ЕЛЕКТРОТЕХНИЧКИ ФАКУЛТЕТ У БЕОГРАДУ

ИМПЛЕМЕНТАЦИЈА СИГУРНОГ IP ТУНЕЛА БАЗИРАНОГ НА AES ЕНКРИПЦИЈИ

– Мастер рад –

Кандидат:

Александар Џелетовић

Ментор:

доц. др. Зоран Чича

Београд, Новембар 2014.

САДРЖАЈ

1. Увод

Оног момента када је Интернет постао широко доступан, корисници су добили бесконачне могућности за комуникацију, информисање, учење, забаву итд. Данас се корисници све више ослањају на Интернет ради обављања уобичајених послова попут плаћања рачуна, куповине, планирања и остваривања пословних и приватних контаката са људима било где у свету. Због бројних могућности које нуди Интернет, потребно је бити и одговоран и опрезан корисник истог, због великог броја злонамерних корисника. Услед различитих облика превара и напада може да буде угрожена приватност и репутација корисника, опљачкан новац на банкарским рачунима, чак и лична и породична безбедност могу да буду угрожени ако се корисник неопрезно понаша на Интернету. Да би спречили или умањили ризик боравка на Интернету потребна је одговарајућа заштита попут заштитног софтвера (анти-вируси), затим ажурирање рачунара ради отклањања сигурносних рупа, блокирање електронске поште нежељеног садржаја, коришћење савремених претраживача итд. Такође, и сам провајдер услуга може понудити могућност заштите корисника попут употребе анти-спам софтвера, креирања безбедних тунела за повезивање пословница у случају пословних корисника итд.

Овај рад се бави заштитом корисничких података тако што се формира тунел за пренос шифрованих IP (*Internet Protocol*) пакета. Шифровање се врши применом AES (*Advanced Encryption Standard*) енкрипције која је тренутни стандард за шифровање симетричним кључем. Имплементација је реализована хардверски и намењена је употреби у Интернет рутерима. Помоћу реализоване имплементације провајдери могу лако да реализују безбедне тунеле за пословне кориснике. Програмски језик коришћен за имплементацију решења је VHDL (*VHSIC Hardware Description Language*). Коришћено је развојно окружење ISE произвођача Xilinx. Главни пројекат, као и тест пројекта намењени верификацији дизајна приложени су у електронском формату.

Остатак рада је организован на следећи начин. Друго поглавље даје основне информације о виртуелним приватним мрежама (*Virtual Private Network – VPN*), као и информације о самим IP тунелима, а у оквиру поглавља биће описана и основна идеја самог решења реализованог у оквиру ове тезе. Треће поглавље детаљно описује хардверску имплементацију предложеног решења. Четврто поглавље бави се верификацијом и анализом перформанси реализоване имплементације. Пето поглавље садржи закључна разматрања, као и предлоге потенцијалне примене реализоване имплементације.

2.ВИРТУЕЛНА ПРИВАТНА МРЕЖА

2.1. Увод

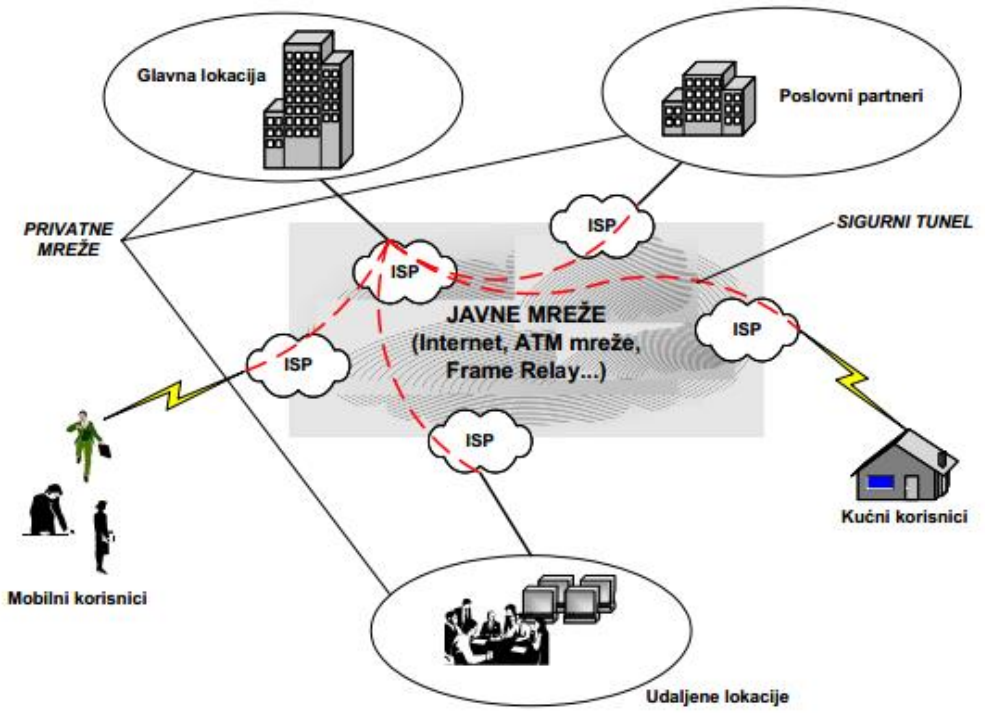
Виртуелна приватна мрежа (VPN) је приватна комуникациона мрежа која се користи за комуникацију у оквиру јавне мреже. Транспорт VPN пакета одвија се преко јавне мреже (типично Интернет) коришћењем стандардних комуникационих протокола. Назив „виртуелна“ се односи на чињеницу да је успостављена приватна мрежа, уствари логичке природе тј. настаје логичком поделом јавне мреже коју у исто време користи велики број других корисника и организација. Назив „приватна“ потиче од чињенице да нема мешања саобраћаја са саобраћајем других корисника, при чему је омогућена подршка за приватни адресни простор и могуће криптовање и заштита саобраћаја. VPN омогућава корисницима на раздвојеним локацијама да преко јавне мреже једноставно одржавају заштићену комуникацију, прецизније да успоставе сопствену (виртуелну) мрежу користећи инфраструктуру коју истовремено користе и други корисници који нису део те остварене виртуелне мреже. На слици 2.1 су приказане могућности коришћења VPN технологије [3].

Предности вирутелних приватних мрежа:

- Повећање географске покривености
- Уштеда у изнајмљеним линијама везе
- Уштеде у удаљеним и међународним *dail-up* позивима
- Могућност брзог додавања нових удаљених корисника

Мане виртуелне приватне мреже:

- VPN мреже захтевају добро знање о заштити података, првенствено у погледу хакерских напада и конфигурације VPN уређаја за рад у спрези са *firewall* уређајима.
- Уколико VPN конекција симулира рад рачунара у *LAN* (*Local Area Network*) мрежи, у зависности од квалитета VPN сервера и везе, проток података између рачунара умрежених у VPN је осетно спорији у односу на LAN.
- Питање интероперабилности различитих произвођача [1].



Слика 2.1 Могућности коришћења VPN технологије

2.2. Захтеви и компоненте виртуелне приватне мреже

Принципски захтеви који се постављају пред једну IP VPN мрежу су следећи:

- Концепт тајности тј. приватности (*security*) у оквиру кога имамо:
  - Аутентификација – идентификација корисника или уређаја пре него што се направи VPN конекција.
  - Интегритет података – представља доказ да приликом преноса садржај није измењен.
  - Поверљивост података – под овим се подразумева да подаци, приликом преноса, нису могли бити прочитани и искоришћени од треће стране.
  - Енкапсулација – под овим се подразумева како ће корисничка информација, као податак, бити енкапсулирана и пренешена преко мреже.
- Вишепротоколна подршка (*multiprotocol support*) – VPN мрежа би требало да подржи размену података под различитим протоколима.
- Управљање адресама (*address menagement*) – приватне адресе које корисници имају у оквиру VPN мреже не смеју да буду доступне на јавној мрежи.
- Гарантовани квалитет услуге (*Quality of Service – QoS*) – под овим се обично подразумевају пропусни опсег доступан корисницима, максимална кашњења пакета и гаранција испоруке пакета [1].

Компоненте VPN конекције су:

- VPN сервер – рачунар који прихвата VPN конекцију од VPN клијента.
- VPN клијент – рачунар који иницира VPN конекцију ка VPN серверу.
- Транзитна мрежа – дељена или јавна мрежа кроз коју пролазе енкапсулирани подаци.
- VPN конекција или VPN тунел – део конекције у којој су подаци шифровани и енкапсулирани.
- Протоколи тунеловања – протоколи који се користе за управљање тунелима и за енкапсулирање приватних података (на пример, *Point-to-Point Tunneling Protocol – PPTP*).
- Подаци који се шаљу кроз тунел – подаци који се обично шаљу кроз приватни *Point-to-Point* линк
- Аутентификација – идентитет клијента и сервера у VPN конекцији се аутентификују. Да би се осигурало да примљени подаци представљају податке које је стварно посало члан конекције (VPN клијент) и да подаци нису пресретнути и модификовани, VPN такође аутентификује податке које су послати [2].

2.3. Тунеловање

Унутар инфраструктуре међусобно повезаних мрежа, тунеловање представља технику преноса података намењених приватној мрежи преко јавне мреже. Протокол којим се имплементира тунеловање, уместо да шаље оригинални оквир, енкапсулира оквир у додатно посебно обликовано заглавље. Такво заглавље осигурава информације нужне за усмеравање енкапсулираних података кроз мрежу која служи за пренос до одредишта. Енкапсулирани подаци шаљу се између крајњих тачака тунела. Тунел је логички пут кроз који енкапсулирани подаци пролазе кроз мрежу. Појам тунел се уводи јер су подаци који путују тунелом разумљиви само онима који се налазе на његовом изворишту и одредишту. Када такав оквир дође до свог одредишта, из њега се извлаче корисни подаци који се затим шаљу на циљано одредиште. Тунеловање укључује читав процес енкапсулације, преноса и екстракције оригиналних података [3].

