 14*	*Дужина:0*	*Дужина:0*
Дужина:0			

Слика 3.13 Порука ServerHelloDone

3.4.8 Порука *ClientKeyExchange*

Поруком *ClientKeyExchange* се серверу прослеђују подаци помоћу којих се израчунава кључ који се користити у комуникацији. Формат поруке зависи од алгоритма који се користи за размену кључева. TLS спецификација подржава два алгоритма за размену кључева, а то су RSA и Дифи-Хелман. Иако се у самој поруци не налази податак о томе који се алгоритам за размену кључева користи, клијент и сервер имају ту информацију јер су се о томе претходно договорили.

На слици 3.14 је приказан формат поруке за случај када се користи RSA алгоритам за размену кључева. На слици се види да је тип поруке 16, након чега следи поље за дужину поруке. Тело поруке се састоји од шифроване *premaster_secret* вредности (видети тачку 3.6.2). *Premaster_secret* вредност је шифрована јавним кључем сервера и из ње се добија *master_secret* вредност (видети тачку 3.6.3) за тренутну сесију. *Premaster_secret* вредност се састоји од два бајта који представљају последњу верзију протокола TLS који клијент подржава и 46 бајтова којима је вредност изабрана на случајан начин.

Тип садржаја	***Бр. верзије***	***Бр. Подверзије***	***Дужина***
22	*3*	*1*	*...*
...	*Тип: 16*	*Дужина:...*	*Дужина:...*
Дужина:...	Шифрована премастер тајна		

Слика 3.14 Формат поруке ClientKeyExchange када се користи RSA алгоритам за размену кључева

Уколико се за размену кључева користи метод *DH_RSA* или метод *DH_DSS*, онда се од клијента захтева да пошаље свој сертификат. Клијент тада може да пошаље сертификат који садржи Дифи-Хелман јавни кључ чији параметри (група и генератор) одговарају параметрима јавног кључа који је примљен од сервера. У том случају порука *ClientKeyExchange* је празна, а у супротном садржи клијентов јавни кључ и његову дужину, као што је приказано на слици 3.15.

Тип садржаја	*Бр. верзије*	*Бр. Подверзије*	*Дужина*
22	*3*	*1*	*...*
...	*Тип: 16*	*Дужина:...*	*Дужина:...*
Дужина:...	Дужина DH Y		
Diffie-Hellman Y вредност			

Слика 3.15 Формат поруке ClientKeyExchange када се користи Дифи-Хелман алгоритам за размену кључева

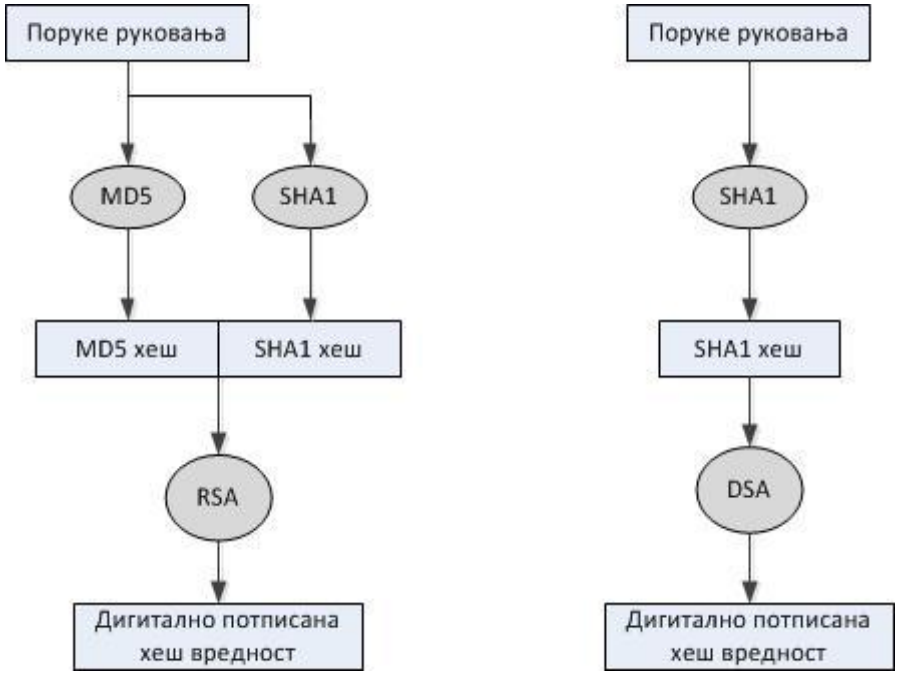
3.4.9 Порука *CertificateVerify*

Поруком *CertificateVerify*, клијент потврђује да има приватни кључ који одговара јавном кључу који се налази у сертификату који је послао серверу. На слици 3.16 се може видети да се порука састоји од дигитано потписане хеш вредности свих претходно размењених порука. Уколико се за потписивање користи алгоритам RSA, у телу поруке се налази дигитални потпис комбинације MD5 i SHA1 хеша. Уколико се за потписивање користи алгоритам DSA, потписује се само SHA1 хеш.

Тип садржаја	*Бр. верзије*	*Бр. Подверзије*	*Дужина*
22	*3*	*1*	*...*
...	*Тип: 15*	*Дужина:...*	*Дужина:...*
Дужина:...			
Дигитално потписана хеш вредност свих претходних порука			

Слика 3.16 Порука CertificateVerify

На слици 3.17 је представљен алгоритам којим се добија дигитално потписани хеш свих претходно размењених порука у случају када се користи RSA и када се користи алгоритам DSA за дигитално потписивање.



Слика 3.17 Дигитално потписана хеш вредност свих претходно размењених порука када се користи алгоритам RSA (лево) и када се користи алгоритам DSA (десно) за дигитално потписивање

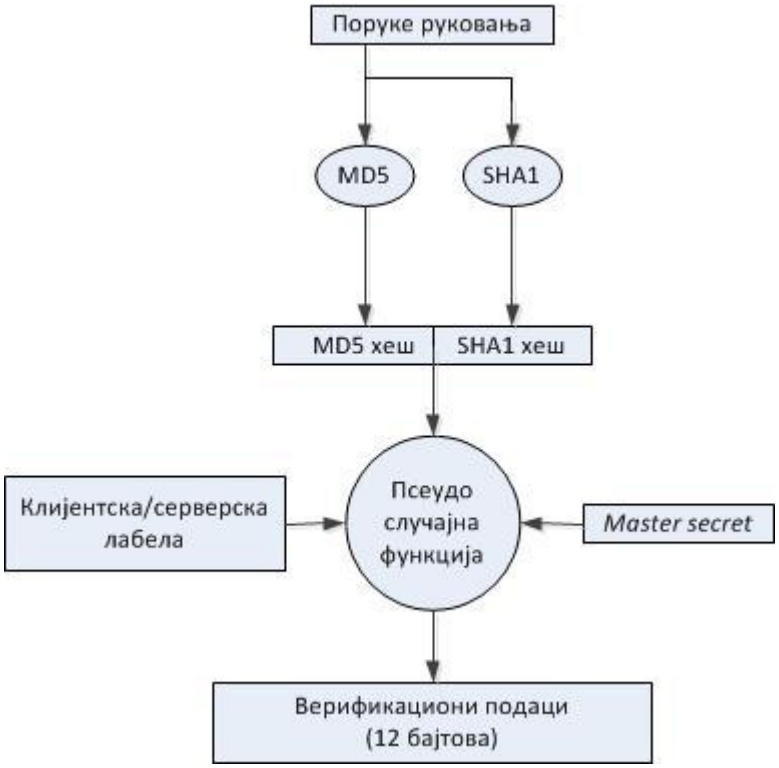
3.4.10 Порука *Finished*

Порука *Finished* је типа 20 и њоме се завршава процес преговора око сигурносних параметара и након што се она прими почиње се са применом сигурносних механизама који су договорени. Порука *Finished* је прва порука која је шифрована договореним сигурносним параметрима и на слици 3.18 је приказан њен формат.

Тип садржаја	***Бр. верзије***	***Бр. Подверзије***	***Дужина***
22	*3*	*1*	*0*
16	*Тип: 20*	*0*	*0*
12	Верификациони подаци (12 бајтова)		

Слика 3.18 Порука Finished

У поруци *Finished* се преносе 12 бајтова који се добијају као резултат псеудо случајне функције, где је улаз *master_secret* вредност (видети тачку 3.6.3), лабела *„client finished“* за клијента или *„server finished“* за сервер и MD5 и SHA1 хеш свих преходно размењених порука руковања. На слици 3.19 је приказано генерисање завршне поруке.



