

УНИВЕРЗИТЕТ У БЕОГРАДУ
ЕЛЕКТРОТЕХНИЧКИ ФАКУЛТЕТ

Марко А. Мићовић

**Заштита приватности на мрежном слоју
применом шифровања са очувањем формата**

докторска дисертација

Београд, 2026.

UNIVERSITY OF BELGRADE
SCHOOL OF ELECTRICAL ENGINEERING

Marko A. Mićović

**Network Layer Privacy Protection
Using Format Preserving Encryption**

Doctoral Dissertation

Belgrade, 2026.

Ментор

др Павле Вулетић, редовни професор
Универзитет у Београду, Електротехнички факултет

Чланови комисије

др Захарије Радивојевић, ванредни професор
Универзитет у Београду, Електротехнички факултет

др Дејан Симић, редовни професор
Универзитет у Београду, Факултет организационих наука

др Зоран Чича, редовни професор
Универзитет у Београду, Електротехнички факултет

др Милош Цветановић, ванредни професор
Универзитет у Београду, Електротехнички факултет

др Саша Стојановић, ванредни професор
Универзитет у Београду, Електротехнички факултет

Датум одбране

Захвалница

Писање захвалнице сам циљано оставио за крај, након свих осталих корака које докторска дисертација носи са собом. Дуго сам ишчекивао овај тренутак и замишљао га као велико олакшање услед завршетка рада на дисертацији. Ипак, гледајући уназад, писање дисертације на моменте ми делује мање захтевно него што је у стварности било, док се захвалница сада чини као нови задатак који није лако обавити. То ме је довело до закључка да је сваки део докторских студија, као и живота генерално, у датом тренутку изазован на свој начин, и да је овај пут био много лакши захваљујући људима којима сам окружен.

Читавом колективу Катедре за рачунарску технику и информатику Електротехничког факултета хвала на дивној атмосфери у којој је задовољство радити. Ментору, професору Павлу Вулетићу, хвала на помоћи током рада на дисертацији, како кроз бројне смернице, тако и кроз спремност да „засуче рукаве“ када је то било најпотребније. Професорима Бошкуну Николићу, Саши Стојановићу, Милошу Цветановићу и Захарију Радивојевићу хвала на приступачности и разумевању, одмереном и брижном вођењу кроз рад на факултету, корисним саветима и усмеравању у стручном развоју, као и дугим разговорима и великодушно издвојеном времену иако би ти разговори често прерастали у моје монологе. Хвала колеги Жики, чије су сурово реалне, а истовремено духовите опсервације рада на факултету и докторских студија увек успевале да ме насмеју.

Најближим колегама, који су ми временом постали пријатељи, Јоцкету, Укију и куму Кристијану, пуно хвала на претресању интересантних тема, заједничком корачању кроз изазове и неисцрпној енергији за стварање незаборавних тренутака. Старим пријатељима, Тамари и Спалету, много хвала на великој количини занимљивог садржаја послатог између тактички провучених питања о прогресу дисертације, као и непроспаваним ноћима „баченим“ на компјутерске игре и бројним одиграним мечевима баскета. Велико хвала тати Александру, мами Весни и сестри Сари, на безусловној подршци и вери у мене, што је темељ на који сам се ослањао током свих година.

На крају, неизмерну захвалност дугујем супрузи Живани и ћерки Тари, на пруженој љубави и мотивацији да напишем дисертацију, као и на одушевљеном лупкању по тастатури, доношењу омиљених играчака уз осмех, неодложном повлачењу за руку и другим неодољивим методама привлачења пажње. Све то ме је из дана у дан подсећало да је време да приведем докторске студије крају, како бих се коначно посветио битним стварима. Поред захвалности, дугујем им и обећање да ћу убудуће мање времена проводити испред екрана.

Наслов дисертације: Заштита приватности на мрежном слоју применом шифровања са очувањем формата

Сажетак: Употреба интернета подлеже опсежном праћењу и анализи, при чему се велика количина података о личности прикупља из различитих извора. На мрежном слоју, подаци о личности могу се прикупљати анализом образаца мрежног саобраћаја. Уколико корисник комуницира путем статичке јавне *IP* адресе, или се може недвосмислено повезати са одређеном јавном *IP* адресом у датом временском интервалу, његова приватност је угрожена. На основу *IP* адресе могуће је открити географску локацију корисника, док нападачи могу реконструисати његове навике и понашање надзором скупа посећених *IP* адреса. Додатно, прецизније информације могу се добити пресретањем корисникових *DNS* упита. Стога је неопходно посветити посебну пажњу механизмима којима се прекида веза између *IP* адресе и идентитета корисника.

Предмет истраживања ове дисертације је заштита приватности на мрежном слоју кроз превенцију профилисања корисника и спречавање цурења личних података. Заштита приватности се постиже техником скривања у маси, која се реализује замагљивањем *IP* адреса, односно изменом њихових хост делова. Ниво заштите приватности директно је сразмеран величини замагљеног дела *IP* адресе, јер већи опсег замагљивања имплицира већу анонимну групу. Како би се приликом замагљивања очувала стандардна структура заглавља пакета, примењује се шифровање са очувањем формата. Ова криптографска техника обезбеђује да шифровани текст задржи исту дужину и формат као и отворени текст.

Циљ истраживања је да се докаже изводљивост примене шифровања са очувањем формата за заштиту приватности замагљивањем делова заглавља пакета у реалном времену, под реалним оптерећењем саобраћаја и без значајне деградације перформанси. У ту сврху пројектован је и имплементиран систем за заштиту приватности на мрежном слоју који не захтева чување стања. Развијене су две варијанте система за различите платформе: прва је реализована за *Netronome* програмабилну мрежну картицу коришћењем *P4* језика и *Micro-C* језика, док је друга заснована на *eBPF* технологији и извршава се унутар језгра *Linux* система.

Резултати експерименталне евалуације потврђују да је примена шифровања са очувањем формата ефикасан метод за заштиту приватности на мрежном слоју, уз занемарљив утицај на перформансе мреже. Имплементација на *Netronome* платформи остварује *TCP* пропусни опсег на нивоу брзине везе и обрађује до 4,5 Мpps, уз додатно кашњење од свега 30 μ s до 60 μ s по пакету у једном смеру. У поређењу са *Oblivious DNS over HTTPS* протоколом, предложени систем заснован на *eBPF* технологији постиже од 20 % до 45 % мање кашњење у једном смеру, у зависности од интензитета *DNS* саобраћаја и симулираног мрежног кашњења између клијента, посредника и разрешивача.

Кључне речи: заштита приватности, шифровање са очувањем формата, рачунарске мреже, систем назива домена, програмабилни уређаји

Научна област: Електротехника и рачунарство

Ужа научна област: Рачунарска техника и информатика

УДК број: 621.3

Dissertation title: Network Layer Privacy Protection Using Format Preserving Encryption

Abstract: Internet usage is subject to extensive monitoring and analysis, with a vast amount of personal data being collected from various sources. At the network layer, personal data can be inferred by analyzing network traffic patterns. If a user communicates via a static public IP address, or can be unambiguously linked to a specific public IP address within a given time interval, its privacy is compromised. Based on the IP address, it is possible to reveal the user's geographical location, while attackers can reconstruct its habits and behavior by monitoring the set of visited IP addresses. Additionally, more precise information can be obtained by intercepting the user's DNS queries. Therefore, it is necessary to pay special attention to mechanisms for breaking the link between IP address and user identity.

The subject of this dissertation's research is network layer privacy protection through the prevention of user profiling and personal data leakage. Privacy protection is achieved using the hiding-in-the-crowd technique, realized by obfuscating IP addresses, specifically by modifying their host parts. The level of privacy protection is directly proportional to the size of the obfuscated part of the IP address, as a larger obfuscation range implies a larger anonymity set. To preserve the standard structure of the packet header during obfuscation, format-preserving encryption is applied. This cryptographic technique ensures that the ciphertext retains the same length and format as the plaintext.

The goal of the research is to demonstrate the feasibility of using format-preserving encryption for privacy protection by obfuscating parts of packet headers in real-time, under real traffic loads, and without significant performance degradation. For this purpose, a stateless network layer privacy protection system was designed and implemented. Two variants of the system were developed for different platforms: the first was realized for the Netronome programmable network interface card using the P4 language and Micro-C language, while the second is based on eBPF technology and executes within the Linux kernel.

Experimental evaluation results confirm that the application of format-preserving encryption is an effective method for privacy protection at the network layer, with negligible impact on network performance. The implementation on the Netronome platform achieves wire-speed TCP throughput and processes up to 4.5 Mpps, with an additional latency of only 30 μ s to 60 μ s per packet in one direction. Compared to the Oblivious DNS over HTTPS protocol, the proposed system based on eBPF technology achieves 20% to 45% lower one-way latency, depending on the intensity of DNS traffic and the simulated network delay between the client, relay, and server.

Keywords: privacy protection, format-preserving encryption, computer networks, system domain name, programable devices

Scientific field: Electrical Engineering and Computing

Scientific subfield: Computer Engineering and Informatics

UDC number: 621.3

Садржај

1	Увод	1
2	Проблем заштите приватности	4
2.1	Заштита приватности на апликативном слоју	4
2.2	Заштита приватности веб комуникације	6
2.3	Заштита приватности на мрежном слоју	7
2.3.1	<i>HORNET</i>	8
2.3.2	<i>TARANET</i>	9
2.3.3	<i>LAP</i>	10
2.3.4	<i>Dovetail</i>	12
2.3.5	<i>PHI</i>	14
2.3.6	<i>PANEL</i>	16
2.3.7	<i>AHP</i>	18
2.3.8	<i>SPINE</i>	21
2.3.9	<i>PINOT</i>	23
2.4	Анализа постојећих решења	27
3	Шифровање са очувањем формата	31
3.1	<i>NIST FPE</i> препоруке	32
3.1.1	<i>FF1</i> алгоритам	34
3.1.2	<i>FF3-1</i> алгоритам	37
3.2	<i>KATS FPE</i> стандарди	39
3.2.1	<i>FEA-1</i> алгоритам	43
3.2.2	<i>FEA-2</i> алгоритам	44
3.3	Одабир алгоритма	45
4	Архитектура <i>LISPP</i> система	47
4.1	Принцип функционисања	48
4.2	Обрада пакета	49
4.3	Шифровање поља заглавља пакета	50
4.4	Модел претњи	51
4.5	Случајеви коришћења	52
4.6	Промена криптографског материјала	53
5	Имплементација <i>LISPP</i> система	54
5.1	Модификација садржаја пакета	54
5.1.1	<i>P4</i> језик	56
5.1.2	<i>eBPF</i> технологија	61
5.1.3	<i>XDP</i> прикључак	66
5.2	Имплементација за <i>Netronome SmartNIC</i> уређај	67
5.2.1	Имплементација у целости на <i>P4</i> језику	69

5.2.2	Имплементација са <i>AES</i> алгоритмом на <i>Micro-C</i> језику	70
5.2.3	Имплементација са <i>FF3-1</i> алгоритмом на <i>Micro-C</i> језику	71
5.3	Имплементација за <i>eBPF</i> технологију	71
5.3.1	Имплементација помоћу <i>XDP</i>	72
6	Евалуација <i>LISPP</i> система	75
6.1	Евалуација <i>Netronome LISPP</i> система	75
6.1.1	Експериментална евалуација	75
6.1.2	Смањење броја <i>AES</i> рунди	78
6.2	Евалуација <i>eBPF LISPP</i> система	79
6.2.1	Експериментална евалуација	80
6.3	Резиме резултата	82
7	Примена <i>LISPP</i> система	83
7.1	<i>DNS</i>	83
7.1.1	Класични <i>DNS (Do53)</i>	86
7.1.2	<i>DNS</i> преко <i>TLS (DoT)</i>	87
7.1.3	<i>DNS</i> преко <i>HTTPS (DoH)</i>	88
7.1.4	Несвесни <i>DNS (ODNS)</i>	89
7.1.5	Несвесни <i>DoH (ODoH)</i>	91
7.1.6	Анонимизовани <i>DNSCrypt (ADNSCrypt)</i>	92
7.1.7	Узајамни несвесни <i>DNS (uODNS)</i>	92
7.2	<i>LISPP DoH (LDoH)</i>	93
7.2.1	Експериментална евалуација	94
7.3	Промена криптографског материјала	97
8	Закључак	102
	Литература	105
	Прилози	119
	А Листинзи за <i>Netronome LISPP</i>	119
	Б Листинзи за <i>eBPF LISPP</i>	123
	В Делови <i>DNS</i> лога	128

Списак слика

2.1	Формирање путање пакета код <i>LAP</i> система	11
2.2	Формирање путање пакета код <i>Dovetail</i> система	13
2.3	Формирање путање пакета код <i>PHI</i> система	15
2.4	Модификација поља заглавља пакета применом <i>PANEL</i> система	17
2.5	Скривање адресе код <i>AHP</i> система	19
2.6	Модификација поља заглавља пакета применом <i>AHP</i> система	20
2.7	Шифровање адреса код <i>SPINE</i> система	22
2.8	Модификација поља заглавља пакета применом <i>SPINE</i> система	23
2.9	Шифровање адреса код <i>PINOT</i> система	25
2.10	Модификација поља заглавља пакета применом <i>PINOT</i> система	26
3.1	Разлика између блоковске шифре и шифровања са очувањем формата	32
3.2	Четири фајстелове рунде алгоритама <i>FF1</i> и <i>FF3-1</i>	33
3.3	Четири фајстелове рунде алгоритама <i>FEA-1</i> и <i>FEA-2</i>	40
4.1	Модификација поља заглавља пакета на ивици мреже применом <i>LISPP</i> система	49
4.2	Дијаграм <i>LISPP</i> обраде пакета	49
4.3	Шифровање изворишне адресе и изворишног порта применом <i>LISPP</i> система	50
4.4	Емпиријска вероватноћа појављивања различитих вредности хост дела <i>IP</i> адресе коришћењем <i>LISPP</i> система за само једну <i>IP</i> адресу са маском /21 и за опсег вредности изворишног порта	51
4.5	Шифровање 8 узастопних <i>DNS</i> упита који долазе од уређаја са <i>IP</i> адресом 147.91.12.136/21 применом <i>LISPP</i> система	52
5.1	Еволуција мрежних уређаја	55
5.2	Разлика између <i>P4₁₄</i> и <i>P4₁₆</i> дијалеката	56
5.3	<i>PSA P4</i> архитектура	57
5.4	<i>v1model P4</i> архитектура	58
5.5	Архитектуре класичне <i>BPF</i> и <i>eBPF</i> виртуелне машине	62
5.6	Прикључци за <i>eBPF</i> програме на путањи пакета	65
6.1	Поставка за тестирање <i>Netronome LISPP</i> система	76
6.2	Проток пакета, кашњење пакета у једном смеру и пропусни опсег са <i>FF3-1</i> на <i>Micro-C</i> језику за пакете различитих величина код <i>Netronome LISPP</i> система	77
6.3	Стандардно одступање емпиријски уочене фреквенције појављивања свих могућих шифрованих <i>IP</i> адреса за различит број <i>AES</i> рунди и <i>DFT</i> амплитуда низа шифрованих адреса добијених коришћењем <i>FF3-1</i> са три <i>AES</i> рунде код <i>Netronome LISPP</i> система	79
6.4	Проток пакета и кашњење у једном смеру са умањеним бројем <i>AES</i> рунди код <i>Netronome LISPP</i> система	79
6.5	Поставка за тестирање <i>eBPF LISPP</i> система	80

6.6	Проток пакета, кашњење у једном смеру и пропусни опсег за пакете различитих величина код <i>eBPF LISPP</i> система	81
7.1	Дијаграм секвенце разрешавања назива домена помоћу <i>DNS</i> сервиса	85
7.2	Принцип рада <i>Do53</i> сервиса	86
7.3	Принцип рада <i>DoT</i> сервиса	87
7.4	Принцип рада <i>DoH</i> сервиса	88
7.5	Принцип рада <i>ODNS</i> сервиса	90
7.6	Принцип рада <i>ODoH</i> сервиса	91
7.7	Принцип рада <i>LDoH</i> сервиса	94
7.8	Поставка за тестирање <i>LDoH</i> и <i>ODoH</i> система	95
7.9	Поређење <i>LDoH</i> и <i>ODoH</i> система у погледу <i>RTT</i> потребног за изоловано разрешење једног назива домена зависно од кашњења унетих између клијента и посредника (d_{12}) и између посредника и разрешивача (d_{23})	96
7.10	Поређење <i>LDoH</i> и <i>ODoH</i> система у погледу просечног <i>RTT</i> потребног за разрешење једног назива домена приликом налета упита, а зависно од кашњења унетих између клијента и посредника (d_{12}) и између посредника и разрешивача (d_{23})	97
7.11	Број активних разрешења током читавог дана	99
7.12	Број активних разрешења током различитих периода дана	99
7.13	Број активних разрешења током вршне вредности	100
7.14	Периоди без активних разрешења	100

Списак табела

2.1	Поређење постојећих решења по кључним карактеристикама	29
3.1	<i>FEA SBox</i>	41
3.2	<i>FEA</i> 8×8 <i>MDS</i> матрица над пољем Галоа $GF(2^8)$	41
3.3	<i>FEA-1</i> константе рунде за различите дужине криптографског кључа	43
3.4	<i>FEA-2</i> константе рунде за различите дужине криптографског кључа	44
5.1	Типови меморија код <i>Netronome Agilio CX</i>	68
6.1	<i>TCP</i> пропусни опсег за три различите <i>Netronome LISPP</i> имплементације . .	76
7.1	Статистички параметри времена разрешења и броја активних сесија	98

1. Увод

Утицај комуникационих апликација и услуга на наше животе, као и ослањање на интернет, свакодневно расту. Овај тренд је праћен бројним доказима да се употреба интернета и апликација опсежно прати и анализира. Већина пружалаца веб услуга константно покушава да прати своје кориснике и прикупи што више њихових података о личности, на основу ствари које корисници претражују, веб страница које посећују, људи са којима комуницирају, производа које купују итд. Уобичајено уверење јесте да се овакво праћење углавном користи зарад пуког циљаног оглашавања, али прикупљени подаци о личности користе се за многе друге намене, као што су персонализација резултата претраге базирана на ранијем прегледању веба, процена кредитног рејтинга на основу претходних куповина и круга пријатеља, ценовна дискриминација према географској локацији и финансијском стању, одобравање осигурања за одређени ризик зависно од навика и животног стила, надгледање од стране државних безбедносних агенција и крађа идентитета [1].