Технологија тунеловања има особине чије предности значајно доприносе њеној употреби, од којих су најважније:

- Сигурност – без обзира што тунел иде кроз несигурне јавне мреже, приступ подацима који су тунеловани није дозвољен неауторизованим корисницима што транспорт чини релативно безбедним
- Ниска цена – пошто се користе јавне мреже трошкови су значајно нижи када се упореде са трошковима потребним за изнајмљивање приватних линија или имплементација приватних интранет мрежа.
- Лакоћа имплементације – нема потребе за променом постојеће инфраструктуре јавних мрежа, па се VPN имплементира само на страни корисника.[4]

2.4. Посотојеће технологије

Данас постоје бројне технологије које имплементирају технику тунеловања. Најважније од њих су следеће:

- DLSW (*Data Link Switching*)
- GRE (*Generic Routing Encapsulation*)
- ATMP (*Ascend Tunnel Management Protocol*)
- Mobile IP – за мобилне кориснике
- IPSec (*Internet Protocol Security Tunnel Mode*)
- PPTP (*Point-to-Point Tunneling Protocol*)
- L2F (*Layer 2 Forwarding*)
- L2TP (*Layer 2 Tunneling Protocol*)

У овом раду биће објашњена само три најчешће коришћене технологије, а то су PPTP, IPSec и L2TP.

2.4.1. IPSec

IPSec је стандард дефинисан од стране IETF (*Internet Engineering Task Force*), а циљ његове израде био је сигуран транспорт информација преко јавних IP мрежа. IPSec је проткол трећег (мрежног) слоја (*Layer 3*), те у себи садржи неколико сигурносних технологија да би осигурао тајност, интегритет и аутентификацију. IPSec имплементира шифровање и аутентификацију у мрежном слоју, осигуравајући тако сигурну комуникацију од почетка до краја унутар мрежне инфраструктуре.

IKE (*Internet Key Exchange*) служи за одређивање сигурносних параметара и размене кључних информација између ентитета који учествују у комуникацији. Сигурносни параметри дефинишу везу између два или више ентитета, како ће ти ентитети користити сигурносне сервисе у циљу успоставе међусобне сигурне комуникације. IPSec сам по себи не поседује механизам за одређивање таквих сигурносних параметара, и због тога је IETF одабрао IKE као стандардну методу за дефинисање сигурносних параметара за потребе IPSec. При томе се користи IKMP (*Internet Key Management Protocol*). IKE ствара аутентификовани, сигурни тунел између два ентитета, те затим дефинише сигурносне параметре потребне за IPSec. Кроз тај процес два ентитета се морају међусобно аутентификовати и договорити заједничке кључеве.

Приликом рада IPSec користи следеће протоколе и стандарде:

- *Diffi-Hellman* методу размене кључева за одређивање кључева између два ентитета, криптографију темељену на јавним кључевима за дигитално потписивање комуникације приликом *Diffi-Hellman* размене кључева, да би се осигурао идентитет обе стране у комуникацији.
- AES, DES (*Data Encryption Standard*) или 3DES стандард за шифровање података
- HMAC (*Hashing Message Autentication*) стандард за аутентификацију
- Дигитална уверења потписана од стране одговарајућег ауторитета

IPSec протокол дефинише информације које се морају додати IP пакету да би се осигурала тајност, интегритет и аутентификација. Протоколи дефинисани у RFC 2406 (ESP – *Encapsulated Security Payload*) и RFC 2402 (AH – *Authentication Header*) део су IPSec архитектуре. Аутентификација заглавља (AH) се користи за аутентификацију извора и интегритет без употребе шифровања, док ESP осигурава исте услуге, али уз додатак механизма за шифровање. Сигурносни кључ познају само пошиљалац и прималац, а уколико

су аутентификациони подаци добри, прималац може бити сигуран да је податак стигао од пошиљаоца, те да није промењен током преноса.

IPSec подржава два начина рада, преносни начин рада (*transport mode*) и тунеловање (*tunnel mode*). У преносном начину рада шифрује се само IP пакет док IP заглавље остаје у оригиналном облику. Апликацијска заглавља су шифрована, а могућност прегледавања пакета је ограничена. Предност овог начина рада је да се сваком пакету додаје свега неколико октета. У овом начину рада уређаји на јавној мрежи могу видети адресе извора и одредишта поруке, што потенцијалном нападачу донекле омогућава одређене могућности анализе промета. Осим овог начина шифровања IP промета, IPSec има могућност IP тунеловања. Тунел се састоји од клијента и корисника који опслужује те кориснике, где су оба конфигурисана да користе IPSec тунеловање и договорене механизме за шифровање. IPSec тунеловање користи договорене механизме за енкапсулацију и шифровање читавих IP пакета што осигурава потпуно сигуран пренос преко јавних или приватних мрежа. Шифровани податак се спаја са одговарајућим нешифрованим IP заглављем, формирајући тако IP пакет који се на крају тунела дешифрује и обликује у IP пакет намењен крајњем одредишту. [3]

2.4.2. PPTP

Протокол PPTP је најстарији протокол за реализацију вирутелних приватних мрежа и базира се на PPP (*Point-to-Point*) протоколу, најпознатијем мрежном протоколу комуникације од тачке до тачке. PPTP протокол врши енкапсулацију PPP пакета од полазне тачке, преко Интернета, до одредишта. За аутентификацију корисника користи PAP (*Password Authentication Protocol*) или CHAP (*Challenge Handshake Authentication Protocol*). PAP протокол представља стандардну процедуру логовања клијента код сервера са корисничким именом и лозинком. CHAP протокол користи периодично генерисан стринг који сервер шаље клијенту. Овај примењује Hash функцију на тај стринг и отисак као резултат враћа серверу. Сервер обавања исту Hash операцију, и аутентификује клијента ако је резултат исти са оним добијеним од клијента [5].

Hash функције се првенствено користе за генерисање фиксне дужине излазних података који се понашају као референце на оригиналне податке. Ово је корисно када су оригинални подаци сувише гломазни да би се користили у целости. Једна практична примена је структура података која се зове хеш табела у којој су подаци смештени асоцијативно. Линерана претрага имена особа у листи траје дуже како се дужина листе повећава, али приликом употребе хеш функције добијамо константно време за претрагу. Друга употреба је криптографија, начини кодирања и очување података.[6]

За тунеловање PPP пакета преко Интернета PPTP користи GRE (*Generic Routing Encapsulation*) протокол. GRE је општи протокол који енкапсулира било који протокол од једне до друге тачке у мрежи. Криптовање тунелованих PPP пакета у PPTP протоколу врши MPPE (*Microsoft Point-to-Point Encryption*) протокол. MPPE проткол корсити криптовање RSA RC4 алгоритма, са кључевима дужине 40 или 128 бита који се мењају периодично.

PPTP протокол је протокол другог нивоа OSI модела и омогућује енкапсулацију, поред IP протокола, још и других протокола вишег нивоа, као што су IPX и NetBEUI. [5]

2.4.3. L2TP

Microsoft и Cisco заједнички су развили L2TP комбинујући најбоље карактеристике PPTP и L2F протокола. L2TP је мрежни протокол који служи за тунеловање PPP оквира преко мреже. L2TP енкапсулира PPP оквир, а слање се врши преко IP, X25, Frame Relay или ATM мреже. Подаци из енкапсулираних PPP оквира могу бити шифровани и/или компримовани. Протокол се такође може користити диреткно преко разних WAN медија без транспортног слоја. Могуће је истовремено стварање више тунела између истих крајњих тачака.

L2TP се састоји из два основна елемента L2TP приступног концентратора (LAC - *L2TP Access Concentrator*) и L2TP мрежних сервера (LNS - *L2TPNetwork Server*). Мрежни сервер представља логичку крајњу тачку PPP комуникације која се тунелује коришћењем приступпног концентратора [3].

2.5. Енкрипција и опис идеје пројекта

Идеја која се реализује у оквиру овог пројекта јесте идеја тунеловања, односно пренос IP пакета процесом тунеловоања, где се IP пакети претходно шифрују и затим им се додаје IP заглавље и онда се преко тунела достављају одредишној страни. На одредишној страни се врши обрнути процес, прво се проверава IP заглавље ако је заглавље добро оно се скида и врши се порцес дешифровања пакета и пакет се доставља кориснику.

Енкрипција или шифровање је процес криптографије којим се врши измена података тако да се подаци или поруке учине нечитљивим за особе које не поседују одређено знање тачније кључ. На тај начин се добија шифрована информација. Да би ови подаци постали разумљиви и употребљиви потрбено је да се дешифрују. Дешифровање се врши процесом супротним од шифровања. Сви системи енкрипције имају у својој основи следеће заједничке елементе:

- Алгоритам – функција, обично са јаком математичком основом, која обавља задатак енкрипције података
- Кључеви – користе се заједно са алгоритмима енкрипције и одређују коначни резултат процеса шифровања
- Дужина кључа – енкрипциони кључеви имају одређену дужину у зависности од тога који енкрипциони системи се користе. Дужина се мери бројем битова, а што су дужи кључеви, тежи су за разбијање система енкрипције
- Отворени текст – информације које желимо да шифрујемо
- Шифрован текст – информације након извршеног шифровања отвореног текста

Постоје две основне врсте енкрипције: симетрична и асиметрична енкрипција. Код симетричне енкрипције се и за шифровање и за дешифровање користи исти кључ. Код асиметричног шифровања постоји посебан кључ само за шифровање и други који служи за дешифровање. Ова два кључа називају се још јавни и тајни кључ. [7]

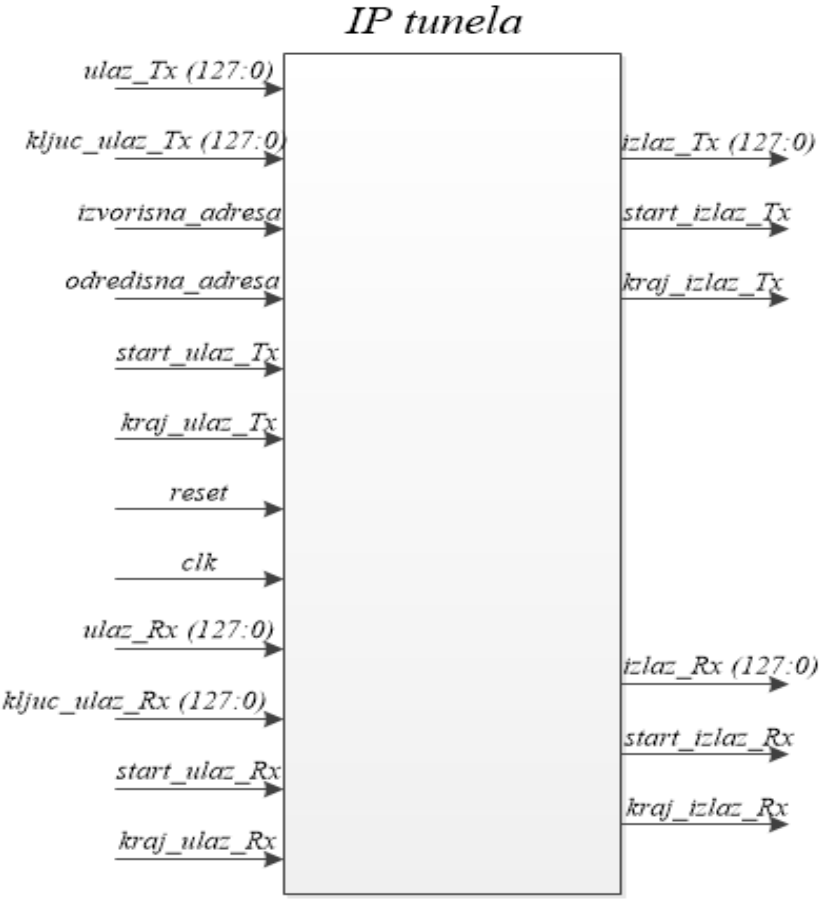
У овом пројекту процес шифровања и дешифровања се врши употребом AES стандарда који спада у класу алгоритама симетричне енкрипције. AES стандард подржава величине кључева од 128, 192 и 256 бита, а у овом пројекту је коришћен кључ дужине 128 бита. Величина блока који се шифрује/дешифрује је такође изражена у битима и износи 128, 192 или 256 бита, у овом пројекту је коришћена дужина блока од 128 бита.