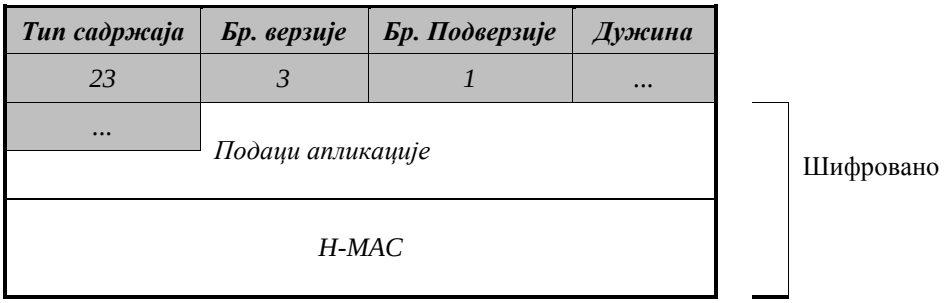
Слика 3.19 Генерисање поруке Finished

3.5 Заштита порука

Прва порука која користи сигурносне параметре договорене током процеса преговора је *Finished* порука. Свака порука која је послата након *Finished* поруке је шифрована и компримована сигурносним параметрима.

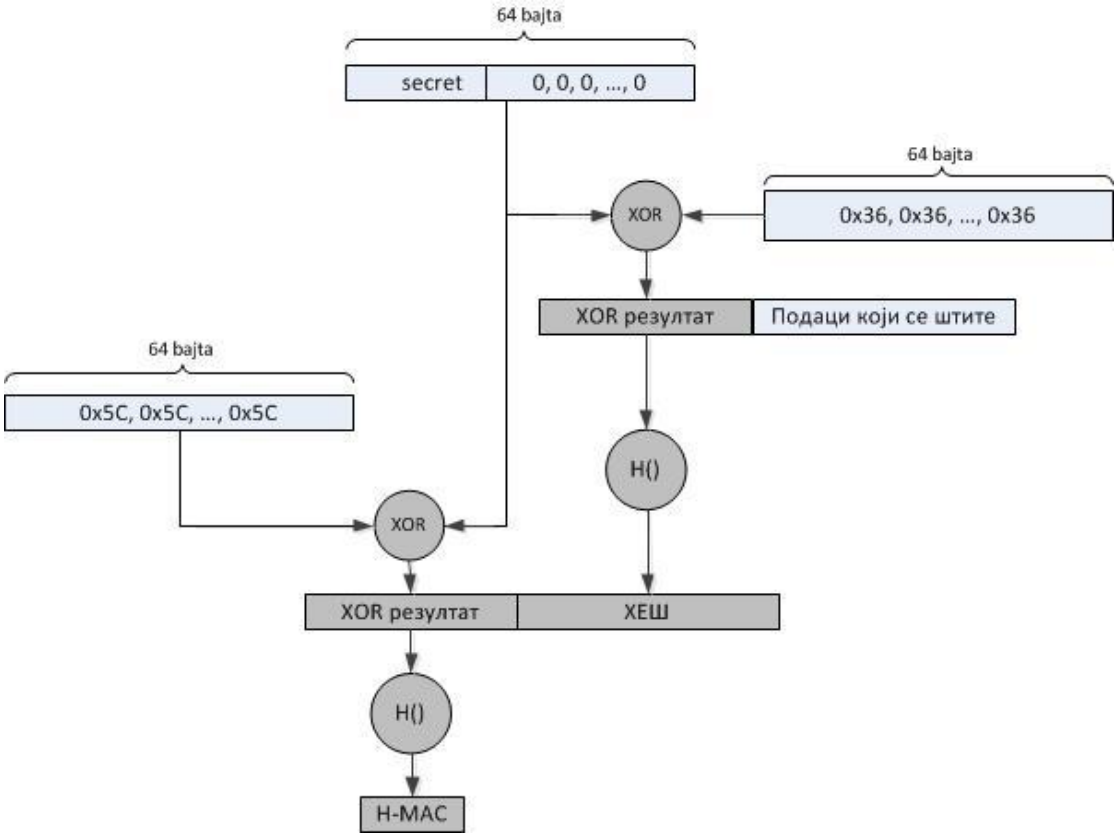
3.5.1 Аутентификациони код поруке

У протоколу TLS се аутентификациони код поруке рачуна користећи H-MAC (енг. *Hashed Message authentication code*). На податке апликације се пре шифровања додаје аутентификациони код, и на слици 3.20 се види да су шифровани и подаци апликације и одговарајући H-MAC.



Слика 3.20 Шифровање података апликације и аутентификационог кода

На слици 3.21 је представљен H-MAC алгоритам. Као што се, на истој слици, може видети он не користи ни један специфичан, већ може да користи било који компетентни хеш алгоритам.

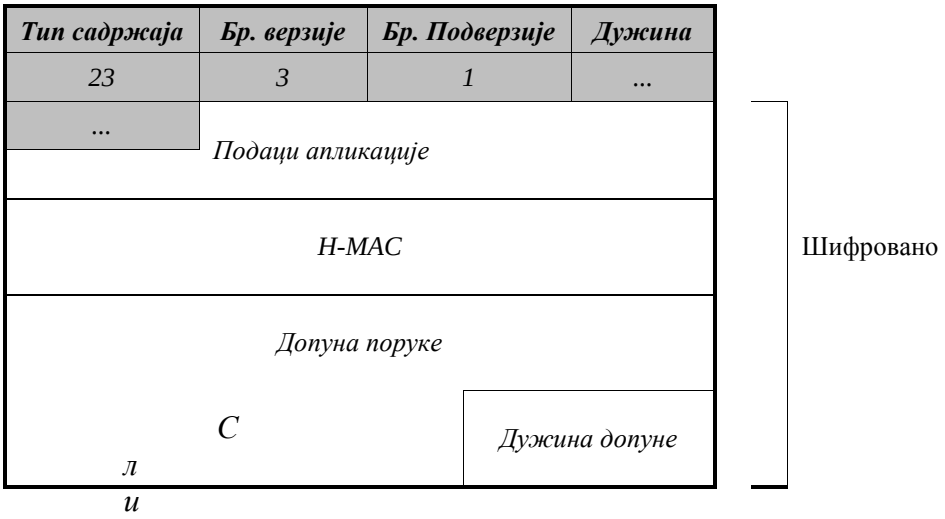


Слика 3.21 H-MAC алгоритам

3.5.2 Шифровање

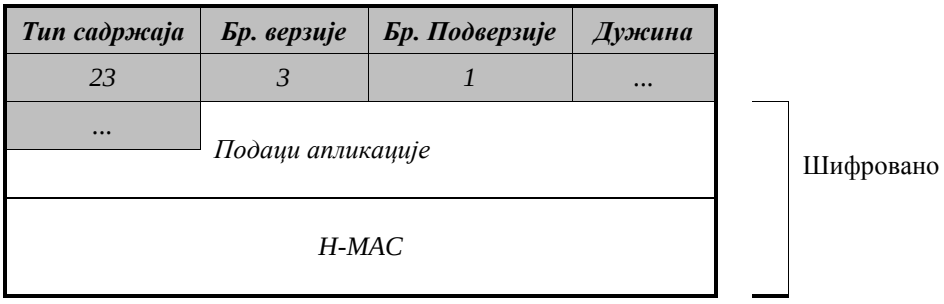
Протокол TLS подржава две врсте шифровања података, блоковско шифровање и ланчано шифровање. Формат поруке зависи од врсте шифровања.

У случају када се користи блоковско шифровање дужина података који се шифрују мора бити садржалац величине блока који се користи за шифровање. С обзиром да у пракси ово често није случај, подаци који се шифрују морају да се допуњују до одговарајуће дужине. Да би корисник који дешифрује поруку знао где се завршава порука, а где почиње допуна, по правилу последњи бајт допуне представља њену дужину. Након дешифровања поруке, корисник одузме од поруке онолико бајтова колика је вредност последњег бајта дешифроване поруке. Оно што је остало је порука и H-MAC вредност. На слици 3.22 је представљен формат поруке у случају када се користи блоковско шифровање.



3.22 Формат шифроване поруке када се користи блоковско шифровање

У случају када се користи ланчано шифровање, шифрује се само порука са додатим аутентификационим кодом, а формат такве поруке је приказан на слици 3.23.