Услед тога су заштита приватности и анонимизација коришћења интернета важне теме истраживања већ неколико деценија. Иако корисник може својеволно предати податке о личности, попуњавањем одговарајућих веб форми, много озбиљнији проблем по заштиту приватности јесте индиректно прикупљање података о личности без знања корисника. Неки од најчешће коришћених начина за прикупљање података о личности корисника, без његовог знања, су непосредни напади на апликације и веб прегледаче (енгл. *web browser*), посматрање садржаја заглавља протокола за пренос хипертекста (енгл. *hypertext transfer protocol*, *HTTP*) и заглавља интернет протокола (енгл. *internet protocol*, *IP*) или кроз анализу образаца мрежног саобраћаја и протокола које дати корисник директно или индиректно користи. Најчешће прикупљани подаци о личности су информације о коришћеном веб прегледачу, оперативном систему, уређају, *IP* адреси, али и друге осетљиве информације као што су географска локација корисника, његове преференце, историја посећених веб страница, итд.

Приватност веб прегледача је предмет бројних истраживања, која се углавном баве праћењем од стране веб сервиса и нападима на сам веб прегледач. Најраспрострањенији методи угрожавања приватности обухватају прикупљање података путем колачића (енгл. *cookie*), али и напредније технике креирања јединственог отиска уређаја и веб прегледача (енгл. *fingerprinting*) које се ослањају на специфичне техничке карактеристике система и начин извршавања програмског кода на клијентској страни. Детаљан преглед и анализа свих поменутих вектора напада, укључујући и механизме који омогућавају профилисање корисника без њиховог знања или сагласности, биће изложени у поглављу 2. Поред претходно описаних ризика по приватност корисника, наметнутих од стране веб страница и веб сервиса, незанемарљив извор угрожавања приватности могу представљати и сами веб прегледачи. Већина комерцијалних веб прегледача у позадини шаље значајну количину података на своје сервере [2], што представља потенцијални ризик по заштиту приватности. Иако слање техничких информација, као што су модел и верзија уређаја, само по себи не мора да угрози приватност, проблем настаје када се ови подаци могу директно повезати са конкретним корисником. Нарочито је осетљиво слање историје прегледања, јер она омогућава једноставну деанонимизацију, посебно у случајевима када су познате *IP* адресе

корисника [3]. Чак и у одсуству експлицитног слања историје прегледања, приватност може бити компромитована ако свака инстанца веб прегледача поседује свој јединствени идентификатор. Овакав идентификатор, у комбинацији са осталим подацима, омогућава повезивање различитих сесија и праћење активности једног корисника током времена. Поврх тога, неки веб прегледачи користе претходно описане механизме за прикупљање података, као што су отисци уређаја и мрежни параметри, чиме се додатно повећава могућност профилисања и праћења корисника без његовог знања или пристанка.

Претходно описане технике угрожавања заштите приватности претежно су примењене од стране сервиса високог нивоа апстракције односно сервиса непосредно сврстаних у апликативни ниво референтног модела за отворено повезивање система (енгл. *Open Systems Interconnection, OSI*). Успешност ових техника, које у великој мери угрожавају заштиту приватности корисника интернета, могуће је ублажити пажљивим избором веб прегледача у погледу његових сигурносних гаранција, уклањањем колачића, превенцијом извршавања јаваскрипт (енгл. *JavaScript*) кода, аскетском употребом веб сервиса итд. Ипак, ризик по заштиту приватности корисника постоји у великој мери и поред свих поменутих мера, а потиче од нижег нивоа апстракције односно од информација доступних на мрежном нивоу *OSI* модела.

Прикупљање делова заглавља мрежних пакета само по себи омогућава откривање значајне количине података о личности и директно угрожава приватност корисника интернета [4]. Претпоставимо да корисник интернета комуницира користећи сталну јавну *IP* адресу или да се може недвосмислено повезати са неком јавном *IP* адресом у одређеном периоду. На основу *IP* адресе корисника може се открити његова локација [5]. Нападачи могу пратити понашање и навике корисника надзирући скуп посећених *IP* адреса [6]. Још прецизније информације могу се добити праћењем корисникових захтева систему назива домена (енгл. *domain name system, DNS*) [7]. Недавна студија је показала да је приватност значајног дела крајњих корисника и даље угрожена [8], упркос томе што *IPv6* има одређене механизме заштите приватности, као што су ротација префикса и *IPv6* проширења за приватност. Због тога Општа уредба о заштити података (енгл. *general data protection regulation, GDPR*) сматра *IP* адресе информацијама које омогућавају идентификацију (енгл. *personally identifiable information, PII*) [9], чиме се придаје посебна пажња њиховој заштити. Из управо наведених разлога, ова докторска дисертација се фокусира на заштиту приватности корисника кроз превенцију профилисања и цурења личних података управо на мрежном слоју *OSI* модела.

Истраживање представљено у овој докторској дисертацији заснива се на неколико полазних хипотеза наведених у наставку. Прва је да постојећи механизми заштите приватности на мрежном слоју, описани у поглављу 2, имају проблеме у погледу перформанси услед интензивне примене алгоритама шифровања или чувања стања током рада, што их чини неповољним за примену у реалним мрежним условима. Друга хипотеза јесте да је успостављање корелације између пакета посматраног корисника могуће отежати замагљивањем (енгл. *obfuscation*) делова заглавља пакета са информацијама које омогућавају идентификацију, попут *IP* адреса. Затим, циљ докторске дисертације је да се покаже да употреба блоковских алгоритама шифровања у циљу замагљивања делова заглавља пакета није прикладна јер није могуће постићи довољно фину грануларност замагљивања, што доводи до нарушавања стандардног формата тих заглавља или до потребе за сложеним праћењем стања активних сесија. Насупрот томе, хипотеза је и да се недавно стандардизовани алгоритми шифровања са очувањем формата (енгл. *format preserving encryption, FPE*), изложени у оквиру поглавља 3, могу ефикасно користити за заштиту приватности на мрежном слоју и спречавање профилисања корисника, те да програмабилни мрежни уређаји омогућавају реализацију заштите приватности на мрежном слоју коришћењем *FPE* уз минимално нарушавања перформанси. На крају, циљ је да се докаже да је реализација

заштите приватности на мрежном слоју коришћењем стандардизоване *FPE* ефикаснија од постојећих механизма за заштиту приватности, без жртвовања сигурности комплетног система.

Истраживање спроведено у склопу ове докторске дисертације разликује се у неколико важних тачака од постојећих предлога за заштиту приватности на мрежном слоју. Реч је о првој студији која истражује употребу *FPE* алгоритама за заштиту приватности на мрежном слоју. Пројектовањем система без чувања стања за заштиту приватности (енгл. *Lightweight Stateless Privacy Protection, LISPP*) [10], чији су детаљи изложени у поглављу 4, ова докторска дисертација представља први предлог примене *FPE* за заштиту поља заглавља мрежних пакета. За разлику од неких постојећих предлога, предложени *LISPP* систем не чува стање током свог рада, што имплицира ниске меморијске захтеве, подршку за лако повезивање на већи број мрежа и слање/примање порука по различитим путевима (енгл. *multihoming*), потпуну транспарентност и рад на нивоу брзине везе (енгл. *line-rate*) коришћењем актуелних програмабилних мрежних уређаја, што је описано у наставку докторске дисертације. *LISPP* ради на мрежном слоју, независан је од протокола, може да заштити приватност података заглавља и *IPv4* и *IPv6* пакета, што није карактеристика до сада предложених система. Успешна и ефикасна имплементација, демонстрирана кроз поглавље 5, требало би да отвори пут за даљу употребу *FPE* у мрежним системима за заштиту приватности свих апликација, а не само веба.

Перформансе имплементираниог *LISPP* система доказане су на стварним уређајима у реалном мрежном окружењу експерименталном евалуацијом чији су резултати изложени у поглављу 6. *LISPP* имплементација за програмабилну мрежну картицу *Netronome Agilio CX* има пропусни опсег (енгл. *bandwidth*) по трансмисионом контролном протоколу (енгл. *transmission control protocol, TCP*) на нивоу брзине везе, остварује проток пакета (енгл. *packet-rate*) до 4,5 Мpps, са пеналом од свега 30 μ s до 60 μ s додатног кашњења (енгл. *latency*) у једном смеру у условима описаним у наставку докторске дисертације. Такав пропусни опсег на приступачној паметној мрежној картици (енгл. *smart network interface cards, SmartNIC*) [11, 12] доказује да је *FPE* адекватан за заштиту приватности на мрежном слоју. Поред имплементације за *SmartNIC* уређај, *LISPP* је реализован и коришћењем *eBPF* технологије како би се омогућила његова примена без потребе за специјализованим хардвером. *LISPP* имплементација за *eBPF* технологију има пропусни опсег по трансмисионом контролном протоколу до 4,88 Gb/s, остварује проток пакета до 2,96 Мpps, са пеналом од само 5 μ s до 10 μ s додатног кашњења у једном смеру. Применом *LISPP* постиже се убрзање разрешења *DNS* упита и до 45% у односу на *ODoH* (енгл. *Oblivious DNS over HTTPS*) уз задржавање истог нивоа заштите приватности. Ова флексибилност омогућава ширу примену *LISPP* система у различитим мрежним окружењима.

Структура докторске дисертације је описана у наставку. Проблем заштите приватности на интернету и преглед постојећих решења у области заштите приватности на мрежном слоју, са посебним освртом на проблеме постојећих предложених решења као последица употребе класичних алгоритама за шифровање, наведен је у поглављу 2. Алгоритми за шифровање са очувањем формата описани су унутар поглавља 3. Архитектура и принципи заштите приватности коришћењем *LISPP* система представљени су поглављем 4. Детаљи и изазови имплементације *FPE* на мрежним акцелераторским картицама и *eBPF* технологији истакнути су у оквиру поглавља 5. Перформансе *LISPP* система и добијени експериментални резултати изложени су у виду поглавља 6. Случајеви коришћења *LISPP* система зарад заштите приватности у контексту *DNS* дискутовани су кроз поглавље 7, док поглавље 8 доноси закључак рада.

2. Проблем заштите приватности

Тема овог поглавља је проблем заштите приватности корисника у контексту савремених рачунарских комуникација. Након уводног разматрања изазова заштите приватности на апликативном слоју и у оквиру веб комуникације, тежиште излагања се помера ка проблему заштите приватности на мрежном слоју. Поглавље се закључује свеобухватном компаративном анализом постојећих решења за заштиту приватности на мрежном слоју, на основу које су изведени закључци о њиховим предностима и ограничењима.

2.1 Заштита приватности на апликативном слоју

Најпознатији метод нарушавања приватности корисника на интернету од стране веб сервиса јесте прикупљање података о личности омогућено чувањем мале количине података, познате под именом колачић, унутар веб прегледача. Приликом прве посете веб страници, *HTTP* захтев корисника неће садржати колачић, на основу чега веб сервер може претпоставити да корисник није раније посетио дату веб страницу. У том случају, веб сервер генерише јединствени идентификатор корисника и шаље га назад у оквиру колачића који веб прегледач треба да сачува. Надаље, сваки пут када корисник поново посети ту веб страницу, веб прегледач у оквиру *HTTP* захтева шаље и садржај колачића односно јединствени идентификатор корисника. На овај начин, веб сервер добија могућност да прати конкретног корисника означеног јединственим идентификатором и прикупља податке о његовом понашању на веб страници [13].

Веб сервиси, који врше праћење, могу пратити корисника и на другим регуларним веб страницама захваљујући колачићима треће стране (енгл. *third-party cookies*), под условом да регуларне веб странице унутар себе имају уграђене делове управо тих веб сервиса [14, 15, 16]. На пример, сваки пут када корисник посети неку веб страницу са уграђеном твитер објавом, твитер ће бити обавештен који корисник је посетио посматрану веб страницу слањем јединственог идентификатора датог корисника из колачића. Читав процес чувања и накнадног слања колачића не захтева никакву интеракцију од стране корисника и притом је потпуно транспарентан за корисника јер веб прегледач не приказује никаква обавештења. Европска унија је усвојила *GDPR*, чији је један од захтева да веб страница ипак мора обавестити корисника о политици чувања колачића, као и да мора добити сагласност корисника за чување свих колачића са изузетком технички неопходних колачића. Усвајање ове уредбе није донело суштинску промену у погледу смањења размера праћења корисника путем колачића [17, 18, 19].

Праћење корисника путем колачића је изводљиво све док корисник користи исту инстанцу веб прегледача, где је сачуван колачић са његовим јединственим идентификатором, и док експлицитно не обрише колачиће са свог уређаја односно из датог веб прегледача. По донекле сличном принципу, за смештање јединственог идентификатора корисника и његово касније дохватање, могу се користити и сесијско складиште података (енгл. *session storage*) и локално складиште података (енгл. *local storage*) унутар веб прегледача. Напреднији методи праћења корисника од стране веб сервиса користе јаваскрипт код

извршаван на клијентској страни, зарад приступа различитим техничким детаљима веб прегледача и оперативног система на којем се дати јаваскрипт код извршава. Ови технички детаљи могу бити искоришћени за стварање отиска корисника који представља јединствени идентификатор корисника формиран на основу детаља мреже, уређаја, оперативног система или веб прегледача.

Према томе, зависно од детаља коришћених за формирање отиска, могу се разликовати отисак мреже (енгл. *network fingerprint*), отисак уређаја (енгл. *device fingerprint*), отисак оперативног система (енгл. *operating system fingerprint*) и отисак веб прегледача (енгл. *browser fingerprint*). Отисак мреже сачињавају IP адреса, географска локација и карактеристике мрежне конекције као што су брзина преузимања и отпремања података, времена путовања пакета у оба смера (енгл. *round-trip time, RTT*) и неравномерност (енгл. *jitter*) односно одступање од вредности. Главни елементи отиска уређаја јесу подаци као што су верзија оперативног система, резолуција екрана, временска зона и листа доступних текстуалних фонтова. Слично, верзија и архитектура оперативног система, системски језик и временска зона дефинишу отисак оперативног система.

Отисак веб прегледача (енгл. *browser fingerprinting*) омогућава идентификацију корисника на интернету [20], при чему је сам отисак могуће формирати прикупљањем широког спектра података преко апликативног програмског интерфејса веб прегледача. Постоји велики број постојећих алата за прикупљање података на основу којих се може формирати овај отисак [21, 22, 23]. Управо услед велике разноликости модерних уређаја и веб прегледача могуће је једнозначно идентификовати кориснике [24]. Отисак веб прегледача може се формирати на више различитих начина од којих су најчешћи утврђивање скупа доступних функционалности веб прегледача, цртање на платну (енгл. *canvas*) програмским путем, увид у историју прегледања и излиставање инсталираних додатних модула (енгл. *plugin*) веб прегледача.