3.ИМПЛЕМЕНТАЦИЈА СИГУРНОГ IP ТУНЕЛА

У овом поглављу биће описан програмски код коришћен за реализацију имплементације сигурног IP тунела. У наредним потпоглављима биће описани интерфејси дизајна, типови променљивих, коришћене процедуре као и унутрашњост црне кутије.

3.1. Интерфејси

У реализованом решењу постоје два дела, један је Tx односно предајни део, док је други Rx пријемни део, и у самом дизајну они су повезани паралелно, пошто је потребно обезбедити симултани пријем и предају података тј. пакета. У коду се може видети да дизајн садржи улазне и излазне интерефејсе, што се може видети и на слици 3.1.1.



Слика 3.1.1 Улазни и излазни интерфејси

Улазни интерфејси *clk* и *reset* се користе и за предајни и пријемни део. Сигнал *clk* се користи као сигнал такта при чему је активна узлазна ивица такта. Сигнал *reset* се користи за ресетовање дизајна тј. за иницијализацију почетних стања и предајног и пријемног дела.

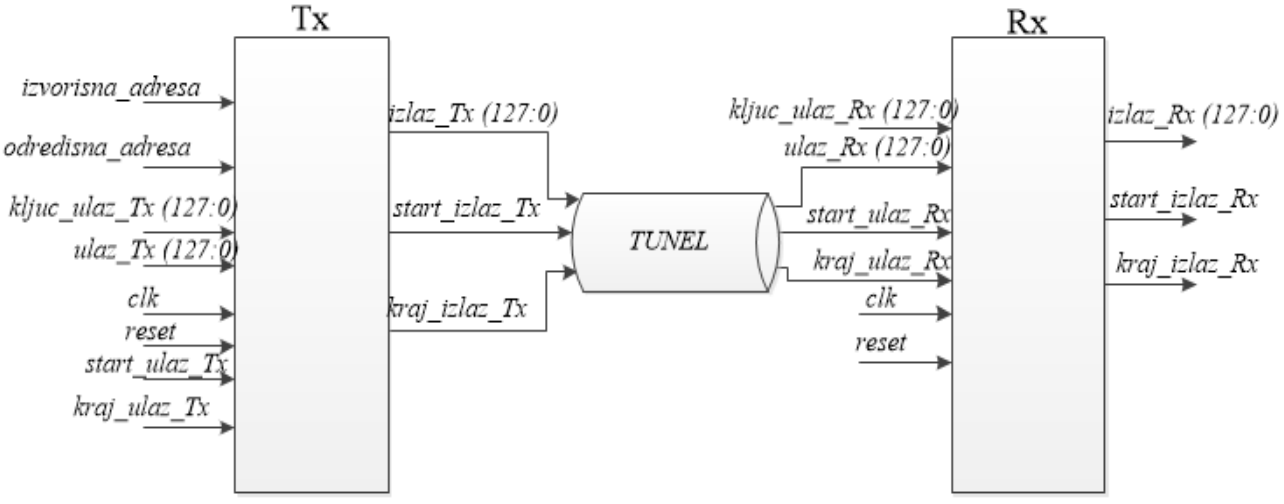
Улазни интерфејси *izvorisna_adresa, odredisna_adresa, kljuc_ulaz_Tx, ulaz_Tx, kraj_ulaz_Tx* и *start_ulaz_Tx* се користе само за потребе предајног дела. Сигнали *izvorisna_adresa* и *odredisna_adresa,* као што и сам назив каже, представљају изворишну адресу корисника који шаље пакет и одредишну адресу корисника који прима пакет (под

корисником се подразумева крајња тачка тунела). Дужина ових адреса је 32-битна, пошто је реализовано решење базирано на IPv4 протоколу. Ови сигнали су потребни како би изворишна страна правилно попунила заглавље које се додаје шифрованим корисничким IP пакетима који се тунелују. Сигнал *kljuc_ulaz_Tx* представља 128-битни кључ којим се шифрује пакет који се шаље кроз тунел. Наравно, на пријемној страни ће бити коришћен исти кључ за дешифровање. Сигнал *ulaz_Tx* представља 128-битни податак (део пакета) који се шаље одредишној страни, а који је потребно пре тога шифровати. Сигнали *start_ulaz_Tx* и *kraj_ulaz_Tx* представљају, респективно, почетак и крај пакета (прва и последња 128-битна реч пакета) који се треба шифровати а затим послати на одредишној страни.

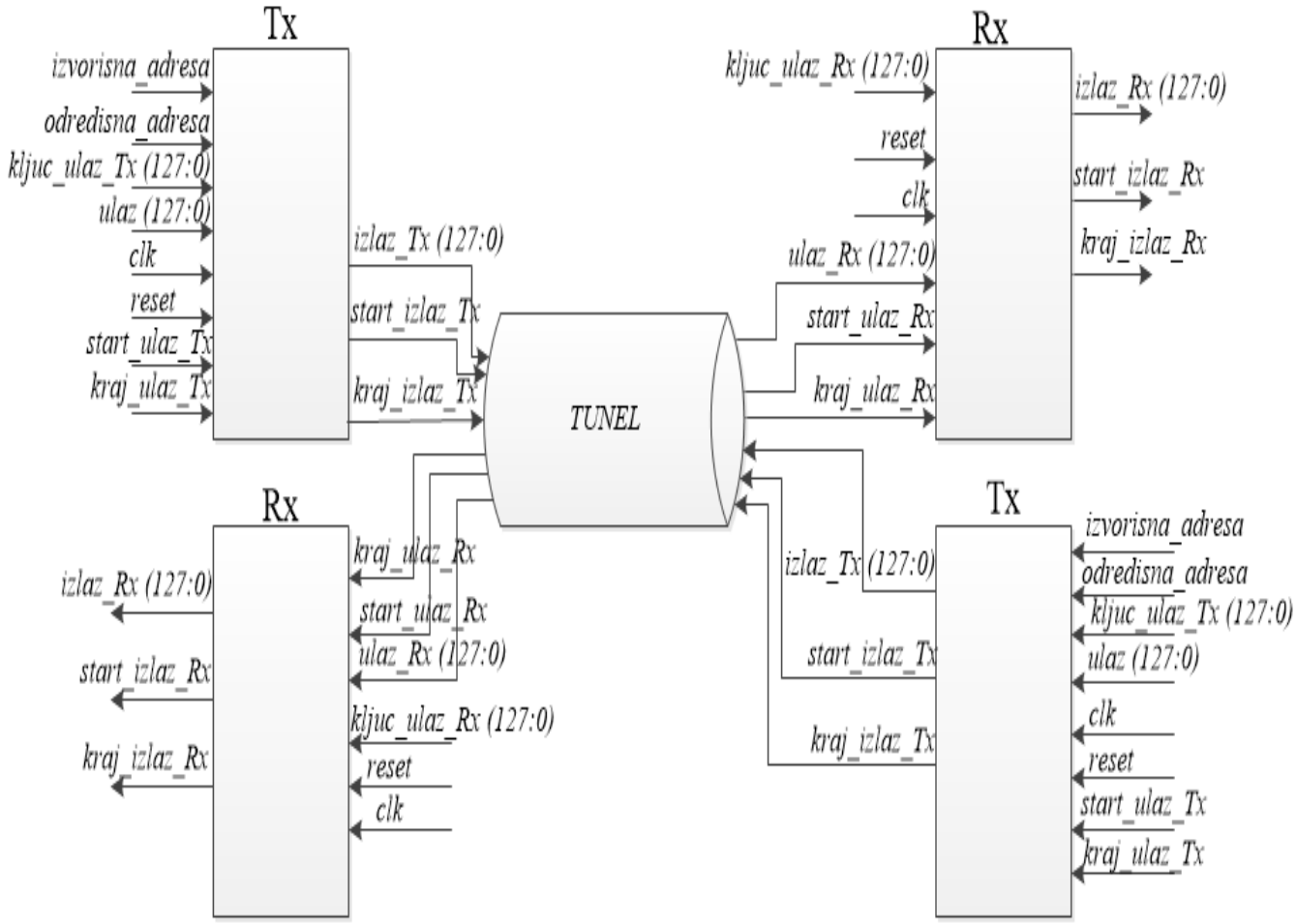
Излазни интерфејси *izlaz_Tx, kraj_izlaz_Tx, start_izlaz_Tx* представљају излазе предајног блока. Сигнал *izlaz_Tx* представља 128-битни податак (речи које представљају шифровани пакет + додато IP заглавље) који је шифрован и који такав треба преко тунела да се проследи до предајног блока. Сигнали *start_izlaz_Tx* и *kraj_izlaz_Tx* представљају, респективно, почетак и крај пакета који је шифорван (прва и последња 128-битна реч тунелованог пакета). Ови сигнали су битни за протокол нижег (другог) слоја да би могао да детектује све делове једног пакета ради енкапсулације у оквире другог слоја.