Слика 3.23 Формат шифроване поруке када се користи ланчано шифровање

3.6 Креирање сигурносних параметара

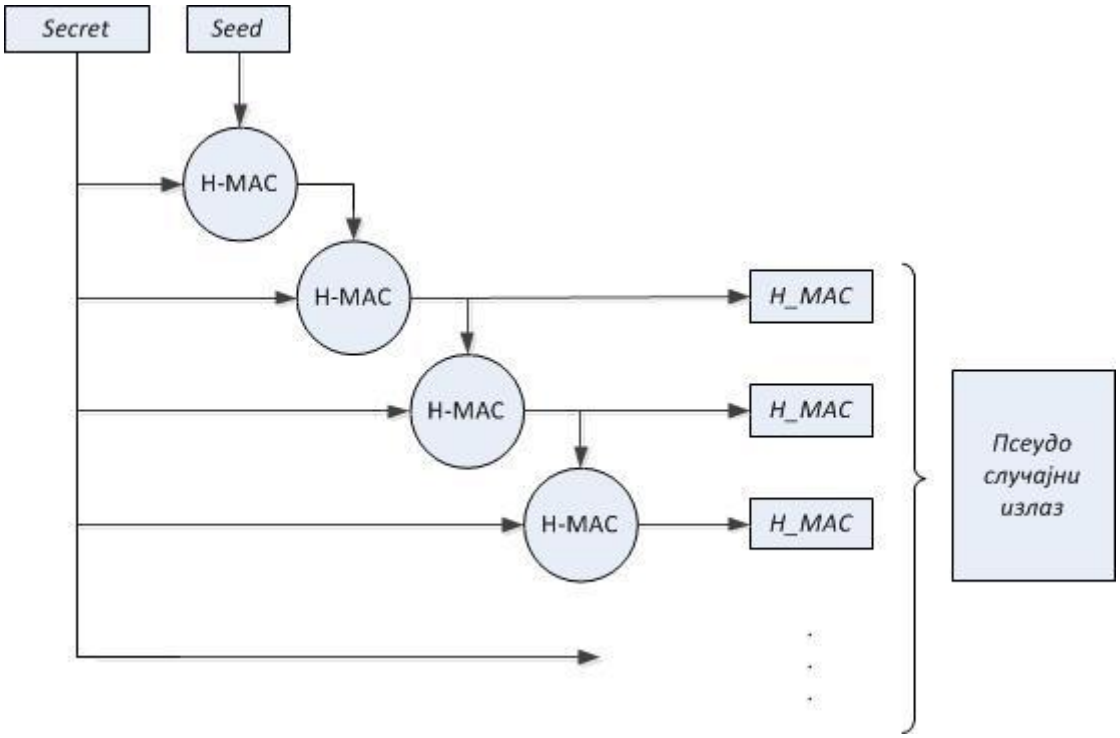
Алгоритми за шифровање и израчунавање хеша се базирају на скупу тајних података који су познати само учесницима комуникације. За размену тих информација на сигуран начин задужен је протокол руковања.

3.6.1 Псеудо случајна функција

За генерисање псеудо случајног излаза, у протоколу TLS, користи се H-MAC алгоритам. Да би се генерисао псеудо случајни излаз, на улазу се узима тајни податак (*secret*) и почетна вредност (seed). Користећи H-MAC алгоритам, на сигуран начин се може добити псеудослучајни низ произвољне дужине. Кораци за генерисање псеудо случајног излаза описани су у табели 3.6, док је на слици 3.24 графички приказан исти поступак.

Корак	Акција
1	Израчунава се H-MAC који на улазу узима тајни податак и почетну иницијалну вредност
2	Израчунава се H-MAC који на улазу узима тајни податак и излаз из претходног корака. Тако добијени резултат је први део псеудо случајног излаза
3	Израчунава се H_MAC који на улазу узима тајни податак и излаз из претходног корака: Тако добијени резултат је следећи део псеудо случајног излаза
4	Поступак се понавља све док се не добије псеудо случајни излаз довољне дужине

Табела 3.6 Поступак генерисања псеудо случајног излаза

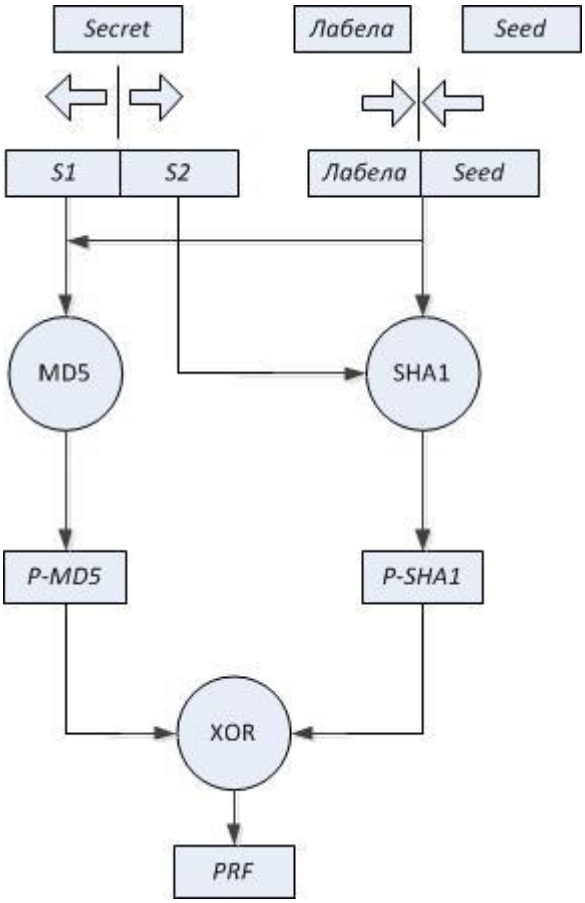


Слика 3.24 Поступак генерисања псеудо случајног излаза

Протокол TLS за генерисање кључа користи псеудо случајну функцију која се назива PRF. PRF комбинује два различита алгоритма за добијање излаза, MD5 и SHA1. Комбинација ова два алгоритма користи се јер се предпоставља да је комбинација сигурнија од сваког од њих појединачно.

На слици 3.25 представљен је поступак израчунавања вредности користећи псеудо случајну функцију. Улазни подаци су тајни податак *(secret)*, почетна вредност *(seed)* и лабела. *Secret* се дели на два једнака дела. Уколико се *secret* вредност састоји од непарног броја бајтова, средњи бајт се придружује и левој и десној страни. Затим се алгоритмом MD5 генерише псеудо случајна

вредност, користећи прву половину *secret* вредности и комбинацију лабеле и *seed* вредности. На исти начин се креира још једна псеудо случајна вредност користећи SHA1 алгоритам, други део *secret* вредности и комбинацију лабеле и *seed* вредности. Над резултатима претходне две операције се примени XOR операција. С обзиром да алгоритам MD5 као резултат даје 16 бајтова, а SHA1 20 бајтова, број итерација за добијање хеш вредности MD5 и SHA1 алгоритмом ће бити различит.



Слика 3.25 TLS псеудо случајна функција

3.6.2 Израчунавање *premaster_secret* вредности

Све тајне информације које деле клијент и сервер се израчунавају користећи *master_secret* вредност. *Master_secret* вредност се рачуна на основу *premaster_secret* вредности. Клијент најчешће *premaster_secret* вредност добија тако што изгенерише сигуран случајан број који само он зна. Након генерисања, клијент шифрује *premaster_secret* вредност јавним кључем који је добио од сервера и шаље га серверу у оквиру поруке *ClientKeyExchange* (ако се користи Дифи-Хелман алгоритам, резултат тог алгоритма се користи као *premaster_secret* вредност) и на тај начин се осигурава да оба саговорника знају *premaster_secret* вредност. У табели 3.7 је приказан поступак израчунавања *premaster_secret* вредности.

Алгоритам за размену кључева	Акција
RSA	Клијент генерише *premaster_secret* вредност као два бајта која садрже верзију протокола TLS (3 и 1), иза којих следи случајно генерисаних 46 бајтова.
Дифи-Хелман	Кључ генерисан Дифи-Хелман алгоритмом користи се као *premaster_secret* вредност.

Табела 3.7 Поступак израчунавања *premaster_secret* вредности

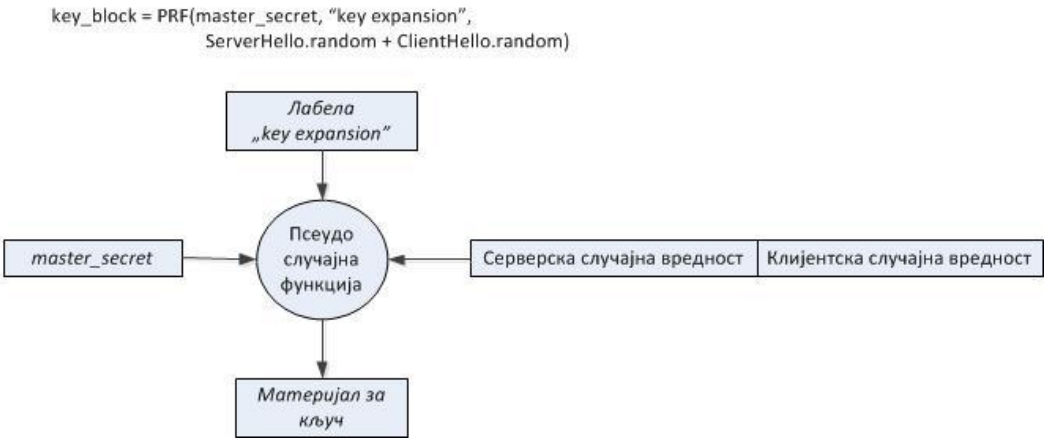
3.6.3 Израчунавање *master_secret* вредности

Након што сервер прими *premaster_secret* вредност, обе стране у комуникацији имају могућност да израчунају *master_secret*. *Master_secret* вредност се добија као резултат псеудо случајне функције, која на улазу прима *premaster secret* вредност као *secret* вредност, за лабелу се користи текст *„master secret“*, а *seed* је низ бајтова добијен спајањем случајног броја клијента и случајног броја сервера. На слици 3.26 је приказана израз, као и графички приказ поступка за израчунавање *master_secret* вредности.