Утврђивање скупа доступних функционалности веб прегледача базира се на испитивању постојања подршке за различита својства, селекторе и филтере каскадног описа стилова (енгл. *cascading style sheets, CSS*) или подршке за различите тагове језика за означавање хипертекста (енгл. *hypertext markup language, HTML*). Постојање подршке за наведене функционалности разликује се од веб прегледача до веб прегледача, тако да конкретан скуп доступних функционалности представља довољан диференцијатор на основу којег се може дефинисати јединствени отисак веб прегледача. Цртање на платну подразумева извршавање једног истог јаваскрипт кода на различитим веб прегледачима и формирање нумеричке представе резултата цртања зарад дефинисања отиска веб прегледача [25]. Иако је увек реч о истом јаваскрипт коду, резултат цртања се разликује услед специфичности оперативног система, инсталираних фонтова, графичке карте и њених драјвера, као и самог веб прегледача. Нумеричка представа резултата цртања може бити Base64 енкодован низ знакова, чије формирање такође у извесној мери зависи од веб прегледача. Захваљујући управо наведеним диференцијаторима, може се дефинисати јединствени отисак веб прегледача на основу нумеричке представе резултата цртања.

Временски напади (енгл. *timing attack*) преко бочних канала могу открити информације попут историје прегледања [26]. Овакви напади заснивају се на мерењу времена потребног за преузимање ресурса; раније посећене веб странице налазе се у кешу, услед чега ће време потребно за њихово преузимање бити значајно смањено. Стога, када корисник приступи веб страници, мерењем времена потребног за преузимање ресурса може се установити да ли је веб страница била раније посећена. На основу историје прегледања веб страница, могуће је дефинисати јединствени отисак веб прегледача анализом да ли су и које тачно веб странице из предефинисаног скупа раније посећене [27, 28]. Коришћењем отиска веб прегледача, корисник може бити праћен на различитим веб страницама које припадају различитим ентитетима. Слично као у случају колачића, процес формирања отиска веб прегледача је

такође транспарентан за корисника, али за разлику од колачића корисник није у могућности да обрише отисак са свог уређаја због тога што се отисак формира извршавањем јаваскрипт кода на клијентској страни и не имплицира складиштење никаквих података. Управо због тога отисак веб прегледача представља озбиљнији проблем по заштиту приватности у поређењу са колачићима.

2.2 Заштита приватности веб комуникације

С обзиром на то да су веб апликације најпопуларније интернет услуге које се данас користе, појавиле су се разне виртуелне приватне мреже (енгл. *virtual private network, VPN*) и посредничке (енгл. *proxy*) услуге које омогућавају скривање оригиналне изворишне *IP* адресе приликом посете веб страница. Међутим, овакви системи имају једну тачку отказа и ослањају се на поверење у пружалаца *VPN* и/или посредничких услуга, који може бити под страном јурисдикцијом. Систем за *етапно рутирање* (енгл. *onion routing*) са ниским кашњењем, *Tor* [29, 30], вероватно је најпознатији и најшире коришћени систем за анонимизацију на мрежном слоју. Он омогућава анонимно прегледање веб страница кроз коло етапних рутера (енгл. *onion routers circuit*), које се састоји од три етапна рутера уз примену троструког шифровања садржаја пакета. *Tor* такође омогућава приступ скривеном веб садржају.

Коришћењем *Tor* система, одредишни веб сервер и посматрачи на било којој појединачној тачки интернета не могу да утврде оба краја веб сесије, чиме се одржава анонимност корисника на мрежном слоју. Међутим, оваква заштита долази са извесним пеналима у погледу перформанси. Сваки пакет је подељен на *Tor* ћелије са додатним заглављима, чиме се директно смањује користан део појединачног пакета. Обрада сваке ћелије укључује три шифровања и дешифровања између крајњих тачака комуникације. Путања пакета између крајњих тачака комуникације намерно није оптимална. Сви ови пенали заједно често доводе до спорог и заморног прегледања веб [31, 32, 33]. Такође, упркос интензивној употреби шифровања, познато је да *Tor* систем јесте подложен временским нападима [34, 35, 36], као и нападима одметнутих *Tor* рутера. Временски напади подразумевају посматрање и праћење саобраћаја на више тачака у мрежи и довођење образаца саобраћаја у корелацију у циљу откривања крајњих тачака комуникације [37]. Спровођење временских напада додатно је олакшано уколико нападач има увид у *DNS* упите и одговоре [38].

Мешајуће мреже (енгл. *mix networks*) [39, 40, 41] представљају старије и робусније решење у поређењу са *Tor* системом, посебно у погледу напада анализом временских образаца саобраћаја. Мешајуће мреже сачињене су од скупа *мешача* (енгл. *mixes*) помоћу којих се постиже анонимно прослеђивање пакета између изворишта и одредишта. Слично као код *Tor* система, путању пакета бира извориште, али може укључивати произвољни број мешача на путу до одредишта. Како би се обезбедила неповезивост између изворишта и одредишта, како у погледу изгледа пакета тако и у погледу временских образаца саобраћаја, сваки мешач криптографски трансформише примљене пакете и мења редослед њиховог прослеђивања даље ка одредишту.

Криптографска трансформација, зарад спречавања успостављања корелације између пакета на улазу и излазу појединачних мешача, реализује се помоћу етапног шифровања од стране сваког мешача на путањи пакета. Додатно, промена редоследа прослеђивања пакета ка одредишту, у односу на метод *први дошао, први изашао* (енгл. *first in, first out, FIFO*), спречава временску корелацију пакета. Ова промена редоследа постиже се додавањем псеудослучајног кашњења сваком појединачном пакету, као и скупљањем већег броја улазних пакета и њиховим истовременим групним прослеђивањем даље. Овим приступом мешајуће мреже могу гарантовати анонимност комуникације чак и у случају нападача

са могућношћу глобалног надгледања саобраћаја, што представља слабост *Tor* система. Међутим, овде се јасно уочава компромис између анонимности и перформанси; што је веће додатно унето кашњење за сваки појединачни пакет, то је боља заштита приватности, али су перформансе лошије. Промена редоследа прослеђивања пакета ка одредишту представља проблем за *TCP* механизам детекције загушења, услед чега се смањује величина *TCP* прозора, а последично долази и до смањења протока. Управо због значајног кашњења из наведених разлога, мешајуће мреже у општем случају су практично неупотребљиве за радње у реалном времену због неприхватљиво лоших перформанси.

2.3 Заштита приватности на мрежном слоју

Прошло је више од 20 година од настанка *Tor* система и више од чак 40 година од дефинисања концепта мешајућих мрежа, а неки обрасци коришћења интернета су се од тада променили. Данас је готово сав веб саобраћај шифрован [42], услед чега се поставља питање да ли су додатни слојеви шифровања података у заглављима уопште потребни и оправдани јер не пружају додатну заштиту од временских напада и одметнутих *Tor* рутера. Велики број слабости [43] *Tor* система инспирисао је нове предлоге за обезбеђивање анонимности на мрежном слоју.

Програмабилни мрежни уређаји, као што су свичеви са мрежним акцелераторима по принципу транспарентне архитектуре (енгл. *bump-in-the-wire*) или *SmartNIC*, постали су веома популарни у последњој деценији међу мрежним професионалцима, при чему је језик *P4* једна од најзначајнијих иновација. Њихова програмабилност и флексибилност омогућили су иновације и такође подстакли истраживања у области заштите приватности. *SmartNIC* уређаји могу се програмирати користећи различите програмске језике и стилове (*Verilog*, *P4*, *C*, асемблер или њихове комбинације), чиме се део обраде мрежног саобраћаја може изместити са централних процесора. Програмабилност *SmartNIC* уређаја омогућава извршавање рачунских задатака и обраду мрежног саобраћаја ближе самој путањи података (енгл. *data path*), скраћујући тиме време обраде и обезбеђујући обраду саобраћаја великом брзином, као и примену нових апликација или измене пакета, без жртвовања перформанси мрежног саобраћаја.

Недавно објављен прегледни рад примењених истраживања у области програмирања равни података (енгл. *data plane*) [44] наводи најновије напоре у заштити приватности на мрежном слоју засноване на новим програмабилним мрежним уређајима. Принцип рада и карактеристике најзначајнијих постојећих система за заштиту приватности биће детаљно изложени у наставку овог поглавља, при чему ће свако решење бити представљено у оквиру посебне секције. Системи за заштиту приватности могу се грубо поделити на оне са *јаким гаранцијама приватности* и оне са *релаксираним гаранцијама приватности* зависно од тога да ли пружају виши или нижи ниво заштите приватности, респективно. Мешајуће мреже пружају највиши ниво заштите приватности, захваљујући отпорности на анализу временских образаца саобраћаја, тако да спадају у системе са јаким гаранцијама приватности. *Tor*, као систем заснован на етапном рутирању, такође спада у системе са јаким гаранцијама приватности, али је подложен временским нападима. Системи са релаксираним гаранцијама приватности генерално су подложни нападима од стране нападача са мањим могућностима у поређењу са глобалним надгледањем саобраћаја и неретко захтевају безусловно поверење у тополошки блиске ентитете.

Заједничко за већину система за заштиту приватности јесте да представљају решења са већим бројем чворова (енгл. *nodes*) односно рутера и/или сервера који сарађују дуж путање пакета. Немали број њих се притом за унапређење заштите приватности ослања на технологије нове генерације интернета (енгл. *next generation Internet*) [45], због чега

се њихова широко распрострањена примена може реализовати тек у будућности након потенцијалне промене архитектуре интернета извршене на одговарајући начин. Ипак, вероватноћа да дође до корените промене архитектуре интернета је изузетно мала услед његове величине и инертности коју намеће координација више од 100 000 аутономних система, милиона рутера и милијарди повезаних уређаја. Очекивање да се архитектура интернета промени преко ноћи је тек илузорно, а самим тим је упитна и могућност усвајања решења заснованих на промени постојећих базичних протокола.

2.3.1 *HORNET*

Систем за брзо етапно рутирање на мрежном слоју (енгл. *High-speed Onion Routing at the Network Layer, HORNET*) [46] формира анонимни канал између крајева комуникације. Исто као и *Tor*, захваљујући етапном рутирању, *HORNET* спада у системе са јаким гаранцијама приватности. Приликом успостављања анонимне сесије користи симетричне и асиметричне криптографске алгоритме, док се у оквиру процеса прослеђивања пакета са подацима примењују искључиво симетрични криптографски алгоритми ради постизања што већег пропусног опсега система. Извориште мрежног саобраћаја бира путању пакета.

HORNET се ослања на функционалност мрежних елемената односно чворова да на основу сопственог стања рутирања могу за сваки пакет одредити искључиво наредни чвор у путањи, док је крајње одредиште познато једино последњем чвору. Зарад обезбеђивања ове функционалности неопходно је користити сегментно рутирање (енгл. *segment routing*) [47] или архитектуре нове генерације интернета као што су *NIRA* [48], *SCION* [49], *Pathlet* [50]. Претпоставка аутора јесте да слој испод *HORNET* система омогућава изворишту да дохвати стање рутирања свих чворова на путу пакета, као и њихове јавне кључеве и сертификате на начин који не угрожава анонимност самог изворишта. За формирање симетричних кључева сесије, дељених између изворишта и сваког појединачног чвора на путањи пакета, користи се модификовани *Sphinx* протокол [51]. Као представник мешајућих мрежа, *Sphinx* протокол уноси значајно кашњење у комуникацију, али је олакшавајућа околност што се користи само приликом успостављања сесије.

HORNET не чува стање сесије на посредним чворовима дуж путање пакета. Стање сесије, чији су најбитнији елементи криптографски материјал и путања пакета, чува се на крајевима комуникације и преноси у оквиру заглавља пакета да би било доступно посредним чворовима. Услед преноса стања сесије долази до повећања заглавља пакета, што резултира повећањем удела режијског саобраћаја односно директним смањењем корисног капацитета везе. Како би се спречило цурење осетљивих детаља стања сесије из заглавља пакета, сваки чвор *HORNET* система користи свој локални симетрични кључ, који никада не напушта дати чвор, за шифровање и дешифровање стања сесије. Шифровањем стања сесије коришћењем локалног симетричног кључа датог чвора, добија се сегмент прослеђивања (енгл. *forwarding segment*) који се уграђује у заглавље пакета као део стања сесије. Након успостављања сесије користи се само тип пакета предвиђен за слање података. Идентична заглавља пакета са стањем сесије се поновно употребљавају за слање сваког пакета са подацима, током читавог трајања сесије, што *HORNET* систем чини подложним нападу понављањем (енгл. *replay attack*). Садржај пакета са подацима се шифрује у више слојева, али његов интегритет није заштићен.

Начелни циљ *HORNET* система јесте да обезбеди анонимност односа (енгл. *relationship anonymity*) [52] чиме онемогућава нападача да препозна парове уређаја који међусобно комуницирају. Конкретно, могуће је сачувати или искључиво анонимност изворишта или истовремену анонимност оба краја комуникације. Користан пропусни опсег система је 93,5 Gb/s на уређајима са пропусним опсегом од 120 Gb/s. Према речима аутора, *HORNET* је подложен временским нападима заснованим на анализи мрежних токова.

2.3.2 TARANET

Систем за очување анонимности на мрежном слоју отпоран на анализу мрежног саобраћаја (енгл. *Traffic-Analysis Resistant Anonymity at the Network Layer, TARANET*) [53] у великој мери се заснива на принципима рада *HORNET* система и представља његову својеврсну надоградњу што га дефинитивно сврстава међу системе са јаким гаранцијама приватности. *TARANET* систем, додатно у односу на *HORNET* систем, пружа заштиту интегритета садржаја пакета са подацима и заштиту од напада понављањем. Приликом успостављања сесије користи мешајуће мреже ради очувања анонимности. У циљу спречавања анализе мрежног саобраћаја, *TARANET* врши обликовање саобраћаја (енгл. *traffic shaping*) са ограниченим кашњењем.

Извориште започиње процес успостављања сесије тако што анонимно дохвата информације о путањи у које спадају сегменти рутирања, јавни кључеви и сертификати свих чворова између крајева комуникације. Информације о путањи се дохватају за оба смера комуникације; за смер од изворишта ка одредишту и у обрнутом смеру. Анонимно дохватање информација о путањи могуће је остварити помоћу истих архитектура нове генерације интернета на које се ослања и *HORNET* систем: *NIRA*, *SCION* и *Pathlet*. Друга опција за анонимно дохватање информација о путањи јесте слање упита ка једном од сервера са детаљима глобалне топологије *TARANET* система; у овом случају анонимност комуникације је обезбеђена ослањањем на независну *TARANET* анонимну сесију успостављену на основу информација из унапред задате конфигурације. Након добијања информација о путањама за оба смера, крајеви комуникације размењују припремне пакете преко управо датих путања. Приликом обраде припремних пакета, сваки *TARANET* чвор на путањи пакета формира симетрични кључ дељен са извориштем који се у току фазе прослеђивања података користи за шифровање у више слојева. Како не би морао да чува симетричне кључеве за сваку сесију, чвор шифрује дати симетрични кључ својим локалним тајним кључем, чиме се добија сегмент прослеђивања. Извориште уграђује сегмент прослеђивања у заглавље сваког пакета са подацима, тако да чвор долази до криптографског материјала за дату сесију операцијом дешифровања својим локалним тајним кључем, уместо претрагом великих табела стања.

Прихватањем сегмената рутирања, сегмената прослеђивања и симетричних кључева дате сесије од сваког чвора на путањи пакета, извориште је у могућности да формира пакете са подацима. Заглавље сваког појединачног пакета са подацима се модификује додавањем кода за аутентификацију поруке (енгл. *message authentication code*) који чува интегритет и заглавља и садржаја пакета са подацима. Садржај сваког појединачног пакета са подацима се шифрује у више слојева.

TARANET постиже обликовање саобраћаја уметањем шума односно лажног мрежног саобраћаја (енгл. *chaff traffic*), што мора довести до извесног нарушавања перформанси система [54]. Са друге стране, временски напади би требало да буду отежани, због тога што нападач не може лако разликовати лажни од стварног мрежног саобраћаја након што сви пакети прођу кроз фазу шифровања. Две најраспрострањеније праксе за генерисање лажног саобраћаја јесу повремено убацивање лажног саобраћаја између крајева комуникације и континуирано убацивање лажног саобраћаја, ради одржавања константног протока података између парова суседних чворова, у мери зависној од количине стварног мрежног саобраћаја. *TARANET* комбинује ова два приступа тако што намеће константни проток података на свим коришћеним везама између крајева комуникације. Ово се постиже увођењем редова резервних пакета (енгл. *spare packet queues*), на сваком појединачном *TARANET* чвору, из којег се пакети узимају и шаљу када постоји разлика у брзини пристизања пакета и устаљене брзине слања пакета. Попуњавање ових редова постиже се специфичном техником деобе пакета (енгл. *packet splitting*); криптографски механизам омогућава изворишту саобраћаја да генерише пакет од којег, на одређеним чворовима,

настају два различита пакета исте величине као и оригинални пакет. Деобом пакета извориште саобраћаја може попуњавати редове резервних пакета на посредним чворовима без нарушавања константног протока података.