Улазни интерфејси *kljuc_ulaz_Rx, ulaz_Rx, kraj_ulaz_Rx* и *start_ulaz_Rx* се користе само за потребе пријемног блока. Сигнал *kljuc_ulaz_Rx* представља кључ којим ћемо дешифровати примљени податак тј. пакет, и он је исти као *kljuc_ulaz_Tx* коришћен на предајној страни дотичног тунела. Сигнал *ulaz_Rx* представља 128-битни податак (део тунелованог пакета) који је шифрован и који је потребно дешифровати да би пријемна страна могла да користи те податке. Сигнали *start_ulaz_Rx* и *kraj_ulaz_Rx* представљају, респективно, почетак и крај тунелованог пакета који се треба дешифровати. На основу њих предајни блок детектује делове тунелованог пакета (на пример, зна да прве две 128-битне речи тунелованог пакета представљају додато IP заглавље). Излазни интерфејси *izlaz_Rx, kraj_izlaz_Rx, start_izlaz_Rx* представљају излазе пријемног блока. Сигнал *izlaz_Rx* представља 128-битни податак који је дешифрован тј. представља 128-битне речи дешифрованог (корисничког) IP пакета. Сигнали *start_izlaz_Rx* и *kraj_izlaz_Rx* представљају, респективно, почетак и крај дешифрованог корисничког пакета (прва и последња 128-битна реч).

За потребе верификације унутар „црне кутије“ пријемни и предајни блок су везани на ред као на слици 3.1.2, тј. симулира се један смер тунела и омогућава истовремено испитивање рада предајног и пријемног блока. Сама имплементација подразумева реализацију са слике 3.1.3 где се на крају тунела истовремено налазе и предајни и пријемни блок који симултано раде омогућавајући двосмерну комуникацију кроз тунел.



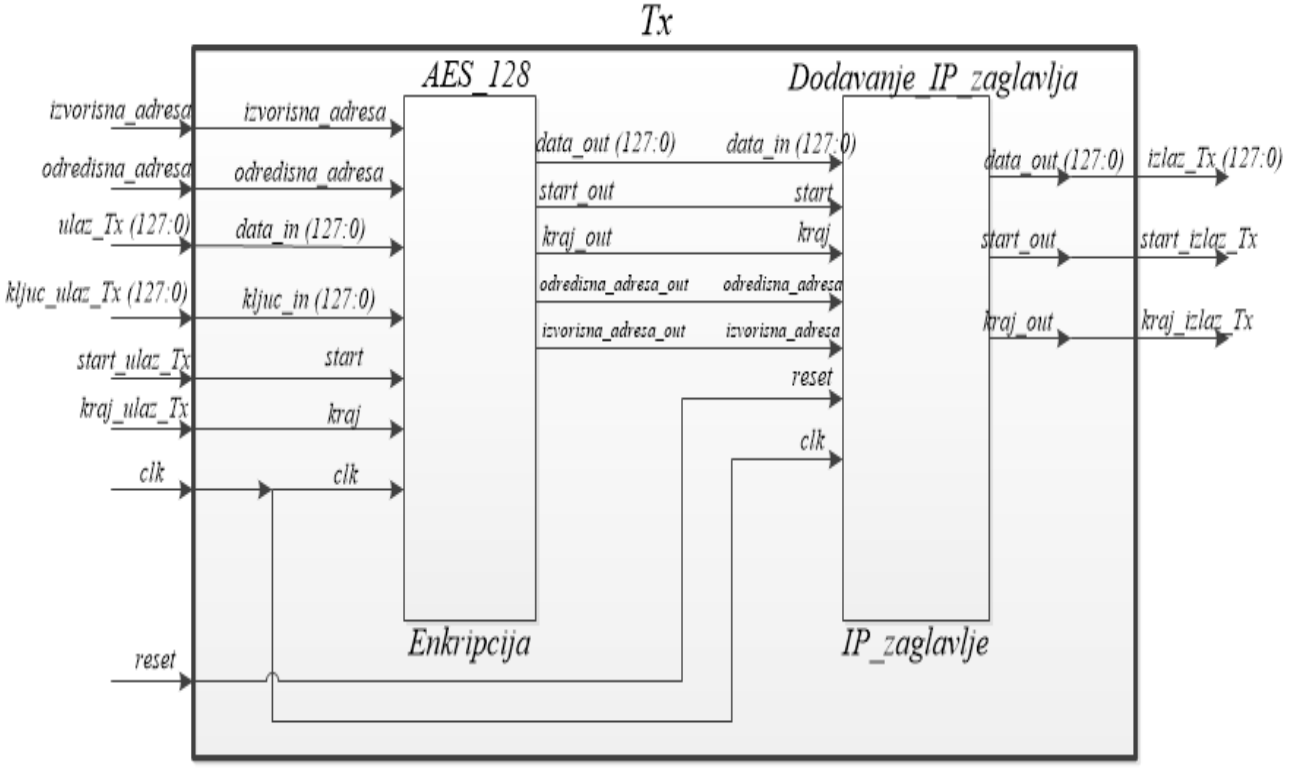
Слика 3.1.2 Излгед ако се предајни и пријемни блоку оквиру дизајна вежу на ред



Слика 3.1.3. Изглед предајника и прејмника везаних у паралела, као у нашем дизајну

3.2. Предајни блок – Tx

На слици 3.2.1 је приказана детаљна шема унутрашњости предајног блока.



Слика 3.2.1 Изглед унутрашњости предајног блока – Tx

И овај блок има улазне и излазне интерфејсе, улазни интерфејси су *izvorisna_adresa, odredisna_adresa, kljuc_ulaz_Tx, ulaz_Tx, kraj_ulaz_Tx* и *start_ulaz_Tx, clk, reset*. Док су излазни интерфејси *izlaz_Tx, kraj_izlaz_Tx, start_izlaz_Tx*. Сви наведени сигнали су објашњени у претходном потпоглављу. У предајном блоку прво се извршава AES поступак шифровања (енкрипција) пакета, а затим се врши процес додавања IP заглавља на основу кога ће се вршити усмеравање пакет кроз тунел.

Као што се види на слици 3.3.1. у оквиру предајног блока постоје два блока а то су *AES_128* и *Dodavanje_IP_zaglavlja*. О блоку *AES_128* ће бити више речи у одељку 3.3.1 а о блоку *Dodavanje_IP_zaglavlja* ће бити више речи у одељку 3.3.2.

3.2.1. AES шифровање

Као поступак за шифровање искоришћен је AES стандард [8].

Сам стандард има два одвојена блока то је блок за шифровање података, у коду дефинисано као *data_enkripcija*, и блок за развијање кључа којим се шифрују подаци, у коду дефинисан као *razvijanje_kljuca*. У зависности од дужине кључа имамо и различити број рунди током којих се шифрује податак. У нашем случају пошто је кључ дужине 128 бита имамо 11 рунди, током којих пролази податак који се шифрује. Пошто је у питању пајплајн имплементација свака рунда се засебно имплементира у хардверу, тачније у сваком такту се имплементира једна рунда за одређени податак. Овом техником се постиже висок проток процесирања, и може се испунити да се у свакој рунди налазе различити подаци, што значи да ћемо имати једанаест података који се истовремено шифрују један за другим.

У рундама за шифровање података се користе четири функције, које се имплементирају следећим редом:

- *SubByte* (S-box)
- *ShiftRow*
- *MixColumn*
- *AddRoundKey*

Функција *S-box* представља табелу која се састоји од прва 4 и последња 4 бита улаза и њихов пресек у табели даје податак који се добија на излазу функције. Улаз у ову функцију је 8-битни податак, а и излаз је такође 8-битни податак. Сама табела се састоји из 256 комбинација. Име функције (процедуре) у коду је *sbox2*. У самом коду табела је представљена преко *case* структуре, где улаз представља 8-битни податак и на основу њега излаз такође представља 8-битни податак, али друге вредности. Пошто је улаз у ову функцију 8-битни податак, а улазни податак који се шифрује има 128 бита, ради лакшег рада са овом функцијом направљен је ентитет у којој се 128-битни податак дели на по шеснаест 8-битних делова, који у паралели пролазе кроз функцију *s-box*. Назив тог ентитета је *sbox*, улаз и излаз су 128-битни подаци. Табела помоћу које се извршава функција дата је у оквиру стандарда [8].

За функцију *ShiftRow* се ствара матрица димензије 4x4 где је свако место у матрици 8-битни податак. У овој функцији се врши замена места редова у поменутој матрици, тачније врши се циклично померање редова матрице, и то циклично померање улево. Први ред се не помера остаје исти, други ред се помера за један бајт односно осам бита, трећи ред се помера за два бајта односно 16 бита и трећи ред се помера за три бајта односно 24 бита. И за ову функцију направљен је ентиет где су улаз и излаз 128 битни подаци. Назив овог ентитета је *shift_rows*. У оквиру овог програма функција је реализована једноставном заменом места 8-битних података у оквиру једног 128-битног података. На табелама 3.2.1.1 приказан је изглед улазног и излазног 16-бајтног односно 128-битног податка.

Табела 3.2.1.1 Изглед улазног и излазног 16-бајтног податка

Улазна табела			
1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

Излазна табела			
1	5	9	13
6	10	14	2
11	15	3	7
16	4	8	12

Функција *MixColumn* посматра такође матрицу 4x4, улаз ове функције је 32-битни податак над којим се врши нелинеарна, инвертибилна операција. Улази у ову функцију се посматрају као полиноми над пољем $GF(2^8)$ тачније у нашем случају то је „1b“ у хексадецималном бројном систему. Код функције (процедура) је написан тако што је улаз подељен на четири нова улаза са по осам бита, и потом се тих осам бита посматрају засебно. Сама матрица је дефинисана на следећи начин:

$$\begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \end{bmatrix} = \begin{bmatrix} 02030101 \\ 01020301 \\ 01010203 \\ 03010102 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} \quad (3.2.1.1)$$

Y_1, Y_2, Y_3 и Y_4 представљају 8-битне излазе, а цела Y матрица представља излаз функције која је дужине 32 бита. Сигнали X_1, X_2, X_3 и X_4 представљају 8-битне улазе док цела матрица X представља улаз функције који је дужине 32 бита. Да би се оваква функција направила потребно је извршити множење са 2 односно са 3. То је урађено на тај начин што се посматра највиши бит 8-битног податка, и ако је тај бит једнак нули онда се множење са два врши једноставним померањем улево 8 битног податка, тј. на крај података се додаје нула. Множење са три се врши тако што се податак прво помери улево, и онда се врши функција ексклузивни или (*xor*) над таквим податком и првобитним 8-битним податком. Ако је највиши бит једнак јединици онда се у оба наврата, и за множење са два и са три, податак прво помери улево а потом се над податком и константом „1b“ (хексадецималан запис) изврши операција *xor*. Тако добијени податак представља множење са два. Док множење са три се извршава тако што се на тај податак извршава операција *xor* са првобитни податком који се посматра. И тако се овај поступак извршава четири пута тј. за све четири колоне улазне матрице тј. све четири 32-битне речи. Ова мања функција која ради са једном колоном матрице у оквиру пројекта је названа *mix_kolona2*. Као и за претходне две функције и за ову је направљен ентитет где су улаз и излаз 128-битни подаци. Пошто је улаз 128-битни податак, у оквиру нашег ентитета, чији је назив *mix_kolona*, позива се функција *mix_kolona2* којом се дели 128-податак на четири 32-битна податка, односно врши се позив функције *mix_kolona2* четири пута.