Слика 3.26 Израчунавање *master_secret* вредности

3.6.4 Генерисање кључа

За генерисање кључа у протоколу TLS користи се псеудо случајна функција. У овом случају за улазне податке се користе *master_secret* као *secret* вредност, *“key expansion”* као лабела и конкатенација случајног броја сервера и случајног броја клијента као *seed*. На слици 3.27 је приказана израз, као и графички приказ поступка за израчунавање материјала за кључ.



Слика 3.27 Израчунавање материјала за кључ

Претходно описани алгоритам се понавља све док се не скупи довољно материјала за кључ. Када се сакупи довољно дугачак низ бајтова као материјал за кључ, тај низ се разлаже на следећи начин:

client_write_MAC_secret[hash_size]
server_write_MAC_secret[hash_size]
client_write_key[key_material_length]
server_write_key[key_material_length]
client_write_InitializationVector[InitializationVector_size]
server_write_InitializationVector [InitializationVector_size]

где дужине *hash_size*, *key_material_length* и *InitializationVector_size* зависе од шифарског система који се користи. Нпр. уколико се користи шифарски систем SSL_RSA_WITH_3DES_EDE_CBC_SHA, вредност за *hash_size* је 20, за *key_maerial_length* је 24, а за *InitializationVector_size* је 8.

4. Слабости протокола TLS

Све већи број апликација које користе протокол TLS отвара могућност откривања и коришћења сигурносних пропуста у протоколу TLS, а који би могли да утичу на сигурност података који се штите. Порастом броја корисника интернета, расте и број корисника електронских трансакција. Протокол TLS је развијен за потребе сигурности у веб апликацијама које раде са поверљивим подацима и веома је једноставан за имплементацију. Као и многе друге апликације, ни протокол TLS није у потпуности сигуран. Он је дизајниран да осигура поверљивост, интегритет и проверу идентитета. Део протокола TLS који обезбеђује поверљивост омогућује злонамерној особи да пресретне осетљиве информације у тренутку слања, али онемогућава њихово дешифровање. Део који проверава идентитет је дизајниран тако да спречи могућност нападом „човек у средини“, где се злонамерна особа представља као сервер и на тај начин жели да дође до поверљивих информација. Иако је део за проверу идентитета добро имплементиран кроз сертификате, постоји могућност да протокол TLS постане жртва напада.

4.1 Заблуде и претпоставке корисника

Велики проблем сигурности протокола TLS проузрокован је претераним поверењем које корисници имају у сам протокол. Како је електронска трговина постала све учесталија у свакодневном животу људи, тако је створен и лажни осећај сигурности. У почетку су људи били неповерљиви према сигурности на интернету и у то време просечни корисник не би слао свој број кредитне картице или неку другу осетљиву информацију преко Интернета. Како се број трансакција преко Интернета повећавао, тако је расла и самоувереност људи у електронске трансакције. Сам поглед на катанац у веб прегледачу ствара у људима лажан осећај сигурности. Ова претпоставка значи да је корисник уверен:

- да протокол TLS није рањив
- да је протокол исправно имплементиран у веб прегледач
- и да је идентификација сертификата у потпуности проверена на веб страницама пре њихове примене

Постоји и велики број корисника који не поседују довољно знање и осећај о поверљивости одређених података. Узмимо за пример када веб страница користи самопотписане сертификате, тј. сертификате који нису одобрени од овлашћених организација. Када веб прегледач, при успостављању TLS везе, установи да серверски сертификат није одобрен ни од једне овлашћене организације, он ће од корисника тражити потврду да жели да настави везу. У случају када корисник одговори потврдно, веб прегледач наставља повезивање што представља велики ризик. Већина корисника који не разумеју ову поруку одговара са „ОК“ чиме отварају могућност злоупотребе својих поверљивих информација.

4.2 Листа овлашћених организација за издавање сертификата

Велика мана имплементације протокола TLS је недостатак адекватне заштите листе овлашћених организација за оверу и издавање сертификата, коју веб прегледач сматра поверљивом и сигурном. Уколико нека малициозна апликација, као што је вирус, црв или тројанац, измени листу тако што у њу дода одређени сертификат, отвара се могућност за напад типа „Човек у средини“.

4.3 Напад типа „Човек у средини”

Напад типа „Човек у средини“ се дешава када се злонамерни сервер постави између корисника и жељеног сервера. Корисник уместо жељеном серверу свој захтев пошаље злонамерном серверу и са њим неопрезно успоставља TLS сесију. Злонамерни сервер, у исто време ка правом серверу шаље свој захтев и на тај начин са њим започиње TLS сесију. Злонамерни сервер сада посредује у комуникацији између корисника и правог сервера, и с обзиром да их може дешифровати у могућности је да види све поверљиве информације које корисник шаље. Ни корисник, ни прави сервер у овом случају неће приметити злонамерну страну, с обзиром да обојица шаљу и примају захтеве кроз „исправну“ TLS везу.

4.4 Напад типа „Отказ сервиса“

Напад типа „Отказ сервиса“ (енг. *denial-of-service*) за циљ има да се нека машина или мрежни ресурс учини недоступним за кориснике. Иако средства за извршавање, мотиви, али и мете напада могу да се разликују, овај напад се углавном заснива на напорима да се трајно или привремено обуставе услуге које пружа мета напада.

Починиоци напада типа „Отказ сервиса“ обично циљају сајтове или услуге хостоване на истакнутим веб серверима као што су банке, сервери који служе за плаћање кредитним картицама, и сл. Један уобичајен метод напада укључује затрпавање мете захтевима у толикој мери да мета напада није у стању да одговори на легитимне захтеве корисника. Овакви напади обично доводе до преоптерећења сервера. Иако за све познате нападе типа „Отказ сервиса“ постоје софтверске заштите које администратори могу да примене да би ограничили штету која може да настане овим нападима, свакодневно се појављују нове врсте ових напада

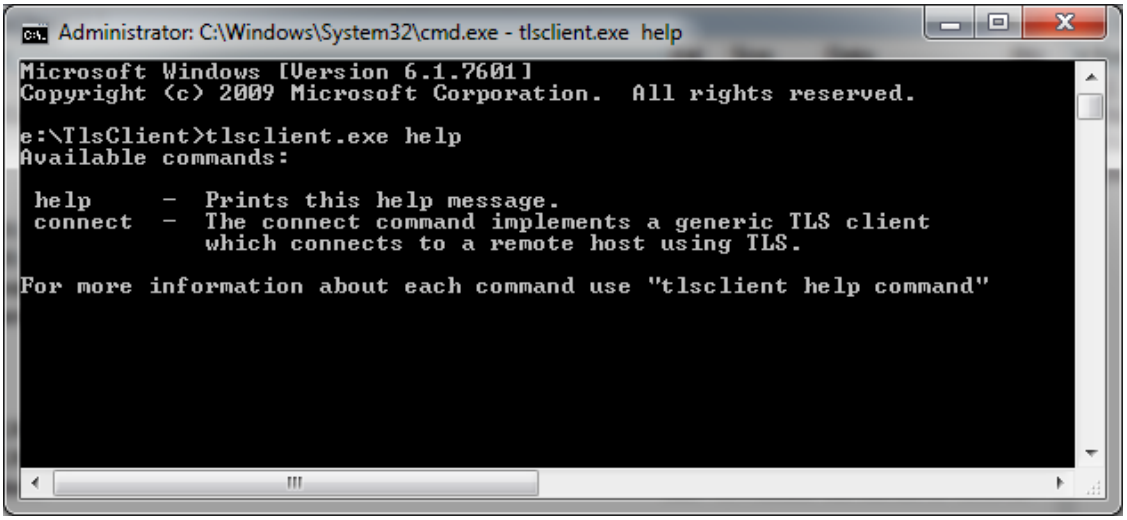
5. Практичан рад

У оквиру практичног рада имплементиран је протокол TLS и креирана је тест апликација *TLSClient* која користи имплементирани протокол TLS 1.0.