Пропусни опсег система је 50 Gb/s на широко распрострањеним мрежним уређајима. Иако не чувају стање сесија експлицитно, *TARANET* чворови морају складиштити редове резервних пакета зарад потребе за обликовањем мрежног саобраћаја. Посматрајући везу пропусног опсега од 10 Gb/s и при протоку стварног саобраћаја од 10 kb/s, чвор треба да чува око 90 MB само за потребе редова резервних пакета.

2.3.3 LAP

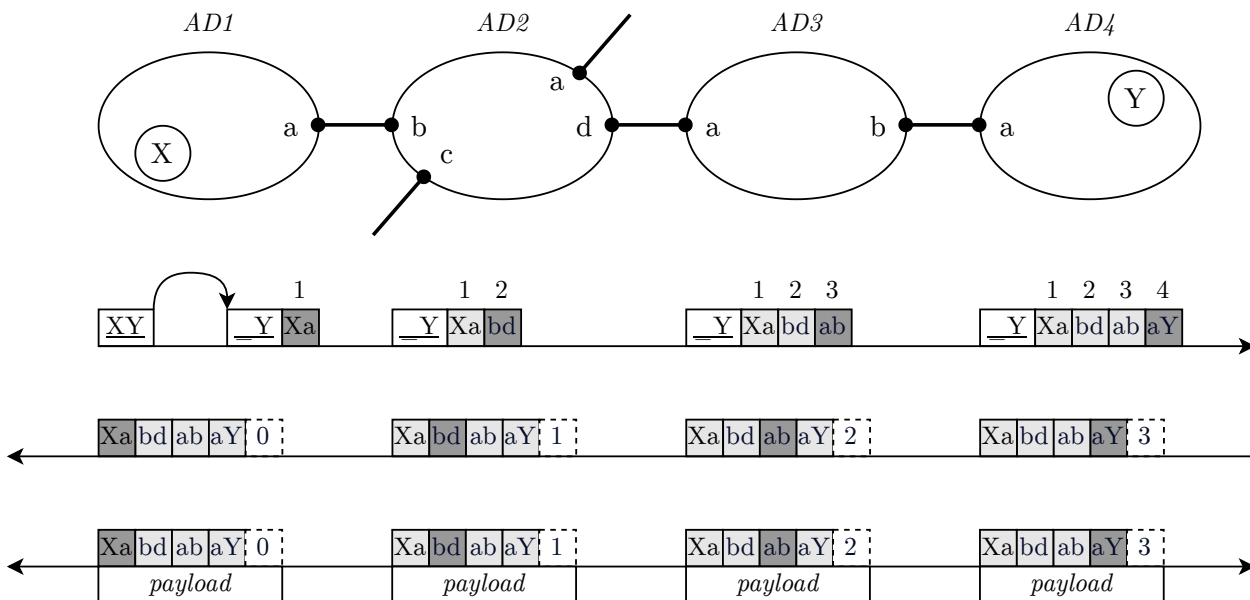
Систем за релаксирану анонимност и приватност (енгл. *Lightweight Anonymity and Privacy, LAP*) [55] пружа релаксиране гаранције приватности у циљу постизања што мањег кашњења пакета. За разлику од система са јаким гаранцијама приватности, *LAP* се фокусира само на заштиту идентитета и локације корисника, примарно ради спречавања праћења навика и понашања корисника од стране злонамерних сервера. *LAP* прикрива тополошку локацију крајева комуникације и нема за циљ да буде решење за заштиту анонимности и приватности против нападача са неограниченим ресурсима и могућношћу праћења целокупног мрежног саобраћаја на произвољним тачкама. Управо оваква релаксација модела претњи омогућава постизање бољих перформанси у поређењу са системима који пружају јаке гаранције приватности.

LAP уводи занемарљив трошак обраде на мрежним елементима и минимално кашњење пакета. Може бити реализован над екстерним протоколом рутирања или над архитектуром нове генерације интернета као што је *SCION* [56]. Успостављање сесије између крајева комуникације врши се ради формирања путање пакета односно стања прослеђивања (енгл. *forwarding state*). Дужина путање пакета, изражена у броју чворова, је минимално већа у поређењу са дужином регуларне путање пакета без примене *LAP* система. У оквиру *LAP* система, чвор представља један аутономни домен (енгл. *Autonomous Domain*).

Никакво стање сесије се не чува на мрежним елементима. *LAP* преноси стање прослеђивања у оквиру сваког пакета да би посредни чворови могли да одреде куда треба проследити дати пакет. Сваки посредни чвор својим наменским тајним кључем шифрује свој део стања прослеђивања како би се заштитила анонимност путање пакета односно крајева комуникације. Појединачни део стања прослеђивања шифрован наменским тајним кључем одговарајућег чвора назива се сегмент, а целокупна путања пакета састављена од низа шифрованих сегмената свих чворова назива се шифрована путања. Наменски тајни кључ за шифровање сегмената формира се на основу псеудослучајне једнократне вредности (енгл. *nonce*) дефинисане од стране изворишта и краткотрајног тајног кључа изведеног из главног тајног кључа датог чвора. Према томе, извориште има директан утицај на наменски тајни кључ других чворова и самим тим може обезбедити да се шифровани сегменти за исту путању разликују од сесије до сесије тако што варира псеудослучајну једнократну вредност. Наменски, краткотрајни и главни тајни кључеви никада не напуштају посматрани чвор, тако да нема потребе за разменом никаквог криптографског материјала између чворова.

Формирање шифроване путање започиње слањем захтева од изворишта ка одредишту што је илустровано на слици 2.1. У склопу обраде пакета захтева при напуштању првог чвора којем извориште припада, ивични мрежни елемент ће избацити *IP* адресу изворишта из поља заглавља за изворишну адресу, док ће додати нови шифровани сегмент са *IP* адресом изворишта и ознаком излазног интерфејса. Како се *IP* адреса изворишта више не налази у пољу заглавља за изворишну адресу, нити је нападач може реконструисати на основу шифрованог сегмента захваљујући примени криптографског алгоритма, овим кораком је надаље обезбеђена анонимност *IP* адресе изворишта. Чвор којем извориште

припада очигледно има увид у IP адресу изворишта, тако да LAP имплицитно намеће постојање поверења крајева комуникација у чворове којима припадају. Такође, нападач са било које тачке мреже може сазнати идентитет одредишта посматрањем пакета захтева у којем се налази IP адреса одредишта. Други чвор додаје свој шифровани сегмент са ознаком улазног и излазног интерфејса у пакет захтева и прослеђује га до наредног чвора на путањи пакета. Процес додавања шифрованих сегмената се овако наставља све док пакет захтева коначно не стигне до одредишта. Ради повећања тополошке анонимности могуће је извршити и проширење шифроване путање укључивањем додатних LAP чворова.



Слика 2.1: Формирање путање пакета код LAP система

Шифрована путања, формирана претходно описаним прослеђивањем и обрадом пакета захтева од стране свих чворова на његовој путањи, омогућава одредишту да пошаље пакет одговора назад ка изворишту без познавања његове IP адресе. Ово је могуће захваљујући томе што шифрована путања садржи одлуке о прослеђивању пакета, донете од стране сваког појединачног чвора на путањи пакета, чијим се репризирањем у обрнутом смеру пакет може проследити од одредишта назад ка изворишту. Поред формиране шифроване путање, пакет одговора мора садржати и вредност помераја за приступ до одговарајућег сегмента за чвор ка којем пакет тренутно путује. Пристизањем пакета одговора, извориште долази у посед шифроване путање, што му омогућава да анонимно шаље пакете са подацима уграђујући у дате пакете шифровану путању уместо IP адреса из којих се може открити идентитет крајева комуникације.

LAP је подложен тополошким нападима (енгл. *topology-based attacks*). Физичке везе одређују куда сваки чвор потенцијално може прослеђивати пакете. Део тих података о топологији свакако је јавно доступан [57] услед чега нападач може значајно смањити анонимни скуп односно може закључити где се извориште оквирно налази. Увидом у заглавље пакета, где се један за другим налазе шифровани сегменти исте величине као и индекс сегмента за обраду, угрожени или одметнути чвор може утврдити дужину дела путање од изворишта до себе. Извођење тополошког напада је додатно олакшано познавањем ове дужине путање од изворишта до нападача. Управо зато LAP има подршку за сегменте променљиве величине чиме је нападач онемогућен да простим пребројавањем сегмената закључи колико далеко од њега се извориште налази изражено у броју чворова. Ипак, сегменти променљиве величине могу осетно увећати величину заглавља пакета смањујући тиме користан пропусни опсег.

LAP систем није транспарентан за крајњег корисника. Ради заштите своје анонимности, крајњи корисник мора да инсталира наменску *LAP* апликацију. Задатак ове апликације јесте да за корисника формира шифровану путању на претходно описан начин.

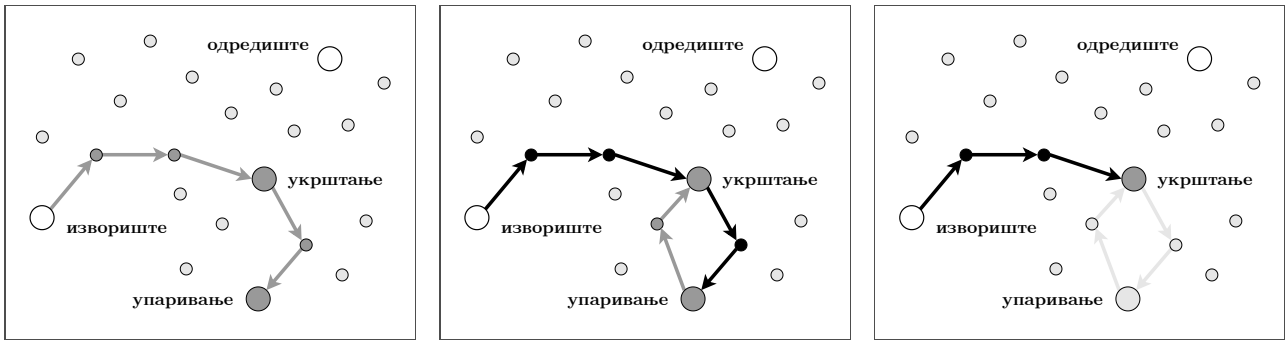
2.3.4 *Dovetail*

Dovetail [58] представља протокол рутирања нове генерације интернета са уграђеном заштитом анонимности од активног нападача позиционираног на било којој, али само једној, тачки мреже. Загарантовани ниво анонимности јесте немогућност довођења у везу (енгл. *unlinkability*) изворишта и одредишта саобраћаја [59], што значи да нападач са било које тачке мреже не може да закључи да ли су извориште и одредиште повезани. *Dovetail* је својеврсна надоградња *LAP* система јер не захтева поверење у пружаоца интернет услуга (енгл. *internet service provider, ISP*) крајњих тачака комуникације. Принцип рада *Dovetail* система заснива се на избегавању рутирања саобраћаја најкраћом путањом и распарчавању информација за рутирање на мрежном слоју. Ипак и поред ових принципа, *Dovetail* спада у системе са релаксираним гаранцијама приватности.

Рутирање саобраћаја најкраћом путањом се избегава јер ставља нападача у позицију из које може угрозити анонимност крајњих тачака комуникације мерењем дужине путање од изворишта до себе као и дужине путање од себе до одредишта. Уколико измерена дужина једне од ове две путање значајно одступа од просека, нападач може донети закључак о локацији крајњих тачака комуникације. Смештањем информација за рутирање у целости у једно поље заглавља пакета отежана је заштита њихове анонимности. Сви мрежни елементи, којима је доступно дато поље заглавља пакета, могу открити идентитет крајева комуникације; ово је случај када се за рутирање користи *IP* адреса. Управо из претходно наведених разлога код *Dovetail* система извориште бира путању пакета, што значи да путања може бити произвољна односно не мора бити најкраћа и притом може бити сачињена од већег броја делова уместо само од *IP* адресе крајњег одредишта. *Dovetail* систем обезбеђује могућност одабира путање пакета од стране изворишта ослањањем на *Pathlet* протокол рутирања нове генерације интернета. Директна импликација овога јесте да крајеви комуникације морају знати нумеричке ознаке *Pathlet* виртуелних чворова, што уноси значајне потешкоће у коришћењу система. Увођење система за разрешавање људски читљивих назива у нумеричке ознаке *Pathlet* виртуелних чворова може ублажити потешкоће у коришћењу *Dovetail* система, али имплементација оваквог система налик *DNS* није тривијалан задатак.

Dovetail путања пакета сачињена је од већег броја сегмената, исто као *LAP*, чиме се постиже распарчавање информација за рутирање. Извориште не формира директно путању пакета кроз до одредишта, што се види на слици 2.2, јер би аутономни систем којем извориште припада у том случају могао да повеже извориште и одредиште. Због тога се као индирекција користи чвор упаривања (енгл. *matchmaker node*) изабран на случајан начин и у који не мора постојати поверење. Извориште шифрује идентитет крајњег одредишта јавним кључем чвора упаривања и формира део путање пакета само до датог чвора упаривања као што је приказано на слици 2.2а; овај део путање назива се први односно почетни део путање. На овај начин аутономни систем којем припада извориште не може да сазна идентитет крајњег одредишта, већ то може да сазна само чвор упаривања који са друге стране не може да установи идентитет изворишта. Чвор упаривања формира наставак путање пакета од себе до одредишта као што је приказано на слици 2.2б; овај део путање назива се други односно крајњи део путање. Зарад постизања вишег нивоа заштите анонимности, саобраћај не би требало да пролази кроз чвор упаривања. Због тога почетни и крајњи део путање пакета морају имати један заједнички чвор укрштања (енгл. *dovetail node*). Поред идентитета крајњег одредишта, извориште шифрује и идентитет

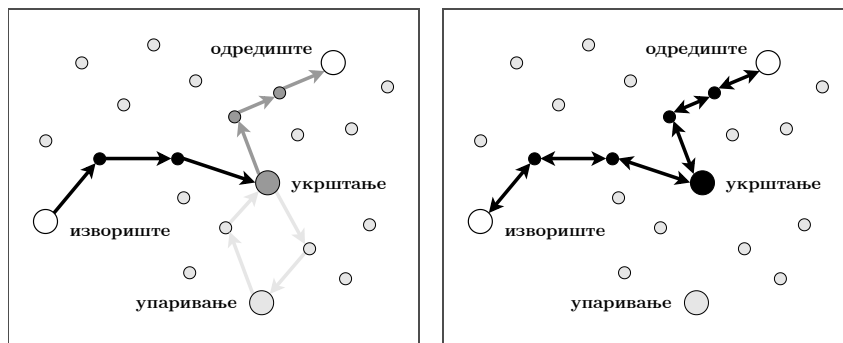
чвора укрштања како би га чвор упаривања укључио у крајњи део путање. Чвор укрштања препознаје преклапање почетног и крајњег дела путање пакета и избацује то преклапање, заједно са чвором упаривања, из коначне путање пакета што је илустровано на слици 2.2в. Како би се избегло понављање истог аутономног система у почетном и у крајњем делу путање пакета, притом без да се чвору упаривања откривају информације о аутономним системима који су део почетног дела путање, чвор упаривања у току успостављања сесије шаље изворишту скуп потенцијалних крајњих делова путање.



(а) Извориште формира почетни део путање до чвора упаривања

(б) Чвор упаривања започиње крајњи део путање

(в) Чвор укрштања детектује и уклања преклапање



(г) Чвор укрштања наставља крајњи део путање до одредишта

(д) Путања пакета успостављена између крајева комуникације

Слика 2.2: Формирање путање пакета код *Dovetail* система

Dovetail разликује пет типова пакета: обичан пакет, пакет за формирање путање, пакет са резултатом формирања путање, пакет шифрованих података и пакет шифрованог одговора. Шифрују се само информације за рутирање из заглавља пакета, док се шифровање корисног дела пакета не врши. Приликом прослеђивања саобраћаја користе се искључиво симетрични криптографски алгоритми и не чува се стање на чворовима. Асиметрични криптографски алгоритми примењују се на изворишту и чвору упаривања у току формирања путање пакета. Одговарајућим руковањем сваког појединачног типа пакета, *Dovetail* пружа гаранције наведене у наставку. Аутономни систем на путањи пакета не може сазнати идентитет аутономних система који му претходе са изузетком непосредног претходника. Информације о сваком појединачном аутономном систему у оквиру заглавља пакета заштићене су помоћу криптографског кључа познатог само датом аутономном систему. Шифровани текстови (енгл. *ciphertexts*) у оквиру пакета различитих сесија нису исти чак и када се крећу идентичном путањом. Коначни шифровани текст за информације сваког појединачног аутономног система зависи од целокупне путање пакета.