Функција *AddRoundKey* врши логичку операцију ексклузивно или над податком који се шифровао и кључем који је дефинисан за дату рунду. У оквиру пројекта ентитет за ову функцију носи назив *add_round_key*.

Ове функције су имплементиране уоквиру *data_enkripcije* где се врше рунде, а улази у овај блок су и једанаест кључева који су различити за сваку рунди и зато их има једанаест. Пошто у нашем случају имамо 11 рунди, не извршавају се све функције у свакој рунди. У почетној рунди извршава се само *AddRoundKey* функција, док у наредних девет рунди извршавају се све функције и то истим редом којим су објашњене. У последњој рунди се извршавају редом функције *S-box, ShiftRow* и *AddRoundKey*, и на излазу из ове рунде се добија податак који је шифрован. Ради прегледности и боље контроле дизајна, направљена су три ентитета која се потом позивају у оквиру ентитета *data_enkripcija*, чији су називи *prva_runda_Tx, runda_Tx* и *poslednja_runda_Tx*. У оквиру ентитета *prva_runda_Tx* позван је само ентитет за функцију *AddRoundKey*, пошто се само она извршава у првој рунди. Затим у *runda_Tx* се врши позивање и повезивање сва четири ентитета појединачних функција рунде. И *poslednja_runda_Tx* се састоји само од *S-box, ShiftRow* и *AddRoundKey*, пошто се само оне извршавају у последњој рунди.

Имплементација пајплајн технике подразумева креирање по једне инстанце прве и последње рунде, односно девет инстанци регуларне рунде уз правилно повезивање (уланчавање) дотичних инстанци. Креиран је нови тип *niz_registara* ради ефикаснијег кода за повезивање инстанци рунди.

```
type niz_registara is array(10 downto 0) of std_logic_vector(127downto0);
signal kljucevi,data_inn,data_outt:niz_registara;
```

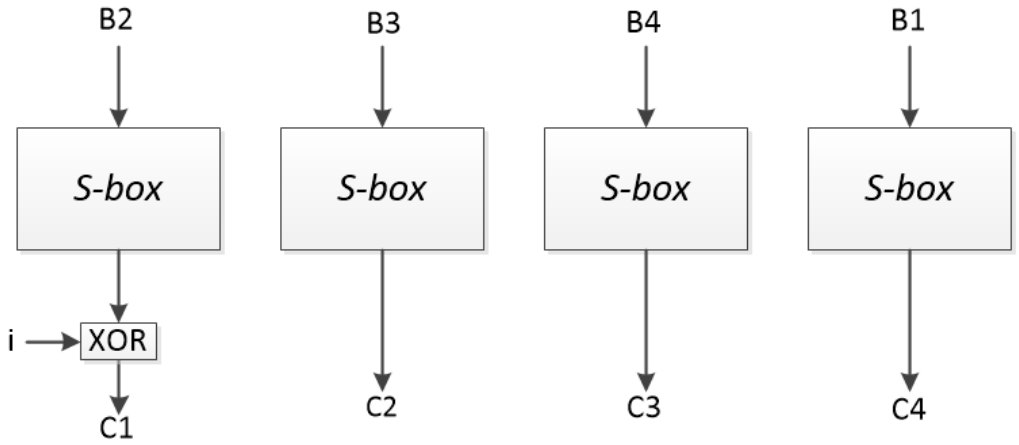
Са овим типом је омогућено ефикасније писање кода за повезивање инстанци рунди.

```
prva_inst:prva_runda_Tx
port map
(
      data_in=>data_inn(0),
      kljuc_in=>kljucevi(0),
      data_out=>data_outt(0)
);

generisi_runde:FOR i IN 1 TO 9 GENERATE
runda_inst: runda_Tx
port map
(
      data_in=>data_inn(i),
      kljuc_in=>kljucevi(i),
      data_out=>data_outt(i)
);
END GENERATE;

poslednja_inst:poslednja_runda_Tx
port map
(
      data_in=>data_inn(10),
      kljuc_in=>kljucevi(10),
      data_out=>data_outt(10)
);
```

У оквиру блока *razvijanje_kljuca* врши се развијање кључева за потребе сваке рунде. И овде као и код шифровања података имамо једанаест рунди (у оквиру кода то је функција *razvijanje_kljuca*), а самим тим и једанаест кључева који се генеришу. Улаз у овај блок представља *kljuc_ulaz_Tx,* који уједно представља и кључ за прву рунду шифровања података. Затим се тај кључ доводи на улаз друге рунде, и где се врши поступак развијања кључа, који се понавља и у наредних десет рунди. На слици 3.2.1.1 приказан је поступак развијања кључа. Од кључа који се доведе на улаз у другу рунду се узме само првих 32 бита и тих 32 бита се подели на четири групе од по 8 бита, где је првих осам бита B1 (највиших 8 бита), других 8 бита B2, трећих 8 бита B3 и четвртих 8 бита B4 (најнижих 8 бита). Затим те четири групе замене места, на начин који је приказан на слици 3.2.1.1, где на место највиших 8 бита долазе бити из групе B2 и тако редом. Потом те групе битова пролазе кроз функцију *S-box*, која је претходно описана. Након проласка кроз функцију *S-box*, над највиших 8 бита и сигналом *i* се врши функција ексклузивно или. Сигнал *i* представља позицију кључа који се посматра, за почетну рунду сигнал *i* нема вредност, зато што је улазни кључ уједно кључ и за почетну рунду. За прву рунду сигнал *i* има вредност 1, за другу 2, за трећу 3, за четврту 4 итд. Само што ове вредности морају да буду у оквиру поља $GF(2^8)$, тако да те вредности представљене у хексадецималном бројном систему износе, респективно, 01, 02, 04, 08, 10, 20, 40, 80 и 1b.



Слика 3.2.1.1 Поступак развијања кључа

Затим се након добијања нових бита C1, C2, C3, C4 одређује првих 32 бита новог кључа на следећи начин:

K1=C1 xor B1

K2=C2 xor B2

K3=C3xor B3

K4=C4 xor B4

Потом се од добијених 32 бита стварају нова 32 бита на следећи начин:

K5=K1 xor B5

K6=K2xor B6

K7=K3xor B7

K8=K4 xor B8

Затим се од нових 32 бита добија следећих 32 бита одговарајућег кључа на следећи начин:

K9=K5 xor B9

K10=K6xor B10

K11=K7xorB11

K12=K8xor B12.

И на крају последња 32 нова бита се добијају на следећи начин:

K13=K9 xor B13

K14=K10xor B14

K15=K11xor B15

K16=K12 xor B16.

Битови B1,B2,B3,...,B16 представљају бајтове кључа који долазе на улаз рунде. Бајтови K1,K2,K3,...,K16 представљају бајтове новог кључа (излаз рунде), где K1 представља највиших 8 бита а K16 најнижих 8 бита. Овај поступак се понавља за свих једанаест рунди.

Заједно са податком који је шифрован такође се на излаз шаљу и сигнали *start_izlaz* и *kraj_izlaz* који сигнализирају, како им и сам назив каже, почетак односно крај пакета. Пошто сам пакет касни једанаест тактова, онда је урађено и кашњење и одговарајућих сигнала (за

сигнализацију старта и краја пакета) са улаза, а то је извршено преко низа од једанаест регистара којима се остварује кашњење сигнала старт и крај са улаза у складу са кашњењем које уноси само шифровање једног 128-битног податка.

3.2.2. Додавање IP заглавља

Изглед IP заглавља приказан је на табели 3.3.2.1. Поље за укупну дужину пакета и поље *checksum*, као и изворишна и одредишна адреса се разликују од случаја до случаја. Остала поља имају константну вредност, у оквиру овог пројекта, где је и замишљено да се заглавље додаје на податке који се шаљу кроз тунел.

Табела 3.3.2.1. IP заглавље

Битови	**0-3**	**4-7**	**8-15**	**16-18**	**19-31**
0-31	*Version*	*IHL*	*Type of Service*	*Total length*	
32-63	*Identification*			*Flags*	*Fragment Offset*
64-95	*Time To Live*		*Protocol*	*Header Checksum*	
96-127	Изворишна адреса				
128-159	Одредишна адреса				

Version – представља верзију IP заглавља, у овом пројекту је коришћена верзија IPv4, тако да поље *Version* има вредност 4, односно то поље ће имати вреднсот 0100 у бинарном бројном систему.

IHL (*Internet Header Length*) – представља дужину додатог заглавља, у овом пројекту постављено је да дужина заглавља има вредност 5 (0101). Та вредност означава да заглавље има дужину 160 бита. Постоји могућност да ово поље има вредност већу од 5, што би онда значила да је дужина заглавља већа од 160 бита и у том случају заглавље би садржало и поље Опције, али пошто се ово поље не користи у нашем случају онда је дужина заглавља увек иста.

Type of Service – односи се на тип сервиса и користи се у процесу остваривања квалитета сервиса. У оквиру ове тезе ово поље се не користи и вредност поља је све нуле. Међутим, у будућим верзијама имплементације не би представљало велики проблем поставити ово поље на жељену вредност у складу са захтевима квалитета сервиса ради квалитетнијег преноса кроз тунел.

Total length – дефинише укупну дужину пакета у бајтовима укључујући и заглавље и податке који се шаљу. Вредност овог поља зависи од броја 128-битних шифрованих делова корисничког пакета + две 128-битне речи заглавља.

Identification – низ бројева који заједно са изворишном адресом, одредишном адресом и корисничким протоколом треба да јединствено идентификује пакет. У реализованој имплементацији ово поље је постављено на све нуле.

Flags – састоји од три бита која се користе у процесу фрагментације. Процес фрагментације је углавном непожељан па је из тог разлога стављено ово поље на све нуле.

Fragment Offset – користи се у процесу фрагментације. Ово поље има вредност све нуле у оквиру реализованог дизајна.

Time To Live (*TTL*) – време живота пакета у скоковима. Поставља се на хексадецималну вредност 40.

Protocol – показује протокол вишег слоја коме треба проследити пакет. У овом пројекту је коришћена хексадецимална вредност 11 што представља *UDP* (*User Datagram Protocol*). У суштини на пријемној страни тунела ће бити игнорисана ова вредност јер се зна да примљени датаграм енкапсулира кориснички датаграм.