5.1 Апликација „TLSClient”

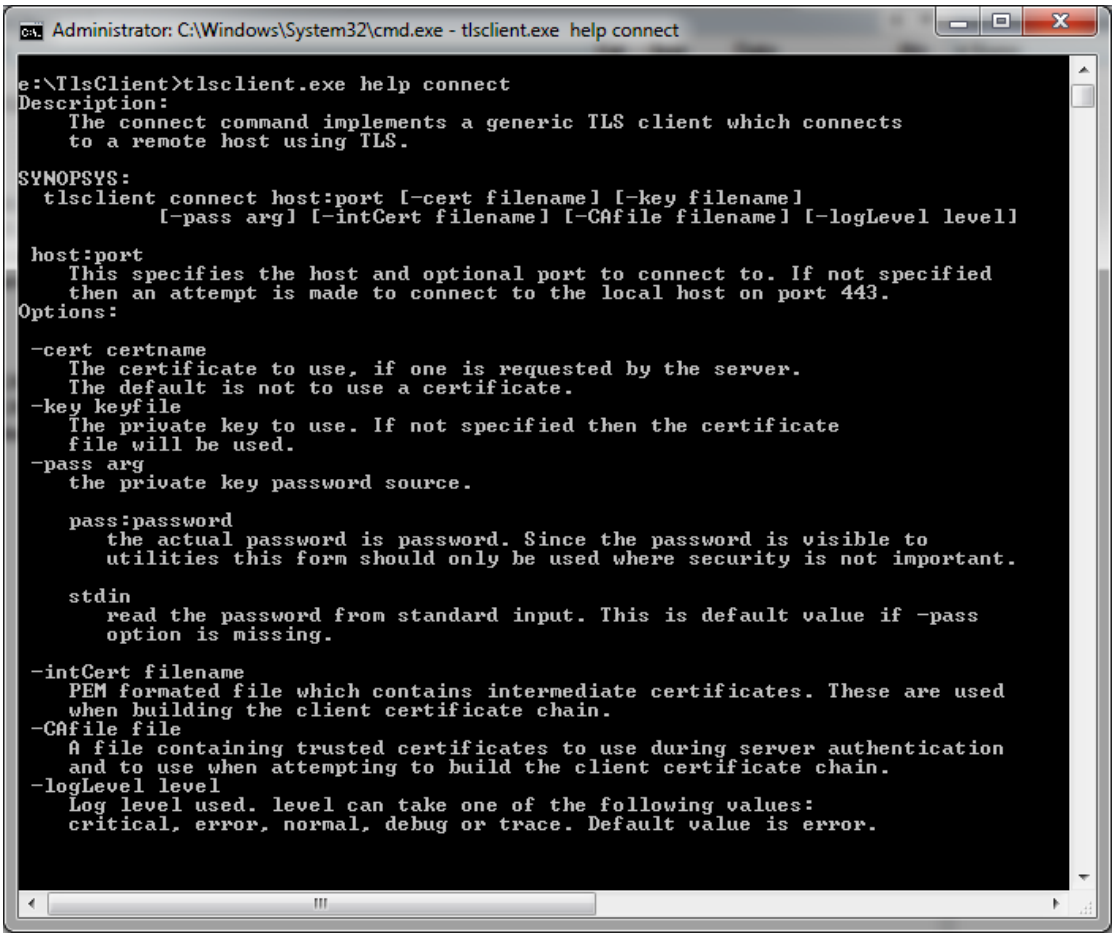
Апликација *TLSClient* служи за успостављање шифроване комуникације са било којим сервером који подржава протокол TLS 1.0. *TLSClient* се покреће из командне линије тако што се наведе назив апликације, а затим и одговарајуће команде и параметри које апликација користи. На слици 5.1 се могу видети команде које апликација подржава.

Командом *help* се излиставају команде које су подржане у апликацији, док се командом *connect* започиње успостављање шифроване комуникације.



Слика 5.1 Команде апликације *TLSClient*

Команда *connect* има и додатне параметре који су потребни приликом успостављања шифроване комуникације, а њихов списак се може добити уколико се након команде *help* навоеде и команда *connect*, што је приказано на слици 5.2.



Слика 5.2 Параметри команде *connect*

Након команде *connect* обавезно мора да се наведе IP адреса сервера са којим се успоставља шифрована комуникација и по жељи се може навести и TCP порт преко кога сервер прима захтеве клијента. Уколико се TCP порт не наведе, клијент ће користити TCP порт 4433 за комуникацију са сервером. IP адреса сервера може бити адреса у текстуалном облику нпр. www.google.com или по жељи може да се користи IP адреса у нумеричком облику нпр. 74.125.239.19.

Остали параметри команде *connect* су опциони, а њихово значење је описано даље у тексту:

-cert certname:

Параметром *–cert* се задаје путања до фајла који садржи клијентски сертификат који ће бити прослеђен серверу уколико се то од клијента захтева. У случају да се овај параметар не наведе, а сервер од клијента затражи сертификат, шифрована комуникација неће бити успостављена.

-key keyfile:

Параметром *–key* се задаје путања до фајла који садржи приватни кључ клијентског сертификата. Уколико се не наведе путања до овог фајла, подразумева се да се приватни кључ налази у истом фајлу где и клијентски сертификат.

-pass arg:

Параметром *–pass* се задаје шифра која је потребна да би се користио приватни кључ клијентског сертификата и она се може задати на два начина:

1) ***–pass pass:password***
   На овај начин се шифра задаје директно у командној линији на месту где се налази реч *password.*
2) ***–pass stdin***
   Уколико се изабере ова опција, шифра се уонси у командној линији када за њом буде потребе. Уколико се *–pass* параметар не наведе, ово ће бити подразумевано понашање апликације.

-intCert filename:

Параметром *–intCert* се задаје путања до фајла у *PEM*[1]формату, а који садржи ланац сертификата којим је клијентски сертификат повезан са CA сертификатом које је издало аутентификационо тело.

-CAFile filename:

Параметром *–CAFile* се задаје путања до фајла у *PEM*[1] формату, који садржи *Root CA* (*CA, Certificate Authority)* сертификате којима се верује.

-logLevel level:

Параметром –logLevel се подешава количина порука које ће тест апликација да исписује.

1) ***critical***
   Када се изабере *critical* ниво логовоања, биће исписани само називи порука који су размењени између клијента и сервера.
2) ***error***
   Када се изабере *error* ниво логовања, поред назива порука, биће исписане и евентуалне грешке до којих дође приликом успостављања шифроване комуникације.
3) ***normal***
   Када се изабере *normal* ниво логовања, логовима се још додаје и садржај размењених порука у хексадецималном облику, као и тумачење садржаја тих порука.
4) ***debug***
   У *debug* нивоу логовања, логовима се додају и неки детаљи везани за обраду пакета добијених од сервера, као и детаљи о креирању пакета који се шаљу серверу.
5) ***trace***
   Када се изабере *trace* ниво логовања, логовима се додају и детаљи везани за коришћење сертификата и кључева, са садржајем свих бафера у хексадецималном облику који у томе учествују.

[1] PEM (Privacy Enhanced Mail) формат је један од првих стандарда за заштиту електронске поште. Иако његова оригинална намена никада није заживела, у данашње време често се користи за шифровање сертификата. У том случају сертификат шифрован PEM протоколом се налази између граничника -----BEGIN CERTIFICATE----- и -----END CERTIFICATE-----.

5.2 Алат OpenSSL

За потребе тестирања апликације *TLSClient* коришћена је апликација *OpenSSL*. *OpenSSL* је апликација која садржи разне криптпграфске алате (алати за креирање кључева и сертификата, алате за потписивање, шифровање и дешифровање, ...), као и имплементацију протокола *SSL* и *TLS*.

С обзиром да је апликација *TLSClient* имплементација клијентске стране протокола *TLS, OpenSSL* се користи за постављање сервера са којим апликација *TLSClient* успоставља шифровану комуникацију. Поред тога *OpenSSL* се користи за креирање сертификата и њихових јавних и приватних кључева које ће сервер и клијент користити приликом успостављања шифроване комуникације.

5.2.1 Креирање „*Root CA*“ сертификата

Инфраструктура система са јавним кључем (PKI, *Public Key Infrastructure)* је аранжман који везује јавне кључеве са одговарајућим корисничким идентитетима, користећи институцију аутехтификационог тела *(CA, Certificate Authority).* Другим речима омогућује корисницима мреже да утврде идентитет осталих учесника на основу њихових јавних кључева. Идентитет се утврђује провером да ли је сертификат са јавним кључем чији се идентитет проверава, издат од стране неког аутентификационог тела коме се верује.

Root CA сертификат је непотписани сертификат који садржи јавни кључ или самопотписани сертификат којим се идентификује аутентикационо тело (*CA, Certificate Authority)* овлашћено за издавање сертификата.

Да би се алатом *OpenSSL* креирао *Root CA* сертификат, потребно је прво креирати његов приватни кључ командом:

openssl genrsa –des3 –out ca.key 4096

На овај начин креира се *RSA* приватни кључ дужине 4096 бита који је шифрован *DES3* алгоритмом и сачуван у фајлу под именом „*ca.key“*. Приликом креирања приватног кључа *OpenSSL* ће тражити да се унесе шифра којом ће тај кључ бити заштићен.