Како спречавање временских напада заснованих на анализи мрежних токова није једноставан задатак, *Dovetail* слично као и *LAP* ни не покушава да пружи заштиту од њих. Исто као и *LAP* систем, *Dovetail* је подложен тополошким нападима и постоји цурење информација о путањи пакета. Заштита анонимности ограничена је само на нападаче

стационаране на једном месту односно једном аутономном систему. У ову групу нападача спадају *ISP* изворишта, *ISP* одредишта, било који аутономни систем на путањи пакета, појединачни чворови као и само одредиште. Ради пружања адекватне заштите неопходно је постићи довољно велики анонимни скуп што захтева да извршен број аутономних система донекле олабави добро утврђену праксу екстерног рутирања [60].

Евалуација *Dovetail* система није обављена са стварним мрежним уређајима у реалним условима мрежног саобраћаја. Извршена је детаљна анализа само у симулираном окружењу. Конкретни подаци о перформансама *Dovetail* система нису познати из управо наведених разлога.

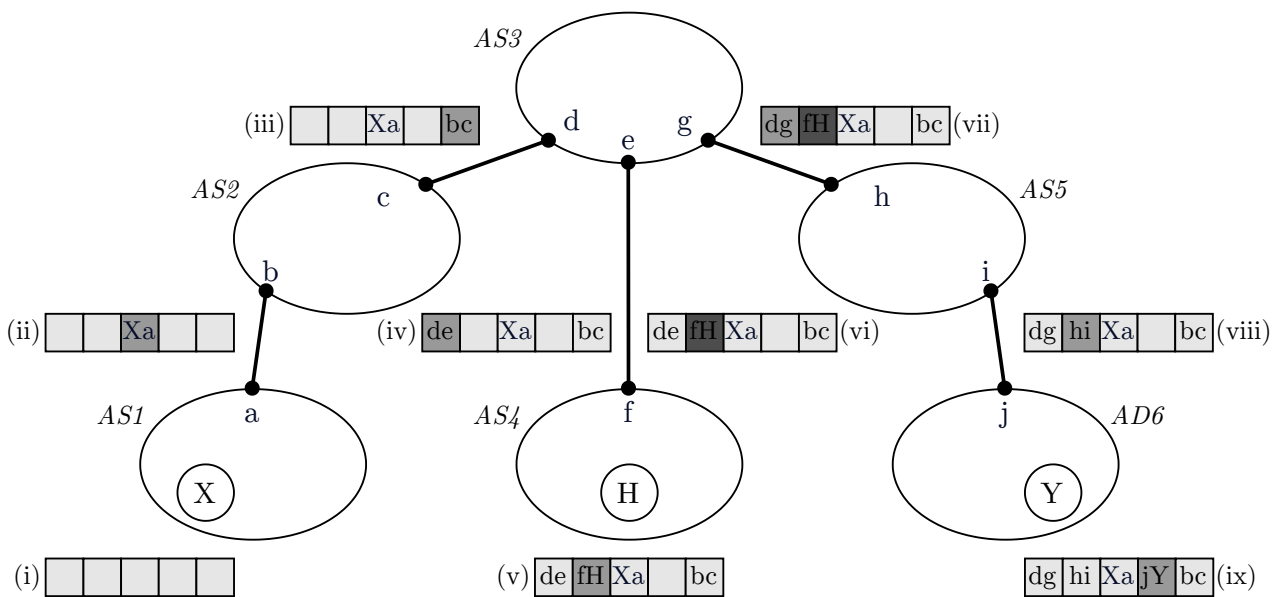
2.3.5 *PHI*

Протокол мрежног слоја за заштиту анонимности скривањем путање (енгл. *Path Hidden Lightweight Anonymity Protocol at Network Layer, PHI*) [61] унапређује гаранције анонимности у поређењу са *LAP* и *Dovetail* системима, задржавајући притом перформансе на истом нивоу. *PHI* представља решење за напад преотимања сесија (енгл. *session hijacking attack*) зарад деанонимизације одредишта. Изменом формата заглавља пакета, *PHI* спречава и цурење информација о путањи пакета, што представља један од механизма за превенцију тополошких напада. Ипак, чак и уз наведена унапређења, *PHI* спада у системе са релаксираним гаранцијама приватности. Слично као и већина других система са релаксираним гаранцијама приватности на мрежном слоју, не чува никакво стање сесије на појединачним чворовима, тајни кључеви су локални за сваки чвор и приликом прослеђивања пакета користе се искључиво симетрични криптографски алгоритми.

LAP и *Dovetail* не врше шифровање корисног дела пакета, иако то може помоћи у спречавању деанонимизације крајњих тачака анализом садржаја пакета са подацима. Зато се уз ова два система често користи и Дифи-Хелман (енгл. *Diffie-Hellman, DH*) алгоритам за размену кључева како би се успоставио дељени тајни кључ. У оквиру *DH* алгоритма оба краја комуникације требало би да међусобно верификују јавне кључеве да би се спречио напад типа човек у средини (енгл. *man in the middle, MITM*). Међутим, како *LAP* и *Dovetail* немају уграђену подршку за размену јавних кључева, једино извориште може верификовати одредиште, али не и обрнуто без угрожавања анонимности изворишта. Због тога злонамерни чвор на путањи може заменити јавни кључ изворишта сопственим и преотети сесију. Пошто одредиште не може да провери порекло кључа, нападач може нарушити анонимност идентитета одредишта. *PHI* онемогућава напад преотимања сесије јер приликом формирања путање пакета пружа подршку и за размену симетричних криптографских кључева уз проверу идентитета. Јавни кључ *DH* алгоритма је део пакета захтева за формирање путање. Хеш вредност *DH* јавног кључа је такође део пакета захтева и назива се идентификатор сесије (енгл. *session identifier, SID*).

Како би се избегло чување стања сесије на чворовима, сегменти са стањем прослеђивања свих чворова на путањи пакета налазе се у заглављу пакета слично као код *LAP* и *Dovetail* система, али уз битну разлику наведену у наставку. Ради спречавања цурења детаља путање пакета сегменти се код *PHI* система не надовезују један за другим, већ сваки чвор смешта свој сегмент у један од слотова у заглављу пакета бирајући притом позицију на псеудослучајан начин што је илустровано на слици 2.3. Извориште унапред дефинише за колико највише сегмената ће бити места и алоцира толики број слотова у заглављу пакета. Избор слота врши се помоћу псеудослучајне функције чије су улазне вредности *SID* и локални тајни кључ датог чвора намењен за избор позиције сегмента. Пре убацивања у слот сваки чвор шифрује сегмент својим тајним кључем намењеним за шифровање сегмента. Извориште иницијално попуњава алоцирани простор у заглављу пакета низом бита псеудослучајних вредности да нападач не би могао јасно да разликује празне и

попуњене слотове. Захваљујући псеудослучајном избору позиције сегмента унутар заглавља пакета, спречава се цурење информација о топологији односно позицији чворова. Ипак, пошто сваки чвор бира место за свој сегмент на псеудослучајан начин и независно од других чворова, може доћи до колизије односно посматрани чвор смештањем свог сегмента може прегазити постојећи сегмент. Због тога сегмент, поред одлуке прослеђивања донете од стране чвора и позиције сегмента претходног чвора, садржи и код за аутентификацију поруке формиран користећи тајни кључ чвора намењен управо за формирање кода за аутентификацију. Код за аутентификацију поруке формира се над шифрованим текстом стања прослеђивања и кодом за аутентификацију поруке претходног чвора. Провером кода за аутентификацију поруке, сваки чвор може да установи да ли је његов сегмент исправан односно да ли је дошло до колизије. Према томе, успех успостављања сесије код *RHI* система директно зависи од вероватноће настанка колизије. Извориште зато шаље већи број пакета захтева за формирање путање, уместо само једног, како би вероватноћа настанка колизије била минимална.



Слика 2.3: Формирање путање пакета код *RHI* система

У склопу успостављања сесије, формира се путања пакета на сличан начин као код *Dovetail* система уз извесне разлике. Извориште не формира директно путању пакета скроз до одредишта, већ се користи индирекција како чворови у непосредној близини изворишта не би могли да утврде идентитет одредишта. Поступак индирекције заснива се на помоћном чвору (енгл. *helper node*) и средишњем чвору (енгл. *midway node*), који у великој мери имају сличне улоге као *Dovetail* чвор упаривања и чвор укрштања, респективно. Извориште бира помоћни чвор на произвољан начин, шифрује идентитет одредишта јавним кључем помоћног чвора и иницира формирање путање до датог помоћног чвора. Сваки чвор самостално доноси одлуку о прослеђивању пакета захтева за формирање путање до помоћног чвора, на основу његове јавно доступне адресе унутар пакета, захваљујући чему извориште не мора унапред да дефинише тачну путању пакета. Донету одлуку о прослеђивању, сачињену од улазног и излазног интерфејса, сваки чвор шифрује својим локалним тајним кључем и смешта у заглавље пакета на претходно описан псеудослучајан начин. Након пристизања пакета захтева, помоћни чвор дешифрује идентитет одредишта и започиње формирање другог дела путање пакета до одредишта. Помоћни чвор би потенцијално могао да надовеже други део на већ формиран први део путање пакета, али се то не ради како путања пакета не би била неоптимална и да се помоћни чвор не би оптеретио прослеђивањем података. Уместо тога примењује се метод повлачења (енгл.

back-off method) којим помоћни чвор у име изворишта тражи средишњи чвор у оквиру првог дела путање. Помоћни чвор шаље пакет за проналажење средишњег чвора у обрнутом редоследу првог дела путање пакета, тако да сваки чвор на основу правила рутирања проверава да ли је он заправо средишњи чвор. Изабрани средишњи чвор шаље захтев за формирањем другог дела путање пакета кроз до одредишта. Формирање путање пакета на овакав начин обезбеђује компатибилност са актуелном архитектуром интернета. *PHI* свакако може радити и над архитектурама нове генерације интернета као што су *NIRA*, *SCION* и *Pathlet*, али не зависи искључиво од њих.

Експериментална евалуација извршена је помоћу софтверског рутера на рачунару са три мрежне картице. Установљено је да *PHI* има перформансе у рангу *LAP* и *Dovetail* система, иако пружа нешто виши ниво заштите анонимности у односу на њих. Формирање путање траје 800 пута краће, а додатно кашњење услед обраде пакета са подацима три пута мање у поређењу са *HORNET* системом.

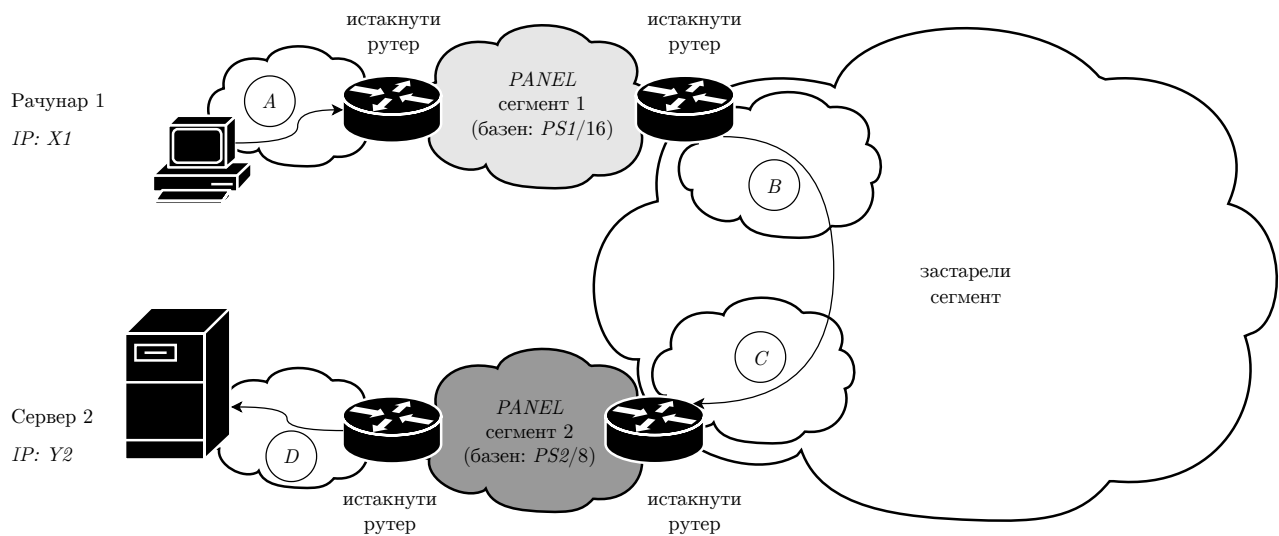
2.3.6 *PANEL*

Практични систем за релаксирану анонимност на мрежном слоју (енгл. *Practical Lightweight Anonymity at the Network Level, PANEL*) [62] пружа заштиту анонимности изворишта његовим скривањем унутар анонимног скупа, неповезивост већег броја сесија истог корисника изменом идентификатора сесије и заштиту детаља путање пакета увођењем једног или више примитивних мешајућих слојева. Стање сесија чува се на мрежним уређајима, уместо његовог слања унутар пакета, што је у супротности са већином система са релаксираним гаранцијама приватности на мрежном слоју. *PANEL* не захтева од крајњег корисника инсталацију наменских апликација, нити било какве друге измене, што га чини у потпуности транспарентним. Примена система је могућа одмах на постојећој архитектури интернета, јер се не ослања на технологије нове генерације интернета.

Посматрано из угла *PANEL* система, путања пакета између изворишта и одредишта логички је подељена на сегменте од којих се сваки састоји од једног или више аутономних система. Зависно од тога да ли активно пружају релаксирану заштиту анонимности применом *PANEL* механизма или по том питању нису уопште ангажовани, сегменти се деле на *PANEL* сегменте и на застареле (енгл. *legacy*) сегменте, респективно. *PANEL* сегменти и застарели сегменти могу коегзистирати на путањи пакета. Другим речима, не мора сваки сегмент на путањи пакета имати подршку за *PANEL* механизме да би *PANEL* систем могао бити примењен. Захваљујући овоме омогућена је локализована примена *PANEL* система на појединим сегментима путање пакета између изворишта и одредишта. Довољан је и само један *PANEL* сегмент на путањи пакета да би се у извесној мери обезбедила заштита анонимности изворишта, док сваки додатни *PANEL* сегмент повећава анонимни скуп и тиме побољшава заштиту приватности.

Истакнути (енгл. *landmark*) рутери, позиционирани на ивици *PANEL* сегмента, једини активно учествују у пружању релаксиране заштите анонимности применом *PANEL* механизма. Принцип рада и одлуке прослеђивања осталих мрежних уређаја унутар *PANEL* сегмента су независни односно нису под утицајем *PANEL* система. Истакнути рутери на ивици *PANEL* сегмента врше замагљивање (енгл. *obfuscation*) изворишне *IP* адресе, идентификатора сесије и других осетљивих поља заглавља пакета, што је илустровано на слици 2.4. Стога, *PANEL* систем имплицитно захтева поверење у први *PANEL* сегмент на путањи пакета, јер први *PANEL* сегмент познаје идентитет изворишта уколико се не примени неки додатни механизам заштите приватности. Како би се приликом пристизања пакета у повратном смеру извршила рестаурација оригиналних вредности замагљених поља заглавља пакета, сви неопходни подаци за сваку појединачну сесију чувају се на истакнутом рутеру који је извршио њихово замагљивање. Чување стања сесије на мрежним

уређајима, уместо његовог преноса путем пакета, је пројектна одлука донета у циљу постизања компатибилности са постојећим мрежама и ради побољшања заштите приватности. Пренос стања прослеђивањем у оквиру пакета захтева измену изгледа *IPv4* и *IPv6* пакета, што нарушава компатибилност са постојећом архитектуром интернета. Пренос стања прослеђивањем у оквиру пакета може довести и до цурења информација о идентитету корисника. Чување стања на истакнутим рутерима намеће ограничење по питању повратне путање пакета, која мора обухватити исте истакнуте рутере само у обрнутом редоследу, да би било могуће рестаурирати оригиналне вредности замагљених поља заглавља пакета. Притом, код *PANEL* система извориште не дефинише путању пакета унапред, већ се користи традиционални модел рутирања где се одлуке о прослеђивању доносе независно од стране сваког истакнутог рутера. *PANEL* систем подржава асиметрично рутирање унутар сваког сегмента, као и између *PANEL* сегмената докле год је очуван редослед *PANEL* сегмената у обрнутом поретку.