Header Checksum – контролна сума за детектовање грешке. Ово поље се рачуна по специјалној формули, где се прво сва поља саберу, тако сабрана поља су дужине 20 бита. Потом се првих 4 бита сабира са преосталих (нижих) 16 бита и добија се 16-битна реч. Потом се тај податак инвертује, чиме добијамо 16 битну реч која представља контролну суму заглавља [9].

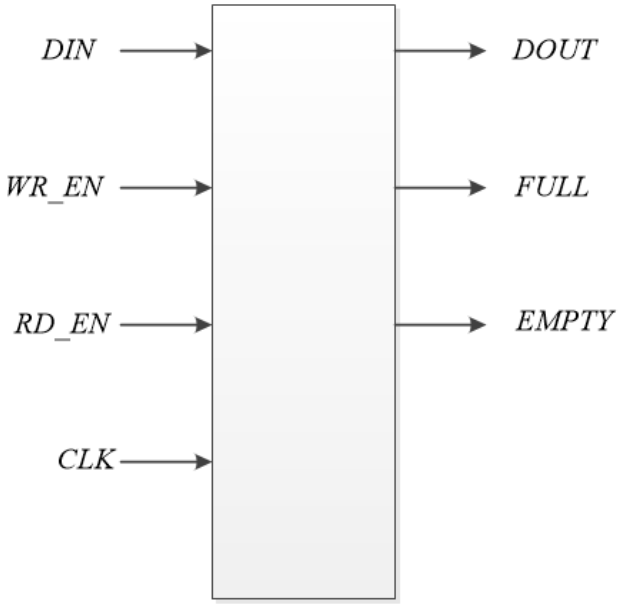
У оквиру блока додавање IP заглавља коришћене су три FIFO (*First In First Out*) меморије. Меморије су креиране помоћу алата *Core Generator*, који обезбеђује *ISE* окружење. Изглед FIFO меморијског блока приказан је на слици 3.2.2.1.

Улази овог блока су:

- *DIN* – податак који се уписује у меморију
- *WR_EN* – сигнал који треба да буде активан када се врши упис у меморију. Када имамо податак на сигналу *DIN* упоредо са тим податаком треба да стигне и активна вредност (активна вредност је 1, а неактивна вредност је 0) на овај сигнал, што омогућава да се податак упише у меморију.
- *RD_EN* – сигнал који треба да буде активан када желимо нешто да прочитамо из меморије. Када овај сигнал добије активну вредност (активна вредност је 1, а неактивна вредност је 0), меморијски блок добија обавештење да треба прочитати податак из меморије, и у наредном такту тај податак ће се наћи на излазу из меморијског блока.
- *CLK* – представља сигнал такта

Излази из овог блока су:

- *DOUT* – податак који се исписује из меморије
- *FULL* – сигнал који нам говори када је меморија пуна, и када он има активну вредност то је знак да је меморија пуна и да не може више да се уписује у њу.
- *EMPTY* – сигнал који нам говори када је меморија празна, и када овај сигнал има активну вредност то је знак да је меморија празна и да не можемо из ње ништа да читамо (активна вредност је 1, а неактиван вредност је 0).



Слика 3.2.2.1. Изглед FIFO блока

Намена сва три FIFO меморијска елемента је иста, а користимо их за потребе правилног попуњавања IP заглавља.

Први FIFO блок се користи за чување података (128-битни делови корисничког пакета) који долазе на улаз блока *Dodavanje_IP_zaglavlja*, и у оквиру пројекта овај меморисјки блок је назван *fifo_generator*. Да бисмо одредили која је укупна дужина нашег датаграма морамо прво да примимо све делове пакета па тек онда можемо да знамо која је дужина корисничког пакета који се шаље (прецизније дужина корисног дела тунелованог пакета), и због тога се користи овај меморијски блок за привремено чување делова шифрованог пакета док се не заврши његов пријем из блока за шифровање. Величина меморије је 1024 128-битних речи што је довољно за два IP пакета максималне дужине.

Други FIFO блок се користи за чување укупног броја делова корисничког пакета (128-битних речи) и у оквиру пројекта овај меморијски блок је назван *fifo_generator_brojac.* Овај податак је неопходан за ишчитавање тачног броја 128-битних делова шифрованог пакета из претходно описаног FIFO блока. Такође, тај податак је битан и за правилно попуњавање поља *Total length* додатог заглавља. Додато заглавље има 160 бита, али је извршено поравнање на нивоу 128 бита да би се поједноставила хардверска имплементација. Из тог разлога се последњих 32 бита друге 128-битне речи тунелованог пакета попуњава нулама. Отуда друга 128-битна реч тунелованог пакета садржи последњи део додатог заглавља и бите попуне ради поравнања на 128-битну границу. Отуда се поље *Total length* додатог заглавља поставља на вредност (*N*+2)*128/8 где *N* представља број 128-битних делова шифрованог корисничког пакета.

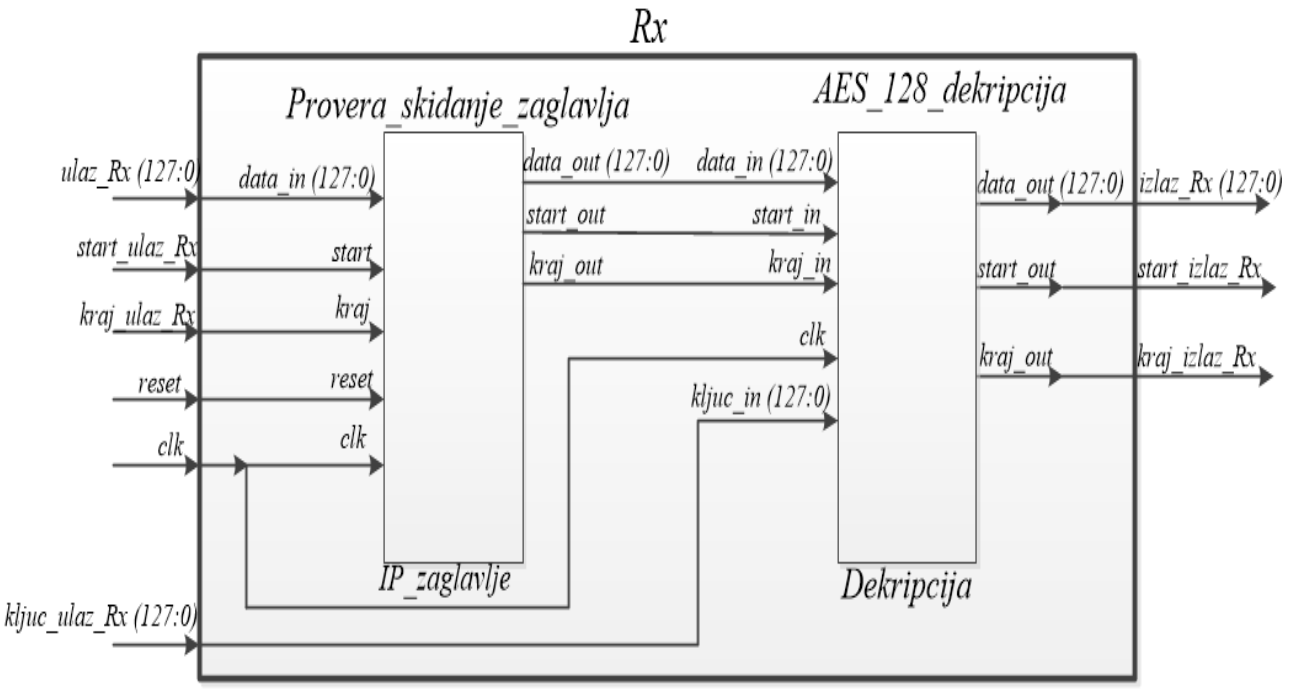
Трећи FIFO блок се користи за чување изворишне и одредишне адресе. У пројекту овај меморисјки блок има назив *fifo_generator_adresa.* Изворишна и одредишна адреса са улаза се лепе у заједничку 64-битну реч која се уписује у меморију. На овај начин је омогућено да се исти предајни блок користи истовремено за више тунела. У суштини изворишна адреса би се могла поставити и на константну вредност, али је унета могућност њеног динамичког уписа ради постизања веће флексибилности у конфигурисању тунела.

Сам код се састоји из две функционалне целине тј. два процеса која се обављају у паралели. Један процес обавља упис података у све FIFO блокове, а сам испис из FIFO блокова, као и креирање тј. додавање заглавља се извршава у другом процесу. Ради боље

контроле процеса креирања тунелованог пакета и читања одговарајућих података из FIFO блокове, дотични процес је организован у виду коначног аутомата. Приликом слања тунелованог пакета на одговарајући излаз генеришу се и излазни сигнали почетка и краја тунелованог пакета којима се сигнализира постављање (слање) прве, односно последње 128-битне речи тунелованог пакета.

3.3. Пријемни блок – Rx

На слици 3.3.1 је приказана детаљна шема унутрашњости пријемног блока.



Слика 3.3.1 Изглед унутрашњости пријемног блока – Rx

Улазни интерфејси овог блока су *ulaz_Rx, clk, kraj_ulaz_Rx, start_ulaz_Rx* и *kljuc_ulaz_Rx* а излазни интерфејси су *izlaz_Rx, kraj_izlaz_Rx* и *start_izlaz_Rx*. Сви наведени сигнали су већ раније објашњени.

У пријемном блоку прво се извршава поступак провере и скидања додатог заглавља тунелованог пакета уколико је заглавље исправно (у супротном се тунеловани пакет одбацује), а затим се врши поступак дешифровања ради добијања оригиналног садржаја корисничког пакета.

Као што се види на слици 3.3.1. у оквиру пријемног блока постоје два блока: *Provera_skidanje_zaglavlja* и *AES_128_dekripcija*.

О блоку *Provera_skidanje_zaglavlja* ће бити више речи у одељку 3.3.1.

О блоку *AES_128_dekripcija* ће бити више речи у одељку 3.3.2.

3.3.1. Провера и скидање IP заглавља

Провера IP заглавља се врши ради провере да ли је тунеловани пакет стигао без грешака у заглављу. Ако постоје грешке, тунеловани пакет се одбацује. Провера подразумева проверу вредности одговарајућих поља у оквиру заглавља. Једно од најважнијих поља приликом саме провере заглавља је *Header Checksum* тј. контролна сума, чијом провером се утврђује да ли постоје битске грешке у заглављу.