Јавни кључ који је основни део самог *Root CA* сертификата креира се командом:

openssl req –new –x509 –days 365 –key ca.key –out ca.crt

Након уношења шифре приватног кључа креираног у претходном кораку и разних података (назив земље, града, радне организације, ...) везаних за сертификат који се креира биће креиран самопотписани сертификат *“ca.crt”* чији јавни кључ одговара приватном кључу „*ca.key*“ са роком важења од 365 дана.

5.2.2 Издавање „*Root CA" потписаног* сертификата

Да би се алатом *OpenSSL* креирао сертификат који је потписан од стране *Root CA* сертификата потребно је креирати његов приватни кључ, а затим и сам сертификат. Када је сертификат креиран, потребно је још само потписати га *Root CA* сертификатом.

Приватни кључ се креира командом:

openssl genrsa –des3 –out signed.key 4096

Затим се коритећи управо креирани приватни кључ креира сертификат командом:

openssl req –new –key signed.key –out signed.csr

И на крају се креирани сертификат потписује и чува у фајл са називом *signed.crt* командом:

openssl x509 –req –days 365 –in server.csr –CA ca.crt –CAkey ca.key –set_serial 01 –out signed.crt

Сертификати и кључеви креирани на овај начин користе се при успостављању шифроване комуникације између сервера и клијента.

5.2.3 Покретање *TLS* сервера

Уграђена команда *s_server* алата *OpenSSL* покреће *TLS* сервер који чека на захтеве за успостављање шифроване конекције од стране клијената. Команда *s_server* има велики број параметара којим се сервер може конфигурисати и у наставку текста ће бити објашњени параметри који су важни за тестирање апликације *TLSClient.*

Команда *s_client* се користи наводећи назив команде за којим следе параметри који су опциони.

openssl s_server [**-accept port**] [**-verify depth**] [**-Verify depth**] [**-cert filename**] [**-certform DER**|**PEM**] [**-key keyfile**] [**-keyform DER**|**PEM**] [**-pass arg**] [**-debug**] [**-msg**] [**-state**] [**-CApath directory**] [**-CAfile filename**] [**-cipher cipherlist**] [**-ssl2**] [**-ssl3**] [**-tls1**] [**-no_ssl2**] [**-no_ssl3**] [**-no_tls1**] [**-tlsextdebug**]

-accept port
TCP порт на коме се чека на захтев од клијента. Уколико се ова опција не наведе сервер ће на захтеве чекати на TCP порту 4433.

-cert certname
Серверски сертификат који ће се користити приликом успостављања шифроване комуникације.

-certform format

Формат серверског сертификата DER или PEM. Уколико се ништа не наведе подразумева се да је серверски сертификат у PEM формату. DER опција се користи када је сертификат који се задаје у ASN1 DER формату. То значи да је сертификат задат бинарном репрезентацијом структуре која га одређује.

-key keyfile

Приватни кључ серверског сертификата. Уколико се не наведе подразумева се да се приватни кључ налази у истом фајлу где и серверски сертикат.

-keyform format

Формат приватног кључа серверског сертификата DER или PEM. Уколико се ништа не наведе подразумева се да је задати кључ у PEM формату.

-pass arg

Параметром *–pass* се задаје шифра која је потребна да би се користио приватни кључ клијентског сертификата и она се може задати на два начина:

3) ***–pass pass:password***

   На овај начин се шифра задаје директно у командној линији на месту где се налази реч *password.*

4) ***–pass stdin***

   Уколико се изабере ова опција, шифра се уноси у командној линији када за њом буде потребе. Уколико се *–pass* параметар не наведе, ово ће бити подразумевано понашање апликације.

-verify depth, -Verify depth

Овим се сервер подешава да од клијента тражи да пошаље свој сертификат, као и максимална дужина ланца клијентских сертификата која је прихватљива. Уколико се наведе опција **-verify** сервер ће од клијента тражити сертификат, али ће се успостављање шифроване комуникације наставити чак и уколико клијент не пошаље свој сертификат, а уколико се наведе опција **-Verify** клијент мора серверу да пошаље свој сертификат иначе ће доћи до грешке и шифрована комуникација неће бити успостављена.

-CAfile file

Root CA сертификат који се користи да би се верификовао сертификат добијен од клијента.

-state

Када се ова опција наведе, на стандардном излазу ће се исписивати стања TLS сесије.

-debug

Када се ова опција наведе, на стандардном излазу се исписују информације битне за успостављање шифроване комуникације, а те информације укључују и хексадецимални испис меморије свих пакета који се размењују.

-msg

Исписују се све поруке протокола *TLS.*

-ssl2, -ssl3, -tls1, -no_ssl2, -no_ssl3, -no_tls1

Овим опцијама се укључују или искључују поједини *SSL* или *TLS* протоколи. Ако се ништа од ових параметара не наведе, сервер је у могућности да, по потреби, користи било који од наведених протокола.

-cipher cipherlist

Овом опцијом се наводи који су све шифарски системи подржани на серверу. Уколико се као вредност ове опције наведе реч ALL, сервер се подешава да поржи све постојеће шифарске системе.

5.3 Примери успостављања шифроване комуникације употребом апликације TLSClient

У наставку текста ће бити приказано коришћење апликације *TLSClient* за успостављање шифроване комуникације са неколико различито подешених сервера. У неким случајевима пре покретања сервера и клијента потребно је креирати и сертификате који ће се користити у процесу успостављања сигурносне комуникације.

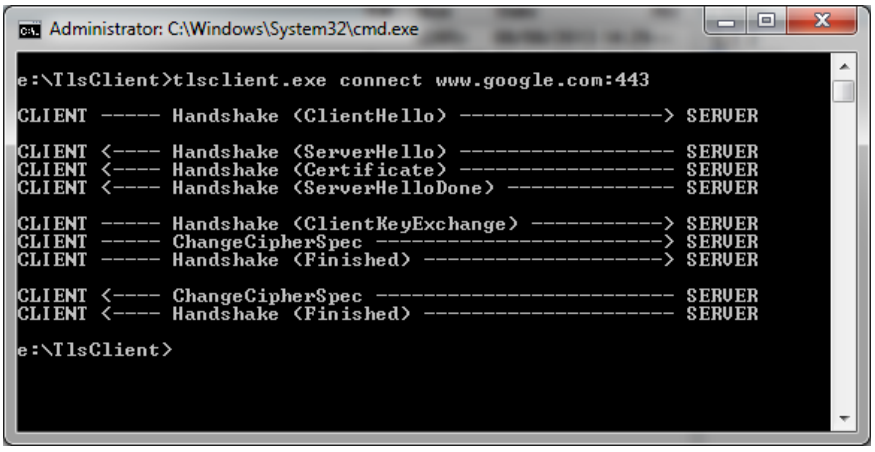
У случајевима где је то потребно подразумева се да сви потребни сертификати постоје и да се налазе у директоријуму из кога се покрећу команде за стартовање сервера, односно клијента.

5.3.1 Успостављање шифроване комуникације између апликације *TLSClient* и сервера *www.google.com*

Да би се успоставила шифрована комуникација са сервером *www.google.com,* потребно је да се у *Windows Command Prompt* прозору унесе команда:

***tlsclient.exe* connect www.google.com:443**

Након покретања команде на стандардном излазу ће се исписати пакети који су размењени приликом успостављања шифроване комуникације, као што је приказано на слици 5.3.



Слика 5.3 Успостављање шифроване комуникације између апликације *TLSClient* и сервера *www.google.com*

5.3.2 Успостављање шифроване комуникације са провером серверског сертификата – успешан случај

У претходном случају клијент није проверавао да ли је сертификат добијен од сервера издат од стране тела за издавање сертификата коме се верује. У овом примеру, клијент ће проверавати да ли је серверски сертификат *srv.crt* издат од стране тела за издавање сертификата чији се јавни кључ налази у сертификату *SrvCa.crt.*