	Src IP	Dst IP	TTL	Src Port	Dst Port	Seq Num		
A	X	1	Y	2	128	1025	443	145326
B	PS1	tag	Y	2	72	tag	443	374892
C	PS1	tag	Y	2	59	tag	443	374892
D	PS2	tag	Y	2	51	tag	443	374892

Слика 2.4: Модификација поља заглавља пакета применом *PANEL* система

Истакнути рутер *PANEL* сегмента врши сакривање информација о изворишту у неколико корака. Први корак јесте замагљивање мрежног дела изворишне *IP* адресе. Сваки истакнути аутономни систем издваја базен *IP* адреса за употребу приликом замагљивања изворишних *IP* адреса. Изворишна *IP* адреса пакета мења се, на сваком истакнутом рутеру на путу пакета од изворишта до одредишта, новом *IP* адресом изабраном на псеудослучајан начин из базена *IP* адреса истакнутог аутономног система.

Други корак јесте нормализација изворишних информација погодних за формирање отиска у које спадају *IP* идентификација и број *TCP* секвенце, јер њихове иницијалне

вредности могу у великој мери зависити од софтвера коришћеног за генерисање што директно омогућава прављење отиска оперативног система датог корисника. Нормализација ових изворишних информација врши се искључиво од стране првог истакнутог рутера на путањи пакета, што значи да сваки истакнути рутер мора утврдити да ли је управо он први на путањи пакета. Ово је могуће утврдити увидом у изворишну *IP* адресу и провером да ли припада истом *PANEL* сегменту као и посматрани истакнути рутер или потиче из базена *IP* адреса неког другог истакнутог аутономног система што сугерише да ју је негде раније на путањи пакета поставио неки претходни истакнути рутер. Приликом успостављања сесије, први истакнути рутер на путањи пакета генерише псеудослучајни померај који ће зависно од смера пакета додавати или одузимати од вредности *IP* идентификације и броја *TCP* секвенце.

Трећи корак јесте скривање информација о путањи пакета постављањем псеудослучајно изабране вредности за *TTL* поље. Нова вредност *TTL* поља бира се на основу емпиријски одређене расподеле вероватноћа за адекватну вредност *TTL* поља. Постављањем псеудослучајно изабране вредности онемогућава се откривање удаљености изворишта пакета од посматране тачке на основу тренутне вредности *TTL* поља.

Четврти корак јесте обезбеђивање неповезивости већег броја сесија истог корисника. У том циљу врши се замагљивање идентификатора сесије сачињеног од хост дела изворишне *IP* адресе и изворишног *TCP* порта. Уместо идентификатора сесије поставља се таг (енгл. *tag*), добијен помоћу генератора псеудослучајних бројева, дељењем на два дела и смештањем у хост део изворишне *IP* адресе и изворишни *TCP* порт. Приликом пристизања пакета проверава се да ли је за њега раније већ генерисан таг, и уколико јесте дати таг се користи, док се у супротном нови таг генерише. Према томе, нови таг се генерише за сваку нову сесију приликом наиласка првог пакета из дате сесије. Измена изворишне *IP* адресе у општем случају представља изазов за прослеђивање пакета у повратном смеру, тако да сваки истакнути рутер мора чувати стање сесије да би касније био у могућности да изврши реверзну операцију.

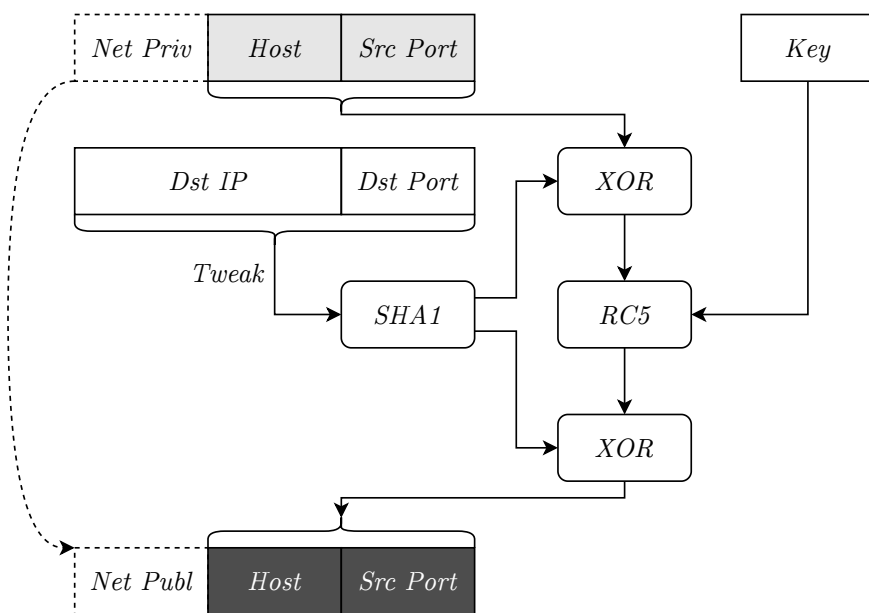
PANEL је имплементиран за *Barefoot Tofino* мрежни уређај са чак 3,3 Tb/s укупног пропусног опсега. На основу резултата експерименталне евалуације, *PANEL* остварује 96 % пропусног опсега на нивоу брзине везе, док додатно кашњење услед обраде пакета износи 3 %. Максимални капацитет доступне меморије за чување стања сесија, достиже се већ при дужини интерног идентификатора сесије од 21 бита, што јасно сугерише да је број подржаних сесија ограничен.

2.3.7 *AHP*

Протокол за скривање адреса (енгл. *Address Hiding Protocol, AHP*) [63] пребацује на *ISP* обавезу заштите приватности свих корисника из његовог домена. *AHP* скрива идентитет крајева комуникације на мрежном слоју скривајући њихове *IP* адресе од посматрача позиционираног изван *ISP* домена. Корелација два мрежна тока између истих крајева комуникације, на основу информација из мрежног или транспортног слоја, такође није могућа. Пошто се *AHP* не заснива ни на једној од технологија нове генерације интернета, могућа је његова примена на постојећој архитектури интернета.

AHP систем захтева да *ISP* сваком интерфејсу у оквиру свог домена додели приватну адресу из опсега 10.128.0.0/16. Мрежни уређај на ивици *ISP* домена шифрује хост део приватне изворишне *IP* адресе и изворишни порт сваког излазног пакета. Процес шифровања поред тајног кључа укључује и идентитет одредишта односно његову *IP* адресу и порт, како два мрежна тока иницирана од стране истог изворишта не би могла бити доведена у корелацију и генерално ради побољшања приватности. *AHP* користи нестандартну блоковску шифру (енгл. *block cipher*) са *параметром модификације* (енгл. *tweak*), засновану на

RC5 блоковској шифри и *SHA1* хеш функцији, чији су детаљи илустровани на слици 2.5. Изабрани параметри *RC5* шифре су 32-битни блок и 20 рунди. Улаз описане блоковске шифре односно отворени текст (енгл. *plaintext*) формира се од 16-битног хост дела приватне изворишне *IP* адресе и изворишног порта. Надовезивањем читаве одредишне *IP* адресе и одредишног порта добија се 48-битни параметар модификације описане блоковске шифре. Применом описане блоковске шифре на претходно дефинисан 32-битни отворени текст, добија се 32-битни шифровани текст који замењује хост део приватне изворишне *IP* адресе и изворишни порт. Последњи корак јесте замена мрежног дела приватне изворишне *IP* адресе са јавним *IP* префиксом.

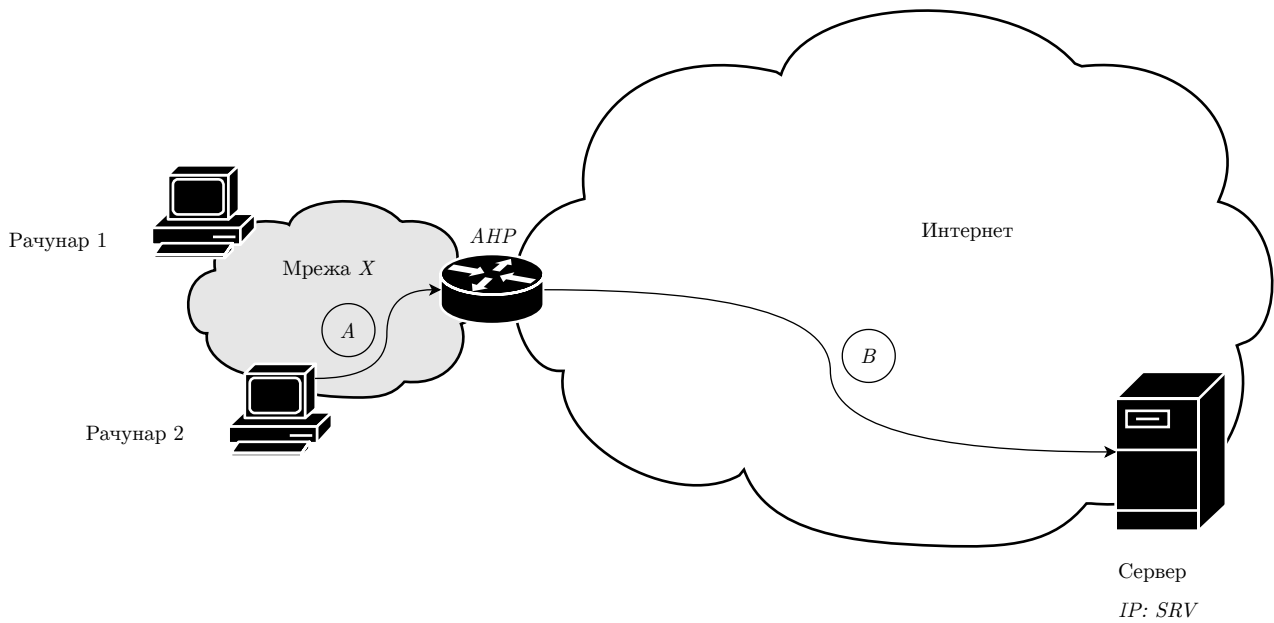


Слика 2.5: Скривање адресе код *АНП* система

Коришћени тајни кључ се мора редовно мењати током времена да би се спречили напади на бази парадокса рођендана. *АНП* не убацује ниједан додатни бит у заглавље ни у садржај пакета на основу којег би се могло закључити када је дати пакет настао и самим тим који тачно тајни кључ је коришћен приликом скривања адресе. Због тога може доћи до колизије формираних *IP* адреса и портова услед коришћења различитих кључева код мрежних токова дугог животног века уколико се тајни кључ промени усред животног века посматраног мрежног тока. Управо услед тога је неопходно одржавати скупове асоцијација пакета и коришћених кључева што имплицира чување незанемарљивог стања на мрежним елементима.

Заштита приватности идентитета је остварива скривањем у маси других корисника односно скривањем унутар анонимног скупа. Уколико нападач не може јасно разликовати чланове анонимног скупа, приватност идентитета сваког члана скупа је сачувана. Притом, што је анонимни скуп већи, то је виши и ниво заштите приватности. Пермутацијом свих бита хост дела *IP* адресе, свака мрежна маска дефинише анонимни скуп у оквиру којег се може сакрити идентитет. Задатак *ISP* јесте да измени хост део *IP* адреса на такав начин да изван *ISP* домена није могуће закључити чији се идентитет крије иза датог хост дела, што је илустровано на слици 2.6. Агрегацијом мањих мрежа из *ISP* домена добијају се довољно велике мрежне маске за успешно чување приватности идентитета. *ISP* може несметано поставити *АНП* без да то корисници уопште и знају, а да одмах од тога имају бенефит у виду повећања нивоа заштите приватности.

Резултат скривања адреса јесте да сваки мрежни ток инициран од стране једног истог изворишта, посматрано са било које тачке изван *ISP* домена, наизглед потиче од различите



	<i>Src IP</i>		<i>Dst IP</i>	<i>Src Port</i>	<i>Dst Port</i>
A	10.128	2	SRV	1025	443
	<i>Src IP</i>	<i>Dst IP</i>	<i>Src Port</i>	<i>Dst Port</i>	
B	PubX	EncH	SRV	EncSrcP	443

Слика 2.6: Модификација поља заглавља пакета применом ANP система

псеудослучајне IP адресе из опсега адреса датог ISP. Стога, резултат скривања адреса у великој мери подсећа на превођење мрежних адреса (енгл. *Network Address Translation, NAT*), у смислу да су IP адресе анонимизоване за посматрача изван ISP домена. Битна разлика јесте то што је скривање адреса од стране ANP система реализовано као бијективно пресликавање захваљујући примени криптографског алгорита и самим тим не захтева одржавање великих табела свих успостављених мапирања као у случају NAT. Згодна особина јесте и то да у случају пристизања легитимног захтева од стране законских органа, ISP је у могућности да једнозначно доведе у везу било који мрежни ток са извориштем које га је заиста иницирало само на основу познавања криптографског материјала коришћеног у датом временском тренутку.

ANP не штити корисника од напада анализом корисног дела пакета, од временских напада нити од напада споредним каналима. Не пружа ни приватност нити проверу аутентичности података. У складу са пројектним одлукама, ANP спада у системе са релаксираним гаранцијама приватности и самим тим не покушава да се надмеће са другим системима са јаким гаранцијама приватности. Главни циљ јесте што распрострањенија примена зарад подизања тренутног нивоа заштите приватности на мрежном слоју свих корисника појединачних ISP домена. Већи број улаза и излаза из мреже је подржан тако што сви ивични ISP мрежни уређаји примењују идентичну претходно описану операцију шифровања користећи исти криптографски материјал. Потпуно поверење у ISP се очигледно имплицитно захтева, јер он врши скривање адреса и гарантује довољно велики анонимни скуп, али имплементација ANP система не спречава додатну примену других механизма за заштиту приватности виших слојева као што је Tor. Посебна погодност ANP система јесте његова транспарентност за крајеве комуникације односно чињеница да не захтева никакве измене апликација на крајевима комуникације.

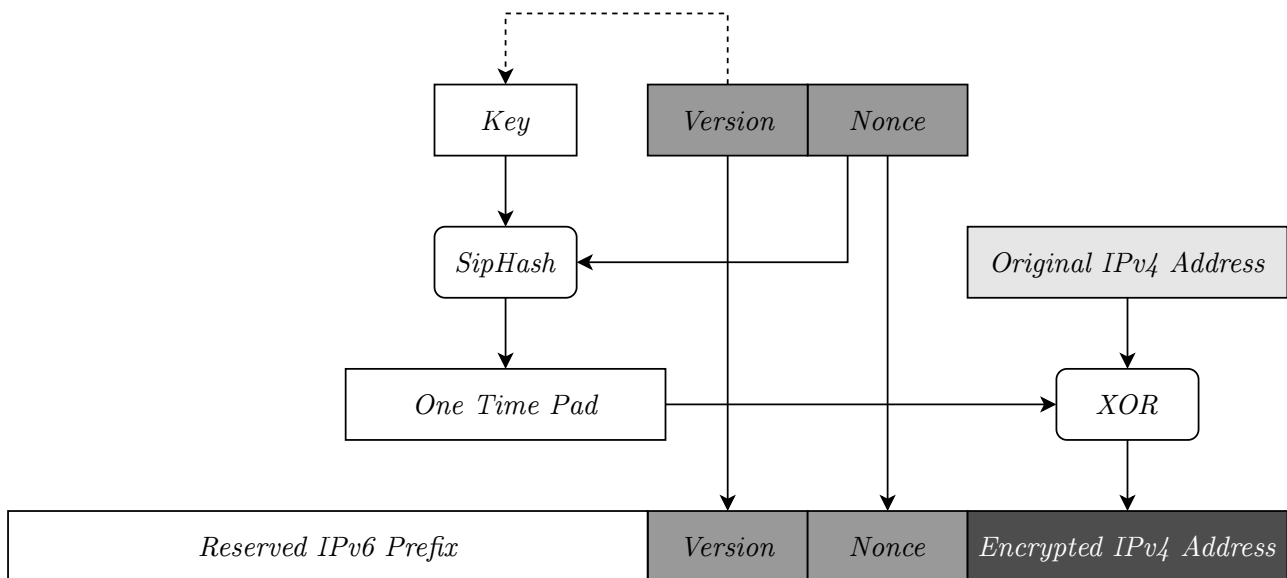
2.3.8 SPINE

Систем за заштиту од надзора помоћу мрежних елемената (енгл. *Surveillance Protection in the Network Elements, SPINE*) [64] омогућава крајевима комуникације, из два различита аутономна система у које постоји поверење, да комуницирају без откривања изворишне и одредишне *IP* адресе. *SPINE* спада у системе са релаксираним гаранцијама приватности, а пружа заштиту од посредних аутономних система односно чворова који покушавају да установе праву изворишну и одредишну *IP* адресу. Без икакве интервенције крајева комуникације, *SPINE* сакрива стварне *IP* адресе и битна поља *TCP* заглавља од посредних чворова, захтевајући сарадњу само између два чвора којима крајеви комуникације припадају. Никаква сарадња посредних чворова на путањи пакета није потребна чиме се смањује скуп чворова који морају модификовати своје понашање. *SPINE* се не ослања на решења нове генерације интернета, већ се може применити на постојећој архитектури интернета и то на једноставним *SmartNIC* уређајима уз пропусни опсег на нивоу брзине везе.