Провера овог поља се врши тако што се сва поља саберу и на тај начин добијамо збир дужине 20 бита. Потом се највиша четири бита добијеног збира сабријау са преосталих 16 бита новог збира. И ако је после овог сабирања резултат „ffff“ то значи да је заглавље исправно. У случају исправне провере додато заглавље се скида (не шаље на излаз овог блока) и шаље се само корисни део тунелованог пакета који садржи кориснички пакет који је шифорван.

Заједно са шифрованим пакетом се шаљу и сигнали који указују на почетак и крај пакета.

3.3.2. AES дешифровање

Поступак дешифровања (декрипције) је такође део стандарда AES. Овај блок се састоји од два блока, слично као и код AES шифровања, први блок представља развијање кључа и тај блок је потпуно идентичан истом блоку код поступка шифровања и други блок је блок којим се врши поступак дешифровања. Сами називи ових блокова у оквиру кода су *razvijanje_kljuca* и *data_dekripcija.* У овом одељку биће објашњен други блок, тачније сам поступак дешифровања, пошто је поступак развијања кључа објашњен у одељку 3.2.1.

Поступак дешифровања је супротан поступку шифровања. Редослед функција које су се обављале у поступку шифровања је сада супротан, што значи да опет имамо четири функције које се извршавају. Ове функције су исто као и у порцесу шифровања представљени као ентитети. Функције за које су направљени нови ентитети су:

- *AddRoundKey*
- *Инверзни MixColumn*
- *Инверзни ShiftRows*
- *Инверзни SubByte* (S-box)

Функција *AddRoundKey* је идентична као истоветна функција код шифровања, тачније врши операцију екслизивно или (*xor*) над податком и кључем за одговарајућу рунду. У оквиру пројекта одговарајући ентитет је назван *inv_add_round_key.*

Инверзна функција *MixColumn* за потребе дешифровања се знатно разликује од функције *MixColumn* за потребе шифровања. Матрица која се сада посматра је:

$$\begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} \qquad (3.3.2.1)$$

Y_1, Y_2, Y_3 и Y_4 представљају 8-битне излазе, а цела Y матрица представља излаз функције која је дужине 32 бита. Сигнали X_1, X_2, X_3 и X_4 представљају 8-битне улазе док цела матрица X представља улаз функције који је дужине 32 бита. Овде нам је потребно сада да извршимо множење са 9, 11 , 13 и 14 (хексадецималне вредности 09, 0b, 0d и 0e). Овај поступак множења је извршен тако што је за сваки 8-битни улаз направљено множење са два, по принципу који је објашњен у одељку 3.2.1. Затим је са тим податком направљено још једно множење са два, и на тај начин се добија податак који је помножен са четири. И на

крају са новим податком урадимо још један поступак множења са два, и тако добијемо првобитни 8-битни податак који помножен са осам. И затим извршимо поступак ексклузивно или (*xor*) над одређеним подацима да би добили жељене производе, на пример, да би добили првобитни податак помножен са 11, извршимо операцију *xor* над првобитним податком, првобитним податком помноженим са два и првобитним податком помноженим са осам. Пошто је и овде улаз у функцију 32-битни податак, а главни улазни податак је 128-битни направљен је ентитет чији су улази и излази 128-битни подаци (ентитет *inv_mix_kolona*). У оквиру тог ентитета је извршено позивање функције *inv_mix_kolona2* одговарајући број пута.

Инверзна функција *ShiftRows* је у овом делу пројекта супротна од исте функције за шифровање. Извршава се исти принцип, ствара се матрица 4x4 и онда се врши замена места одређених бајтова. Улазна и излазна матрица је приказана у табели 3.3.2.1. У овом случају се врши супротан процес, пошто су бити у делу за шифорвање „промешани“ ми сада вршимо враћање бита у одговарајући редослед. У оквиру пројетка ентитет је назван *inv_shift_rows.* Који се састоји само из улазног и излазног 128-битног податка.

Табела 3.3.2.1 Изглед улазног и излазног 16-бајтног податка

Улазна табела			
1	5	9	13
6	10	14	2
11	15	3	7
16	4	8	12

Излазна табела			
1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

Инверзна функција *SubByte* сада има другачију табелу којом се врши дешифровање података. Та табела је исте дужине, 16x16, као и табела за шифровање. Пошто је улаз у овај блок 8-битни податак, направњена је функција која има улаз и излаз дужине 8-бита. Та функција је направљена употребом *case* структуре, и њен назив је *inv_sbox*. Ова функција је затим позвана у оквиру ентитета који је назван *inv_sbox2* и овај ентитет има улазни и излазни 128-битни сигнал. У оквиру овог ентитет наша функција је позвана шеснаест пута, зато што смо морали да направимо да улаз у функцију буде 8-битни податак, а наш 128 битни податак се састоји од шеснаест 8-битних података. Табела за потребе дешифровања дата је у оквиру стандарда [8].

Сам код дешифровања је сличан, али опет у неким деловима се разликује од кода шифровања. И овде као и код процеса шифровања направљена су три ентитета која одговарају рундама. Први етитет је *prva_runda_Rx* где се врши позивање ентитета који се користе у првој рунди, а то су инверзне функције *AddRoundKey, ShiftRows* и *SubByte*. Други ентитет је *runda_Rx* где се врши позивање ентитета који се користе регуларним рундама, а то су инверзне функције *AddRoundKey, MixColumn, ShiftRows* и *SubByte*. И трећи ентитет је *poslednja_runda_Rx* где се врши позивање само ентитета *AddRoundKey.*

Само додељивање кључева у оквиру процеса дешифровања се разликује из разлога што је нама у процесу дешифровања у првој рунди потребан кључ који је намењен једанаестој рунди, пошто идемо супротним редоследом од шифровања где је тај кључ био намењен последњој рунди овде мора бити намењен првој рунди, онда је извршен посутпак намерног кашњења кључева. То је извршено тако што је дефинисан нови тип сигнала за

сваки кључ посебно, који представља низ чија дужина зависи од самог закашњења. Пошто је нама потребно да нам у првом такту када нам дође податак који треба дешифровати на улазу имамо кључ број једанест, у наредном такту кључ број десет, па у наредном кључ број девет, итд.. Ово кашњење се извршава да би имали одговарајући кључ у тренутку који нама одговара и због тога је потребно да кључ број једанаест (у оквиру пројекта то је *kljuc10* зато што су кључеви дефинисани од 0 до 10) имамо одмах на улазу када нам стигне податак, па у следећем такту кључ број десет (*kljuc9*) итд. Овде је приказан тип за кључ један (*kljuc0*):

```
type kljuc0 is array (1to10) of std_logic_vector(127downto0);
signal kljuc0_in1:kljuc0;
```

Пошто се нама у процесу развијања кључа прво развије први кључ па затим други и тако редом до једанаестог кључа, то значи да се наш једанаести кључ развије тек након десет рунди и због тога је први кључ закашњен десет рунди, други девет, трећи осам итд. док једанаести кључ није закашњен он нам је потребан одмах на почетку поступка дешифровања. Кашњење је реализовано једноставном *for* петљом:

```
kljuc0_in1(1)<=kljuc0_in;
for i in 2 to 10 loop
      kljuc0_in1(i)<=kljuc0_in1(i-1);
end loop;
```

Креирање инстанци три ентитета *prva_runda_Rx, runda_Rx*, и *poslednja_runda_Rx* је реализовано тако да се оствари најплајн имплементација ради омогућавање високих протока дешифровања, слично као у блоку за шифровање. Креиран је нови тип сигнала ради остваривања ефикасног кода креирања и повезивања инстанци рунди:

```
type niz_registara is array(10downto0) of std_logic_vector(127downto0);
signal kljucevi,data_inn,data_outt:niz_registara;
```

Код за креирање и повезивање инстанци рунди:

```
prva_inst:prva_runda_Rx
port map
(
      data_in=>data_inn(0),
      kljuc_in=>kljucevi(0),
      data_out=>data_outt(0)
);
generisi_runde:FOR i IN 1 TO 9 GENERATE
runda_inst: runda_Rx
port map
(
      data_in=>data_inn(i),
      kljuc_in=>kljucevi(i),
      data_out=>data_outt(i)
);
END GENERATE;
poslednja_inst:poslednja_runda_Rx
port map
(
      data_in=>data_inn(10),
      kljuc_in=>kljucevi(10),
      data_out=>data_outt(10)
);
```

У блоку data_dekripcija се такође врши кашњење улазних сигнала који сигнализирају старт и крај пакета, тако да ти сигнали синхроно излазе са корисничким пакетом на излазу блока.

4. ВЕРИФИКАЦИЈА ДИЗАЈНА И АНАЛИЗА ПЕРФОРМАНСИ

4.1. Верификација дизајна

У овом потпоглављу биће описана процедура верификације дизајна. За верификацију је коришћена функционална симулација у оквиру *ISim* симулатора.

У оквиру симулације су на ред повезани предајни и пријемни блок ради тестирања једног смера тунела. Послати IP пакети би требало да се појаве у идентичном облику на излазу тунела тј. пријемног блока са синхронизованим сигналима који приказују почетак и крај пакета. У верификацији биће приказана група од два IP датаграма, сви подаци су представљени у хексадецималном бројевном систему. Није вођено рачуна о исправности садржаја корисничких датаграма пошто је циљ тунела да пренесе садржај који прими непромењен кроз тунел, па је циљ верификације да утврди да се кориснички садржај који се тунелује заиста непромењен и преноси кроз тунел.