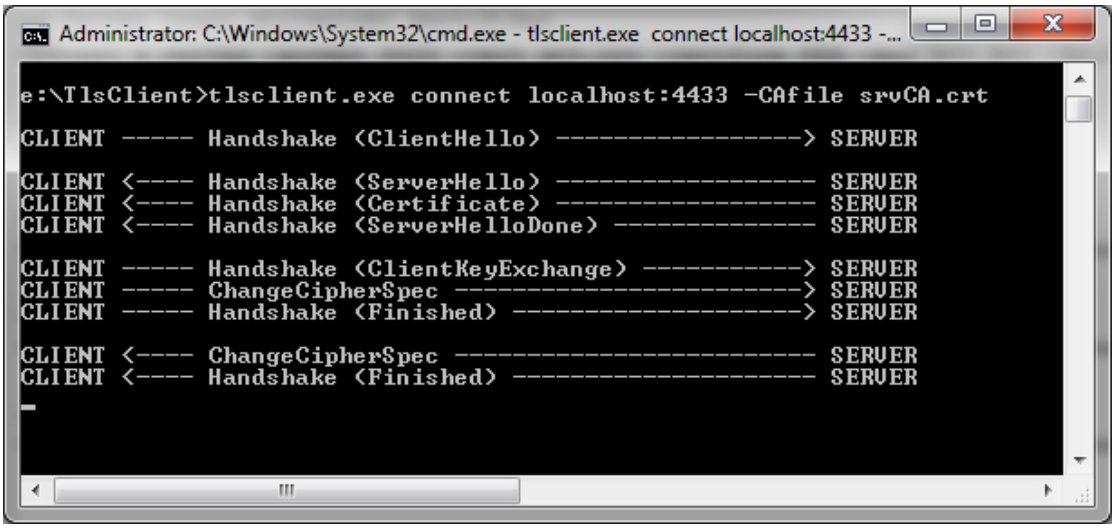
Да би се контролисао сертификат који ће сервер слати приликом успостављања шифроване комуникације, сервер се покреће алатом *OpenSSL* навођењем следеће команде:

openssl s_server -accept 4433 -cert srv.crt -certform PEM -key srv.key -keyform PEM -pass pass:qwerty -debug -tls1 -cipher ALL

Клијент се у овом случају покреће навођењем параметра *–CAfile* којим се задаје путања до фајла у PEM формату који садржи јавне кључеве тела за издавање сертификата којима се верује.

tlsclient.exe connect localhost:4433 -CAfile SrvCA.crt

Након покретања команде на стандардном излазу ће се исписати пакети који су размењени приликом успостављања шифроване комуникације, што је приказано на слици 5.4.



Слика 5.4 Успостављање шифроване комуникације са провером серверског сертификата – успешан случај

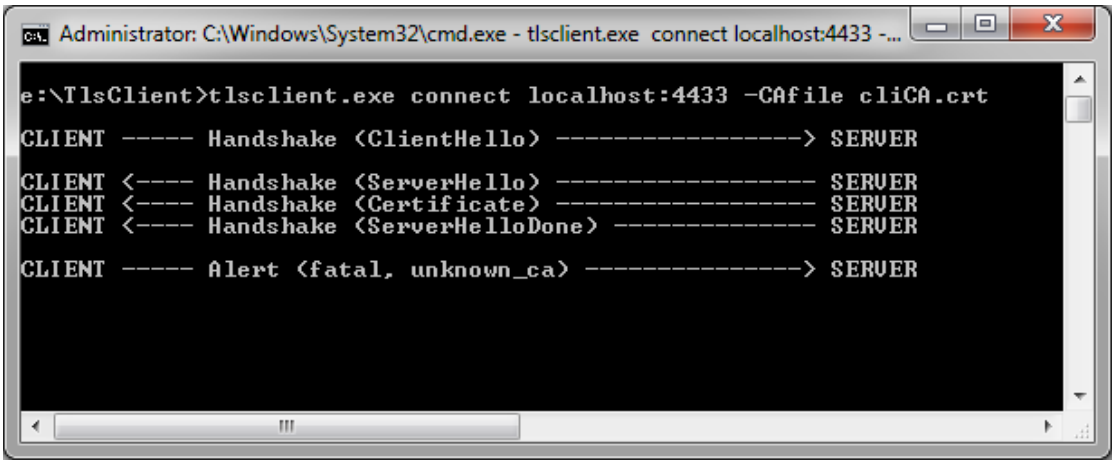
5.3.3 Успостављање шифроване комуникације са провером серверског сертификата - неуспешан случај

Уколико се клијент подеси тако да проверава серверски сертификат, а сервер пошаље сертификат који није издат ни од једног тела за издавање сертификата којима се верује, клијент ће зауставити успостављање шифроване комуникације обавештавајући сервер о томе поруком узбуне.

Сервер се покреће на исти начин као и у претходном случају док се клијент покреће тако што му се зада други фајл са јавним кључевима тела за издавање сертификата којима се верује, а у коме нема тела које је издало сертификат добијен од сервера.

tlsclient.exe connect localhost:4433 -CAfile CliCA.crt

Поруке размењене приликом успостављања шифроване комуникације у овом случају су приказане на слици 5.5.



Слика 5.5 Успостављање шифроване комуникације са провером серверског сертификата – неуспешан случај

5.3.4 Успостављање шифроване комуникације са слањем клијентског сертификата

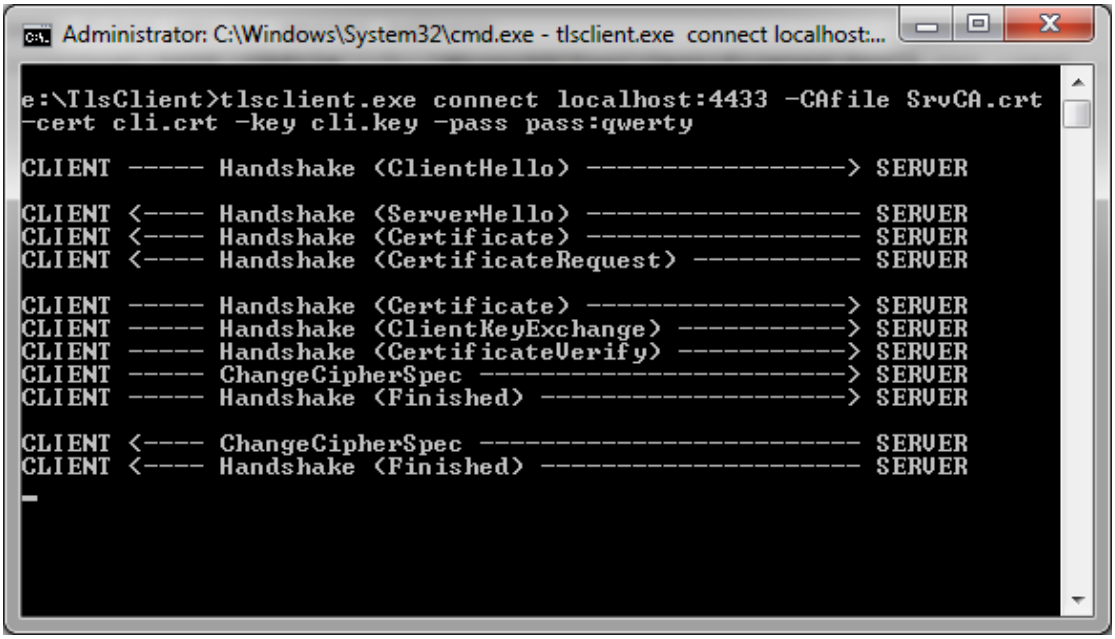
Уколико се сервер подеси да захтева од клијента да му пошаље свој сертификат, клијент ће то морати да уради да би се успоставила шифрована комуникација између те две стране. У овом случају сервер се покреће следећом командом:

openssl s_server -accept 4433 -cert srv.crt -certform PEM -key srv.key -keyform PEM -pass pass:qwerty -Verify 5 -CAfile srvCA.crt -debug -tls1 -cipher ALL

Док се клијент покреће командом:

tlsclient.exe connect localhost:4433 -CAfile SrvCA.crt -cert cli.crt -key cli.key -pass pass:qwerty

Поруке размењене приликом успостављања шифроване комуникације у случају када и клијент серверу шаље свој сертификат су приказане на слици 5.6.



Слика 5.6 Успостављање шифроване комуникације са слањем клијентског сертификата

5.4 Детаљи имплементације апликације TLSClient

Апликација *TLSClient* је креирана користећи развојно окружење Microsoft *Visual Studio 2010*. Отварањем решења *TLSClient.sln* у *Solution Explorer* погледу могу се видети фајлови који садрже изворни код. Сви наведени фајлови осим *tomcrypt* и *tommath* (*.c* и *.h*) фајлова су развијени за потребе овог мастер рада.

Назив фајла	Улога у пројекту
TLSClient.cpp	Имплементација *TLSClient* апликације. Садржи главну функцију за покретање апликације, као и помоћне функције за читање из датотеке и повезивање на сервер користећи *Socket* објекте.
TLSPacketList.cpp	Садржи функције за парсирање порука протокола TLS и њихово исписивање у логу апликације.
Logging.cpp	Садржи функције које служе за креирање логова апликације.
MicroSSL.cpp	Омотач око класа које су имплементиране у фајловима TLS.cpp, RSAKey.cpp, Certificate.cpp и CertStore.cpp.
TLS.cpp	Имплементација протокола TLS. Садржи функције за генерисање пакета који се шаљу серверу, за парсирање пакета добијених од сервера, као и функције за креирање сигурносних параметара и генерисање кључева. Принципи по којима су имплементиране поменуте функције објашњени су раније у овом документу.
RSAKey.cpp	Имплементација класе која служи за рад са *RSA* кључевима.
Certificate.cpp	Имплементација класе која служи за рад сертификатима.
CertStore.cpp	Имплементација класе која служи за креирање скупа сертификата, додавање новог сертификата у скуп, као и добијање постојећег сертификата из скупа.
tomcrypt.c, tommath.c	Изворни код *TomCrypt* алата за развој апликација. Ови фајлови су добијени конкатенацијом важних фајлова из *LibTomCrypt* и *LibTomMath* пројеката.