Могућност опажања односно праћења *IP* адреса крајева комуникације представља озбиљну претњу по приватност и анонимност корисника, јер *IP* адресе откривају информације о крајевима комуникације чак и када је саобраћај између њих шифрован [65]. Праћење *IP* адреса је најчешће могуће зато што су *IP* адресе крајева комуникације уобичајено видљиве посредним чворовима јер представљају идентификаторе крајева комуникације и као такве се користе у процесу прослеђивања саобраћаја. *SPINE* ово онемогућава тиме што у излазном смеру шифрује *IP* адресе пре него што пакети стигну до посредних чворова, док у долазном смеру обавља инверзну операцију односно дешифрује *IP* адресе из пакета пристиглих од посредних чворова. Захваљујући овоме, посредни чворови виде само шифроване *IP* адресе уместо правих *IP* адреса крајева комуникације. *SPINE* шифрује и бројеве *TCP* секвенце и потврде у циљу спречавања успостављања релације између пакета и припадајућег мрежног тока.

SPINE шифрује *IP* адресе помоћу једнократне заштите (енгл. *one-time pad*) реализовано применом логичке операције ексклузивно ИЛИ на *IP* адресу и излаз хеш функције са тајним кључем над псеудослучајном једнократном вредношћу као што је илустровано на слици 2.7. Тајни кључ за хеш функцију је познат свим ивичним мрежним уређајима у оквиру чворова којима припадају крајеви комуникације, тако да пакети могу излазити и улазити у дате чворове са било ког ивичног мрежног уређаја. Псеудослучајна једнократна вредност је јавна и шаље се заједно са шифрованом *IP* адресом у оквиру заглавља пакета. Због тога што нападач може доћи у посед парова отвореног и шифрованог текста, тако што сам пошаље пакете са задатим *IP* адресама и посматра дате пакете након напуштања *SPINE* мрежних уређаја, за хеш функцију је одабран *SipHash* као довољно криптографски јака псеудослучајна функција. *SipHash* алгоритам користи 128-битни тајни кључ и производи 64-битни излаз. Додатна погодност *SipHash* алгоритма јесте његова оптимизованост за кратке поруке и то што се базира искључиво на операцији сабирања и ексклузивно ИЛИ чинећи га погодним за имплементацију на *SmartNIC* уређајима. Бројеви *TCP* секвенце и потврде се шифрују на идентичан начин, користећи исту псеудослучајну једнократну вредност али различити тајни кључ. Тајни кључ за шифровање *IP* адреса и тајни кључ за шифровање бројева *TCP* секвенце и потврде заједно чине један скуп кључева.

SPINE користи описану једнократну заштиту уместо неког од стандардних криптографских алгоритама како би постигао што боље перформансе. Једнократна заштита сматра се безбедном све док се вредност за шифровање добија на заиста случајан начин и док нема њеног понављања односно вишеструке употребе. Ради тога *SPINE* генерише нову псеудослучајну једнократну вредност за сваки пакет, користи криптографски јаку хеш функцију и редовно мења кључ. Аутори система наводе да дистрибуцију кључева до сваког *SPINE* мрежног уређаја врши централни координатор, при чему не објашњавају како

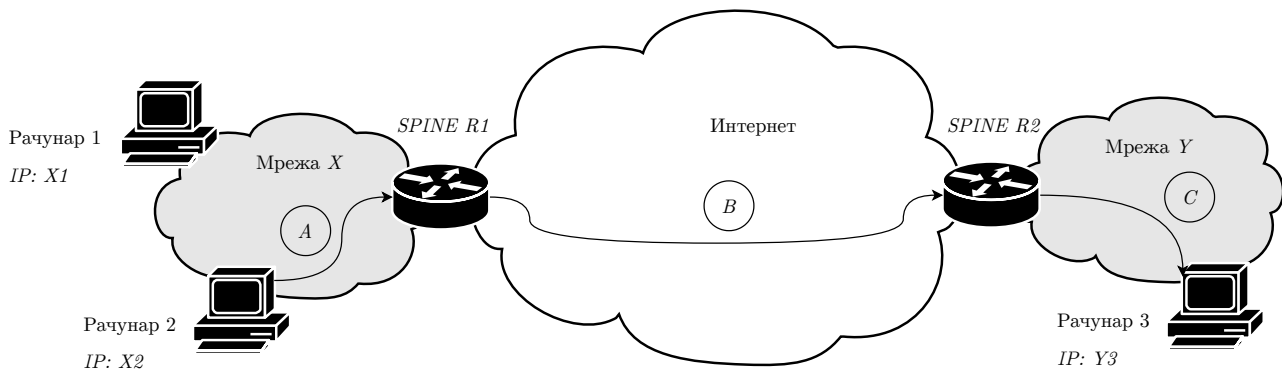


Слика 2.7: Шифровање адреса код *SPINE* система

дати централни координатор функционише. Ипак честа замена кључева може довести до проблема јер се може десити да се користе различити кључеви за шифровање и дешифровање уколико до замене кључева дође након што пакет напусти чвор изворишта а пре него што се одговор по датом пакету врати назад. *SPINE* систем зато у сваком тренутку чува три скупа кључева: претходни, тренутни и наредни. Извориште у оквиру пакета шаље управо и верзију скупа кључева коришћеног за шифровање, тако да одредиште на основе послате верзије одабира исправан скуп кључева за дешифровање. Претходно поменути централни координатор је задужен и за ажурирање верзија кључева на свим *SPINE* мрежним уређајима.

Смештање једнократне псеудослучајне вредности, верзије скупа кључева и шифрованих *IP* адреса у заглавље пакета јесте главни изазов са којим се *SPINE* суочава. Решење за овај проблем јесте трансформација *IPv4* заглавља пакета у *IPv6* заглавље приликом напуштања чвора изворишта и затим инверзна трансформација *IPv6* заглавља пакета у *IPv4* заглавље приликом пристизања пакета назад до чвора одредишта. *SPINE* користи *IPv6* заглавље пакета јер је довољно велико да се у њега складиште све потребне претходно наведене информације. Највиших 64 бита *IPv6* адресе представљају валидни и унапред резервисани *IPv6* префикс који припада чвору изворишта односно одредишта. Наредна 32 бита *IPv6* адресе садрже верзију скупа кључева и једнократну псеудослучајну вредност, док су последња 32 бита *IPv6* адресе предвиђена за шифровану *IPv4* адресу што је приказано на слици 2.8. Бројеви *TCP* секвенце и потврде просто се замењују својим шифрованим вредностима. Унапред резервисани *IPv6* префикс притом омогућава и једноставно препознавање пакета чија поља заглавља треба дешифровати приликом уласка пакета у чвор, пошто се дати префикс не користи ни за једну другу сврху поред *SPINE* система. Такође, прослеђивање пакета функционише на регуларан начин пошто је реч о валидном *IPv6* префиксу. Негативна страна постојања унапред резервисаног *IPv6* префикса свакако јесте откривање информација о изворишту и одредишту пакета до извесне мере, али ово није могуће избећи с обзиром на то да посредни чворови морају знати на који начин да прослеђују пакете.

Како је применом *TLS* протокола садржај пакета углавном већ шифрован, *SPINE* не врши шифровање корисног дела пакета већ се фокусира на замагљивање *IP* адреса и поља *TCP* заглавља. Да би то постигао *SPINE* се ослања на *IPv6* протокол, прецизније на чињеницу да ширина *IPv6* адресе износи 128 бита што омогућава складиштење шифрованих



	<i>Src IPv4</i>	<i>Dst IPv4</i>	<i>Src Port</i>	<i>Dst Port</i>								
A	X	2	Y	3	1025	443						
	<i>Src IPv6</i>			<i>Dst IPv6</i>			<i>Src Port</i>	<i>Dst Port</i>				
B	IPv6 Prefix			V	N	EncIPv4	IPv6 Prefix	V	N	EncIPv4	EncSrcP	EncDstP
	<i>Src IPv4</i>	<i>Dst IPv4</i>	<i>Src Port</i>	<i>Dst Port</i>								
C	X	2	Y	3	1025	443						

Слика 2.8: Модификација поља заглавља пакета применом *SPINE* система

IPv4 адреса и пратећих података. Због тога се *SPINE* користи за заштиту приватности искључиво *IPv4* саобраћаја, који се енкапулира у *IPv6* пакете, што за директну последицу има да заштита приватности у случају *IPv6* саобраћаја није изводљива. *SPINE* је сличан *AHP* по томе што укључује аутономне системе у процес заштите приватности. *AHP* је најјефикаснији код великих аутономних система са кратким префиксима јер је у том случају већи домен могућих *IP* адреса, док је *SPINE* подједнако ефикасан за све величине аутономних система захваљујући употреби стандардних *IPv6* префикса. *SPINE* је успешно имплементиран за програмабилне *SmartNIC* уређаје са подршком за *PISA* архитектуру, али је тестиран само у емулираном окружењу.

2.3.9 *PINOT*

Систем за замагљивање детаља мрежног саобраћаја на програмабилним мрежним уређајима (енгл. *Programmable In-Network Obfuscation of Traffic, PINOT*) [66] шифрује *IPv4* адресу изворишта како би сакрио адресу корисника од непосредних аутономних система на путањи пакета и од самог одредишта мрежног тока. Никакве измене нити инсталација додатних програма на страни изворишта нису потребне, нити је неопходна сарадња између већег броја аутономних система. *PINOT* имплицитно захтева поверење у онај мрежни домен којем извориште припада, што имплицира релаксиране гаранције приватности.

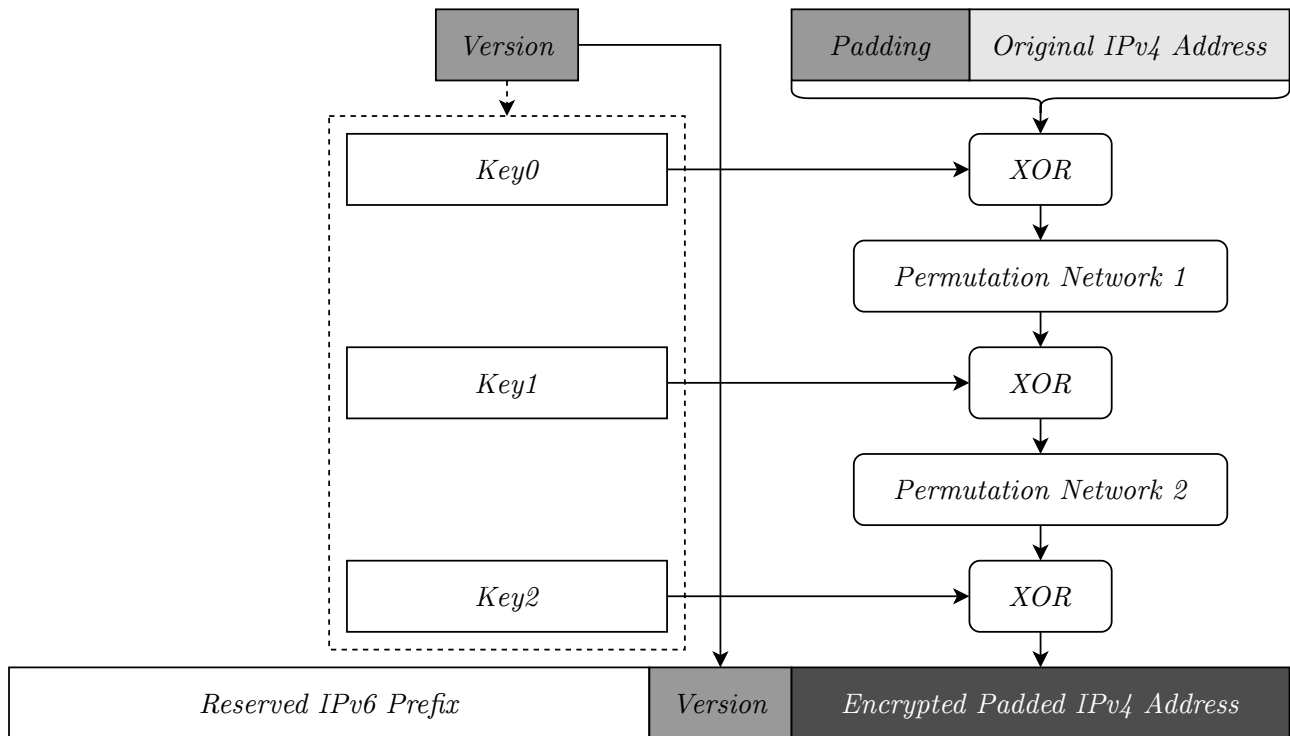
Постојећи протоколи шифровања, као што је *TLS*, гарантују приватност података односно корисног дела пакета мрежног тока, али не скривају податке из заглавља пакета, као што су *IP* адресе односно идентитети крајева комуникације. Директна импликација јесте да су *IP* адресе изворишта видљиве и поред примене *TLS* протокола, што се може искористити за идентификовање изворишта мрежног саобраћаја и тиме угрожавање приватности. *PINOT* пружа релаксирану заштиту анонимности корисника, што представља искорак ка свеобухватнијој заштити приватности корисника на мрежном слоју. Релаксирана заштита анонимности корисника подразумева да аутономни систем, унутар којег је *PINOT* примењен, зна оба краја комуникације, али посредни аутономни системи на путањи пакета

не знају ко је извориште мрежног саобраћаја. Притом, *PINOT* се фокусира на заштиту *IP* адреса корисника приликом приступа често коришћеним јавним сервисима као што су *DNS*, *VPN* и протокол мрежног времена (енгл. *network time protocol*, *NTP*). Како су *DNS* упити и одговори обично у отвореном тексту, *DNS* разрешивачи или било који прислушкивач на путањи пакета могу да виде *IP* адресу корисника као и домен за разрешавање што директно може да се користи за профилисање корисника и њихових активности на интернету. Пружаоци *VPN* услуга такође могу угрозити приватност корисника бележећи *IP* адресе корисника и продајући их зарад персонализованог оглашавања или ненамерним објављивањем односно цурењем датих *IP* адреса. Нападач може потенцијално да искористи *NTP*, увођењем властитих јавно доступних *NTP* сервера, како би открио тренутно активне *IPv4* и *IPv6* кориснике и тиме сузио скуп уређаја за скенирање на познате слабости.

PINOT се извршава на ивици мрежног домена у који постоји поверење и ту шифрује *IPv4* адресе изворишта мрежног тока пре него што пакети напусте дати мрежни домен. Слично као и *SPINE*, *PINOT* конвертује *IPv4* заглавље пакета у *IPv6* како би се у *IPv6* адресу сместила шифрована *IPv4* адреса и додатни подаци потребни за процес дешифровања. Управо захваљујући *IPv6* адресама, *PINOT* не захтева чување стања на мрежним уређајима за појединачне мрежне токове. Коришћењем *IPv6* адреса гарантује се и исправно прослеђивање пакета у оквиру постојеће архитектуре интернета. Нападач не може никако искористити шифровану изворишну *IPv4* адресу из пакета, док на основу новоформиране изворишне *IPv6* адресе може да сазна једино мрежни домен односно најчешће аутономни систем из кога је пакет потекао, али не може да сазна идентитет датог корисника нити да повеже различите пакете потекле од истог изворишта. *PINOT* користи ефикасну шифру засновану на Ивен-Мансур (енгл. *Even-Mansour*, *EM*) шеми јер је *EM* шему могуће имплементирати на *SmartNIC* уређајима тако да се све операције изврше у једном пролазу пакета кроз ток обраде (енгл. *processing pipeline*) ради постизања већег протока пакета. Поред програма за шифровање и дешифровање *IP* адреса коришћењем *EM* шеме извршаване на равни података мрежних уређаја смештених између крајњих корисника и ивице заштићене мреже, *PINOT* обухвата и софтверски контролер за управљање криптографским кључевима. Захваљујући софтверском контролеру и размени кључева, *PINOT* може да се примени на више ивичних мрежних уређаја унутар једног аутономног система уз директну подршку за асиметрично прослеђивање саобраћаја.