Први IP датаграм:

Изворишна адреса: 9c2a700b

Одредишна адреса: a0f3c1cc

Пакети:

cdb9c7ef0fd5757ebdc8052949d1de43

c046cccfb22e5985238cbfecc9e5f5e7

9439bd8da20d050f8beccbf35bf8bc71

Други IP датаграм:

Изворишна адреса: a0f3c1cc

Одредишна адреса: 9c2a700b

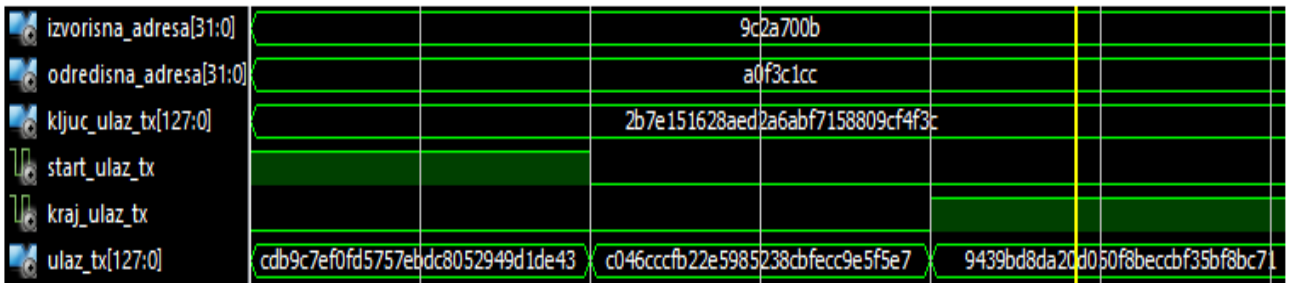
Пакети:

4b01461c400643e3ef4124e787971bfb

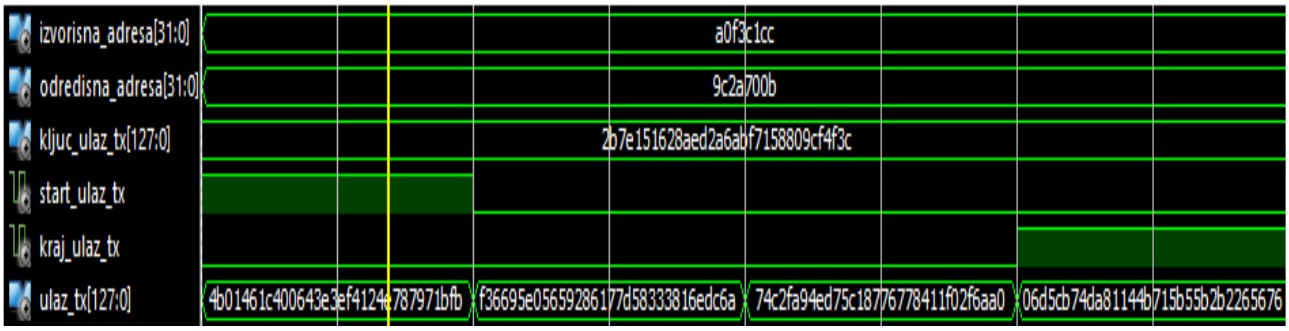
f36695e05659286177d58333816edc6a

74c2fa94ed75c18776778411f02f6aa0

06d5cb74da81144b715b55b2b2265676



Слика 4.1.1. Приказ улазних аргумената првог IP датаграма



Слика 4.1.2. Приказ улазних аргумената другог IP датаграма

На сликама 4.1.1. и 4.1.2. се јасно види да на улазу имамо наше пакете из првог и другог IP датаграма, видимо и да је кључ којим се шифрују/дешифрују подаци увек исти. Такође видимо и да су сигнали који сигнализирају почетак и крај пакета такође добро синхронизовани са 128-битним деловима пакета.

Ови сигнали потом пролазе кроз процес шифровања, где се сваки пакет шифрује на начин објашњен у одељку 3.2.1. У табели 4.1.1. су приказане шифроване вредности пакета.

Табела 4.1.1. Приказ шифрованих пакета

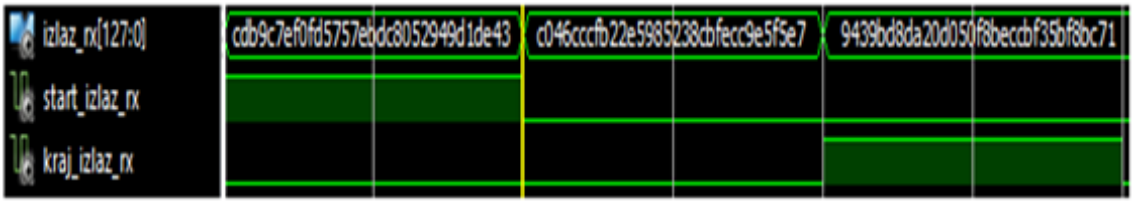
Оригинални пакети	**Шифровани пакети**
cdb9c7ef0fd5757ebdc8052949d1de43	4c20a3b0995bc66e92a64532f999b158
c046cccfb22e5985238cbfecc9e5f5e7	019d5d815139140b58cb215a4f724012
9439bd8da20d050f8beccbf35bf8bc71	bd9647abf642c95464bc9c08348a7854
4b01461c400643e3ef4124e787971bfb	d6ac16e82561c7200a0cc74d10975769
f36695e05659286177d58333816edc6a	e6e79ee1819d729967e7efaccf9ecd45
74c2fa94ed75c18776778411f02f6aa0	dbd0b885906d848746b596eae6645a54
06d5cb74da81144b715b55b2b2265676	cadc30b7f79f2f98eaf1b587b3367d6e

Затим ови пакети пролазе кроз блок за додавање заглавља и након тог блока се шаљу на пријемну страну. Изглед додатог заглавља за ова два IP датаграма је дат у табели 4.1.2. и то у хексадецималном запису.

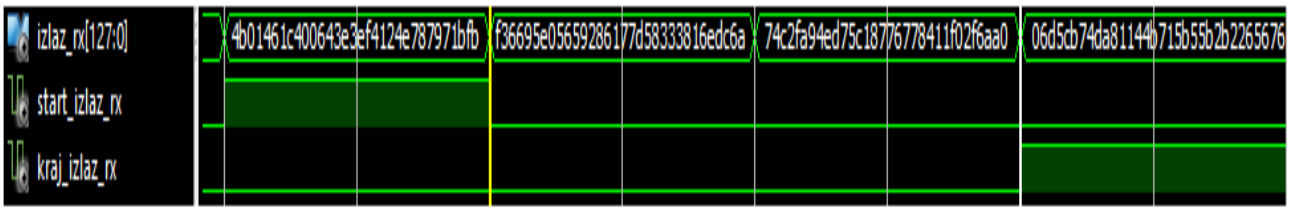
Табела 4.1.2 Изглед заглавља

IP датаграм	**IP заглавље**
Први	4500 0280 0000 0000 4011 8977 9c2a700b a0f3c1cc
Други	4500 0300 0000 0000 4011 88f7 a0f3c1cc 9c2a700b

На пријемној страни се прво врши провера заглавља, ако је оно исправно онда се шифровани пакети шаљу блоку за дешифровање, ради добијања оригиналних корисничких пакета који се потом шаљу на излаз пријемног блока. Поступак дешифровања који је објашњен у одељку 3.3.2. се врши са истим кључем као поступак шифровања.



(а)



(б)

Слика 4.1.3. Приказ излазних аргумената (а) првог IP датаграма и (б) другог IP датаграма

На слици 4.1.3. видимо да су на излаз тестираног ентитета послати управо они пакети које смо послали тј. садржај пакета на улазу у тунел и на излазу из тунела се поклапају. Може се такође приметити да су и сигнали који приказују почетак и крај пакета такође добро синхронизовани. Овиме је потврђени исправан рад реализоване имплементације.

4.2. Анализа перформанси

У овом пројекту је коришћен FPGA чип XC5VLX300T фамилије Virtex5 [10]. Перформансе дизајна су приказане у табели 4.2.1.

Табела 4.2.1 Перформансе дизајна

	Коришћено	**Доступно**	**Проценат искоришћености**
Број слајс регистара	8118	207360	3%
Број слајс LUT-ова	35578	207360	17%
Број потпуно искоришћених парова LUT-FF	7313	36383	20%
Број пинова	842	960	87%
Број блокова RAM/FIFO	6	324	1%
Број глобалних тактова (BUFG/BUFGCTRL)	1	32	3%

На основу приказаних резултата се може видети да дизајн има скромне захтеве. Употреба пајплајн технике за остваривање високог протока пакета захтева засебно имплементирање инстанци сваке рунде што повећава количину употребљених ресурса пре све регистара (који се користе између пајплајн фаза) и комбинационе логике која користи LUT-ове. За линкове мањих капацитета могла би се користити хардверски скромнија имплементација шифровања и дешифровања у оквиру којих би се смањио број инстанци регуларних рунди (тада се не би могла користити пајплајн техника).

На основу табеле 4.2.1 се може видети да је за потребе овог пројекта искоришћен велики број пинова самим тим и узет је уређај код кога је број пинова нешто мало већи од потребног броја. Међутим, реално је да се реализовани дизајн повеже са другим интерним блоковима (на пример, са блоком који имплентира други слој) па пинови не представљају проблем.

5.ЗАКЉУЧАК

У овом раду је урађена реализација сигурног IP тунела, при чему је сигурност постигнута употребом AES шифровања. Реализована имплементација користи пајплајн технику у процесу шифровања, односно дешифровања чиме је омогућена подршка линковима високих протока. Такође, имплементација троши реалтивно скромне хардверске ресурсе чиме је погодна за имплементацију у постојеће рутере.

Реализована имплементација може наћи примену у постојећим IP мрежама јер омогућава једноставно креирање тунела једноставним додавањем IP заглавља тунелованом садржају. Подешавањем одредишне адресе у заглављу се омогућава рутирање пакета кроз мрежу до жељеног одредишта (супротног краја тунела), а у случају пресретања пакета, нападач не може да открије кориснички садржај јер је тај садржај шифрован.

У будућим унапређењима имплементације може се укључити и подршка за квалитет сервиса подешавањем вредности поља за тип сервиса. Такође, идеја реализоване имплементације се може потенијално применити и на креирање IPv6 тунела.

Литература

[1] Презентација „Виртуелне приватне мреже“ Ђорђе Илић, Електротехнички факултет у Београду. Преузето са: home.etf.rs/~vm/diplomski/VPN_prezentacija1.ppt

[2] Конфигурација VPN конекције http://www.link-elearning.com/dlmaterijali/_materijali//DL2277_novo/SadrzajNJpdf/2277_36-KnewE0wsKmGz.pdf?&hSajt=2a304a1348456ccd2234cd71a81bd338

[3] Основни концепт VPN технологије http://www.cis.hr/www.edicija/_LinkedDocuments/CCERT-PUBDOC-2003-02-05.pdf

[4] VPN Technologies: Definitions and Requirements. Преузето са: http://www.vpnc.org/vpn-technologies.html

[5] Безбедност података у VPN на јавној глобалној мрежи. Преузето са: http://www.academia.edu/934414/Bezbednost_podataka_u_VPN-u_na_javnoj_globalnoj_mre%C5%BEi

[6] Хеш функција . Преузето са: http://sr.wikipedia.org/sr/He%C5%A1_funkcija#Standardna_upotreba_he.C5.A1iranja_u_kriptografiji

[7] Data Encryption. Преузето са: http://www.parliament.uk/documents/post/postpn270.pdf

[8] AES стандард. Преузето са: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[9] Internet protocol. Преузето са: https://tools.ietf.org/html/rfc791

[10] Спецификације за фамилије чипова *Virtex*5. Преузето са: http://www.xilinx.com/support/_documentation/data_sheets/_ds100.pdf