Табела 5.1 Фајлови који садрже изворни код пројетка *TLSClient*

5.4.1 LibTomCrypt пројекат

LibTomCrypt је свеобухватан криптографски алат који омогућава програмерима рад са мноштвом познатих блоковских шифара, једносмерних хеш функција, псеудо-случајних бројева, система са јавним кључевима и обиљем других рутина.

LibTomCrypt је од самог почетка дизајниран да буде веома једноставан за употребу. Он има модуларан и стандардан интерфејс који омогућава да се нове шифре, хешеви и псеудо рандом генератори додају или стари уклањају на веома једноставан начин и без утицаја на остале делове алата.

Изборни код алата *LibTomCrypt* је доступан јавности и слободан је за коришћење чак и у комерцијалне сврхе.

У имплементацији *TLSClient* апликације, *LibTomCrypt* се користи:

- у класи *TLS* за израчунавање хеш вредности у функцијама за креирање сигурносних параметара;
- у класи *RSAKey* за рад са системима са јавним и приватним кључевима;
- у класи *Certificate* за парсирање сертификата и операције над њима.

5.4.2 Класа Certificate

Класа *Certificate* је задужена за парсирање сертификата који су задати у *PEM* или *DER* формату, као и за обављање основних операција са сертификатима. Неке од тих операција су:

- Провера исправности сертификата на основу његовог времена важења;
- Провера издаваоца сертификата;
- Добијање јавног кључа сертификата;
- Потписивање приватним кључем сертификата, као и провера тог потписа;
- Шифровање и дешифровање користећи јавни и приватни кључ сертификата;

Класа *Certificate* се користи у фази успостављања шифроване комуникације за шифровање порука јавним кључем сервера, као и за потписивање поруке *CertificateVerify* приватним кључем клијента уколико постоји потреба да се ова порука шаље. Такође се користи за утврђивање интегритета сервера, провером да ли је сертификат или ланац сертификата који је примљен од сервера издат од сертификационог тела коме се верује.

5.4.3 Класа *CertStore*

Класа *CertStore* служи да креира листу сертификационих тела којима се верује и да омогући проверу да ли је задати сертификат издат од једног од сертификата из те листе.

Основне функције које су имплементиране у класи *CertStore* су:

- Додавање новог сертификационог тела у листу;
- Брисање листе сертификата;
- Дохватање првог сертификата из листе;
- На основу тренутног, дохватање следећег сертификата из листе;
- Провера да ли је дати сертификат издат од стране једног од сертификационих тела којима се верује, а који се налази у листи;

5.4.4 Класа RSAKey

Класа *RSAKey* имплементира основне операције са јавним и приватним *RSA* кључевима као што су потписивање, провера потписа, шифровање и дешифровање.

5.4.5 Класа *TLS*

Класа *TLS* је, могло би се рећи, најважнија класа развијена за потребе овог мастер рада јер представља имплементацију протокола *TLSv1.0*. Најбитније функције које су имплементиране у класи TLS су:

- ***BeginTlsHandshake*** – Креира *Handshake* поруку типа *ClientHello* која се шаље серверу да би се започело успостављање шифроване комуникације;
- ***ContinueTlsHandshake*** – Парсира поруке добијене од сервера, а затим једну по једну обрађује. Обрада порука зависи од фазе преговора, а разликују се случајеви када се обрађују поруке добијене од сервера пре или после слања *Finished* поруке. Уколико се обрађују поруке које су стигле пре слања *Finished* поруке, генеришу се одговарајући одговори серверу, у супротном се очекује *Finished* порука од сервера, и уколико он стигне обавља се верификација те врсте поруке;
- ***VerifyServerFinishedMessage*** – Функција која обавља верификацију примљене *Finished* поруке. Верификација се врши користећи алгоритам описан у тачки 3.4.10;
- ***TlsParseRecords*** – На пријему обрађује поруке протокола *TLS* блокова као што је описано у тачки 3.1;
- ***ProcessServerMessagesToCertificate*** – Обрађује *Handshake* поруку типа *ServerHello* као једину поруку која се у овој фази може наћи испред поруке типа *Certificate*;
- ***ProcessServerMessagesFromCertificate*** – Обрађује редом поруке које су стилге од сервера након поруке типа *Certificate*. У зависности од тога која порука је примљена, унутар ове функције се генеришу и одговарајуће поруке које ће бити послате као одговор;

- ***TlsAddHandshake*** – Генерише одговарајуће поруке протокола руковања које се серверу шаљу као одговор на поруке које су од сервера примљене;
- ***TlsAddAlert*** – Генерише одговарајуће поруке протокола узбуне ако се за тим укаже потреба;
- ***TlsAddChangeCipherSpec*** – Генерише поруку протокола размене сигурносних параметара;
- ***HmacMD5***- Имплементација H-MAC алгоритма описаног у тачки 3.5.1 користећи *MD5* хеш;
- ***HmacSHA*** - Имплементација H-MAC алгоритма описаног у тачки 3.5.1 користећи *SHA1* хеш;
- ***AddMACAndEncrypt*** – Функција која додаје аутентификацију (*MAC*) и шифрује податке приликом слања. За више детаља погледати тачку 3.1;
- ***DecryptAndVerifyMAC*** – Функција која након пријема дешифрује податке и проверава веродостојност примљених података поредећи *MAC*;
- ***PRF*** – Имплементација псеудо случајне функције описане у тачки 3.6.1;
- ***GenerateKeys*** – Функција која израчунава материјал за кључ користећи поступак описан у тачки 3.6.4;

5.4.6 MicroSSL интерфејс

MicroSSL је омотач око класа *Certificate, CertStore, RSAKey и TLS* па самим тим представља интерфејс који апликација TLSClient користи како би позивала методе поменутих класа.

5.4.7 Апликација TLSClient

Имплементација апликације *TLSClient* се налази у фајлу *TLSClient.cpp* у коме се налазе функције за успостављање конекције на жељени сервер коришћењем *Socket* објеката као и за прекидање те конекције, функције за испис упутства за коришћење апликације и на крају и главна функција која парсира аргументе задате у командној линији и врши успостављање шифроване комуникације.

5.5 Закључак

Уз све већу присутност интернета у свакодневном животу, нагло се повећала и трговина путем Интернета. То је донело и све већу потребу за сигурним преносом средстава плаћања. Протокол TLS пружа могућност успостављања сигурне комуникације, али да непажња корисника оставља простор за разне типове напада. Иако је протокол TLS рањив, савесна употреба протокола значајно смањује могућност за нападе, па се од корисника очекује да више пажње посвете сигурности.

Апликација *TLSClient* служи за тестирање основних функционалности имплеметације TLS протокола и као таква не садржи све функционалности које су неопходне једном TLS клијенту. Уколико се укаже потреба, апликација *TLSClient* се лако може унапредити додавањем нових функционалности. Нека од унапређења би могла бити:

1) Омогућавање размене шифрованог садржаја након успостављања шифроване комуникације.
2) Коришћење листе овлашћених организација за издавање сертификата која се налази у оперативном систему. Тренутно се ова листа задаје кроз параметар *–CAfile.*
3) Контрола шифарских система који су подржани. Тренутно постоје четири подржана шифарска система и не постоји могућност да се неки од њих не користи приликом успостављања шифроване комуникације.
4) Омогућавање наставка неке од предходних сесија. Тренутно не постоји та могућност.

6. Литература

1. R.Rivest, “The MD5 Message-Digest Algorithm”, The Internet Engineering Task Force (IETF), RFC 1321, April 1992

2. D. Eastlake, P. Jones, “US Secure Hash Algorithm 1 (SHA1)”, The Internet Engineering Task Force (IETF), RFC 3174, September 2001

3. E.Rescorla, “Diffie-Hellman Key Agrement Method”, The Internet Engineering Task Force (IETF), RFC 2631, June 1999

4. B. Schneier, “Applied cryptography”, John Wiley & Sons, 1995

5. T. Dierks, C. Allen,“The TLS protocol Version 1.0”, The Internet Engineering Task Force (IETF), RFC 2246, 1999

6. М. Ковинић, „Увод у криптографију и инфраструктуру јавних кључева“, Академска мрежа Србије (AMPEC), 2010

7. Б. Прачар, „Слабости протокола SSL/TLS на напад човеком у средини“, 2008