PINOT смешта читав криптографски алгоритам у програм за раван података програмабилних мрежних уређаја у циљу постизања протока на нивоу брзине везе. Ово није једноставно постићи за криптографске алгоритме као што је *AES* и управо зато се прибегава нестандартном решењу базираном на *EM* шеми шифровања са две рунде. *EM* шему шифровања је могуће имплементирати коришћењем табела за претраживање и логичке операције ексклузивно ИЛИ, без потребе за сложеним криптографским операцијама у равни података. Коришћена шема шифровања прво примењује логичку операцију ексклузивно ИЛИ на улазну вредност и засебан тајни кључ као што је илустровано на слици 2.9. Тако добијена вредност се пропушта кроз две рунде где се у свакој од њих вредност прво пермутује уз помоћ нестандартне супституционо-пермутационе мреже (енгл. *substitution-permutation network*), након чега се примењује логичка операција ексклузивно ИЛИ на тако добијену вредност и тајни кључ. Свака рунда користи засебан кључ и користи засебну супституционо-пермутациону мрежу. За супституцију се користи већи број осмобитних супституционих блокова идентичних онима код *AES* алгоритма, док се за пермутацију користи произвољно дефинисано мешање бита. Улазна вредност шеме за шифровање се формира од оригиналне *IPv4* адресе и псеудослучајног проширења (енгл. *padding*) захваљујући којем се постиже недетерминистички излаз при сваком новом шифровању једне исте *IPv4* адресе. Наведени недетерминизам помаже у спречавању корелације између пакета истог мрежног тока, јер би у супротном нападач могао лако да закључи да су пакети део

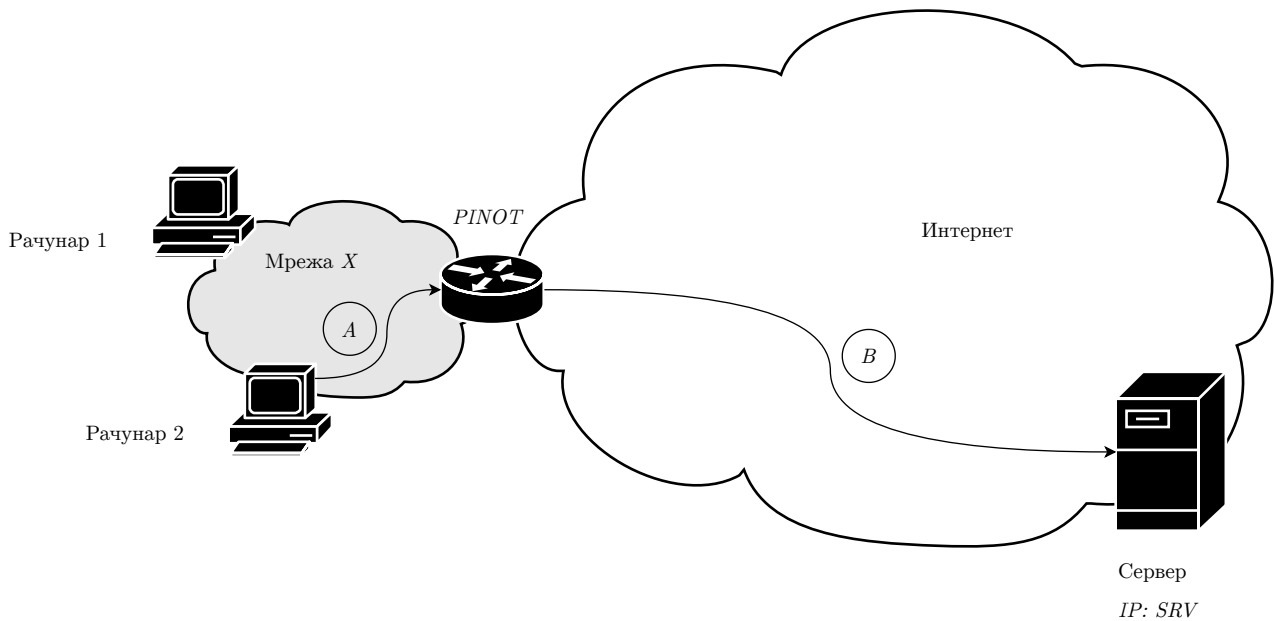
истог мрежног тока на основу идентичног шифрованог текста $IPv4$ адресе.



Слика 2.9: Шифровање адреса код *PINOT* система

PINOT на сваких t секунди ротира скуп тајних кључева сачињен од три претходно описана криптографска кључа. Ротирањем скупа кључева ограничава се број парова отвореног и шифрованог текста, које нападач може да прибави за дати скуп кључева, чиме се смањује успешност напада. Захваљујући учесталом ротирању скупа кључева на сваких t секунди, минимизира се и начињена штета у случају потенцијалног компромитовања скупа кључева. Ипак, учестало ротирање скупа кључева може довести до проблема уколико се различити скупови кључева користе за шифровање приликом изласка пакета из заштићеног домена и дешифровање приликом повратка тог истог пакета у заштићени домен. Овај проблем *PINOT* решава по узору на *SPINE* тако што чува три скупа кључева, док се унутар сваког појединачног пакета налази идентификатор коришћеног скупа кључева.

Шифровањем проширене изворишне $IPv4$ адресе добија се 56-битни шифровани текст који не може да се користи као валидна $IPv4$ адреса. Са друге стране, читав шифровани текст мора негде бити смештен зарад каснијег дешифровања. Како би се избегло чување стања на мрежним уређајима и ради осигуравања исправног прослеђивања мрежног саобраћаја, $IPv4$ пакет се замењује $IPv6$ пакетом приликом напуштања заштићеног домена. $IPv6$ заглавље пакета складишти шифровану проширену изворишну $IPv4$ адресу у оквиру изворишне $IPv6$ адресе, као што је приказано на слици 2.10. Највиших 64 бита изворишне $IPv6$ адресе представљају валидни $IPv6$ префикс који припада управо заштићеном домену. Наредних 8 бита $IPv6$ адресе садрже верзију скупа кључева коришћених за шифровање, док су последња 56 бита $IPv6$ адресе предвиђена за шифровани текст проширене изворишне $IPv4$ адресе. Сва остала поља $IPv4$ заглавља пакета, која имају одговарајућа поља у $IPv6$ заглављу, се једноставно копирају из $IPv4$ у $IPv6$ заглавље пакета. Одредишна $IPv4$ адреса се замењује својим $IPv6$ панданом, одакле директно проистиче и ограничење *PINOT* система да између изворишта и одредишта мора постојати и $IPv4$ и $IPv6$ повезаност. Ради убрзавања замене $IPv4$ адресе са $IPv6$ адресом, *PINOT* унапред шаље *DNS* упите за одредишта од интереса како би ускладиштио парове $IPv4$ и $IPv6$ адреса у табели за претраживање која ће се накнадно користити.



	<i>Src IPv4</i>	<i>Dst IPv4</i>	<i>Src Port</i>	<i>Dst Port</i>	
A	X	2	SRV	1025	443

	<i>Src IPv6</i>		<i>Dst IPv6</i>	<i>Src Port</i>	<i>Dst Port</i>	
B	X IPv6 Prefix	V	EncPadIPv4	SRV IPv6	1025	443

Слика 2.10: Модификација поља заглавља пакета применом *PINOT* система

PINOT измешта поверење са сервиса треће стране на аутономни систем тако да нико изван заштићеног домена датог аутономног система не може да види стварне изворишне *IPv4* адресе корисника посматрајући садржај пакета. За разлику од *NAT*, који кориснику додељује једну *IP* адресу на одређени временски период, *PINOT* додељује псеудослучајну *IP* адресу сваком пакету што је делотворнија заштита приватности корисника од праћења образаца мрежног саобраћаја. Ово посебно долази до изражаја због тога што *NAT* испољава изванредан степен детерминизма приликом избора *IP* адресе, услед чега је могуће повезати више сесија једног истог корисника и тиме угрозити његову приватност [67], што није случај код *PINOT* система. Скенирање портова је такође отежано захваљујући примени *PINOT* система, јер се мапирање између *IP* адреса и стварних крајњих интерфејса често мења.

PINOT систем не чува стање што га чини скалабилним и такође омогућава асиметрично прослеђивање саобраћаја. Дистрибуирањем тајних кључева помоћу софтверског контролера, аутономни систем може да примени *PINOT* на већи број ивичних тачака заштићеног домена, тако да излазни саобраћај може слободно напустити заштићени домен преко било које излазне тачке, при чему се повратни саобраћај може вратити преко било које улазне тачке. *PINOT* се покреће само у склопу једног аутономног система и може да врши замагљивање *IPv4* адреса свих корисника из дате заштићене мреже, што значајно олакшава примену *PINOT* система у односу на друге системе где постоји потреба за сарадњом између више аутономних система.

Код програмабилних мрежних уређаја са *PISA* архитектуром, ток обраде пакета је подељен на неколико фаза. Свака фаза дозвољава ограничен број претраживања табела и логичко-математичких операција. Програм писан за раван података гарантује пропусни опсег на нивоу брзине везе само уколико може да се смести унутар датих фаза и самим тим

не захтева више од једног пролаза пакета кроз ток обраде пакета. Ово представља директан ограничавајући фактор због којег *PINOT* систем користи *EM* шему шифровања; смештање неког од стандардних криптографског алгоритама у просечан мрежни уређај уз само један пролаз пакета кроз ток обраде пакета је изузетно захтевно у случају програмирања равни података.

PINOT систем постиже анонимност изворишта тако што нападач не може да сазна праву *IP* адресу изворишта, уз изузетак аутономног система којем извориште припада. Обезбеђује и неповезивост пакета; за задати скуп пакета, нападач не може да закључи да ли пакети потичу од истог изворишта посматрајући *IP* адресе из датих пакета. *PINOT* је пројектован тако да су баријере за његову примену минималне, јер није потребна никаква модификација постојеће архитектуре интернета и протокола, нити је потребно користити посебне апликације на страни корисника. *PINOT* има пропусни опсег на нивоу брзине везе.

2.4 Анализа постојећих решења

Зарад брзог прослеђивања и скалабилности на мрежним уређајима са ограниченим процесорским ресурсима по пакету и меморијским простором по сесији, већина постојећих решења приликом прослеђивања пакета користи само симетричне криптографске алгоритме, а стање прослеђивања чува унутар самог пакета уместо складиштења на мрежном уређају. На почетку сваке сесије крајеви комуникације размењују пакете, како би том приликом сви аутономни системи на путањи пакета формирали свој сегмент стања прослеђивања и уградили исти у дате пакете, што има свој негативан утицај на перформансе. У неким случајевима, као што је *HORNET*, пакети се додатно шифрују од стране система. У другим случајевима налик *Dovetail* систему, не постоји додатно шифровање, већ се систем заснива на неком другом механизму заштите. Иако сви ови системи имају релативно велики пропусни опсег и мало кашњење, њихове сигурносне гаранције суштински нису веће од оних које *Tor* гарантује. Слично као *Tor*, сви ови системи су подложни временским нападима заснованим на довођењу образаца мрежног саобраћаја у корелацију, док су неки новији системи попут *HORNET* такође показали да имају и друге, раније непознате, рањивости [68]. Ово само потврђује постојање неизбежног компромиса између гаранције високог нивоа приватности са једне стране и одржавања великог пропусног опсега односно малог кашњења са друге стране [69].

За разлику од већине система са релаксираним гаранцијама заштите приватности, *PANEL* чува стање на мрежним уређајима. Систем не захтева од крајњег корисника инсталацију наменских апликација, нити било какве друге измене, што га чини у потпуности транспарентним. *PANEL* систем представља једну од потврда да је пружање релаксираних заштите анонимности изворишта могуће извести скривањем изворишта унутар анонимног скупа [52], као и да се неповезивост већег броја сесија истог корисника може постићи изменом идентификатора сесије. Систем притом не захтева промене у погледу формата пакета, а његова примена је могућа одмах на постојећој архитектури интернета, јер се не ослања на технологије нове генерације интернета. Експериментална евалуација, користећи имплементацију за *Barefoot Tofino* мрежни уређај, показала је да систем остварује 96 % пропусног опсега на нивоу брзине везе, док додатно кашњење услед обраде пакета износи 3 %.

Нешто старији *ANP* систем за скривање *IP* адреса, који такође чува стање током свог рада, врши шифровање хост дела класе *B IPv4* адресе и изворишног порта користећи шему засновану на *RC5*, са додатком случајног параметра модификације. *ANP* систем не уноси ниједан додатни бит својих метаподатака у заглавље или садржај пакета, тако да пакет није означен по питању тајног кључа коришћеног за операцију шифровања

односно скривања адресе. Систем захтева чување стања јер води евиденцију мапирања мрежних токова како би могао да изврши избор исправног криптографског кључа приликом операције дешифровања односно замене адреса на повратној путањи пакета. У том случају, шифровани пар (изворишна *IP* адреса, изворишни порт) користи се као кључ. Иако томе није посвећено пуно пажње у раду, систем пати од проблема парадокс рођендана и има вероватноћу од 0,5 за колизију при 64 000 истовремених мрежних токова.

Серија радова о заштити приватности на мрежном слоју истраживачке групе са Принстон универзитета примарно је инспирисала истраживање у склопу ове докторске дисертације. *SPINE* је систем за замагљивање *IP* адреса и бројева *TCP* секвенце и потврде. Овај систем шифрује изворишну *IP* адресу из заглавља оригиналног *IPv4* пакета и кодира шифровани текст у новокреирано *IPv6* заглавље пакета, притом одбацујући оригинално *IPv4* заглавље пакета. Да би се избегло шифровање једне *IP* адресе увек у исти шифровани текст током коришћења једног истог кључа, *SPINE* укључује псеудослучајну једнократну вредност у процес шифровања *IP* адресе. Једнократна вредност бира се за сваки пакет на псеудослучајан начин, па је очекивано да се разликује од пакета до пакета, услед чега се добијају и различити шифровани текстови за исту *IP* адресу из различитих пакета. Да би се омогућило реверзибилно дешифровање, *SPINE* кодира шифровану *IP* адресу и коришћену једнократну вредност у новокреирано *IPv6* заглавље пакета. *IPv6* адреса, која је дужа од *IPv4* адресе, може да складишти и шифровану *IPv4* адресу и једнократну вредност. Ради постизања бољих перформанси, *SPINE* користи једноставну нестандартну шифру засновану на логичкој операцији ексклузивно ИЛИ. *SPINE* систем је налик *VPN* у погледу тога да два аутономна система која сарађују представљају крајње тачке новокреираних *IPv6* тунела. Шифровање оригиналних *IPv4* адреса и њихово складиштење у новом *IPv6* заглављу пакета скрива детаље оригиналне комуникације од посредних аутономних система. *SPINE* систем не чува стање јер не мора да памти мапирање између оригиналних *IPv4* адреса и новокреираних *IPv6* адреса; дато мапирање имплицитно обезбеђује процес шифровања/дешифровања. Међутим, за рад овог система је неопходно мапирање између одредишне *IPv4* адресе и одговарајућег *IPv6* префикса крајње тачке, као и претходна размена криптографских кључева.

Wang је користећи *P4* језик развио *PINOT* систем за шифровање *IP* адреса према шеми у којој се изворишна *IP* адреса проширује случајним подацима до величине блока криптографског алгоритма, а затим шифрује тако проширена. Шема шифровања је сложенија него у случају *SPINE* система, али је и даље реч о нестандартном криптографском алгоритму. *PINOT* користи поједностављену супституционо-пермутациону мрежу ширине 56 бита у две фазе ради постизања већег протока пакета. Како ова нестандартна шема шифровања није подвргнута анализи од стране криптографске заједнице није позната ни њена отпорност на потенцијалне нападе. Као и код *SPINE* система, пошто је дужина шифрованог текста већа од *IPv4* адресе, и да би процес био реверзибилан при повратку пакета са другог краја комуникације, шифровани подаци се кодирају у *IPv6* пакет. Систем не чува стање за излазне изворишне *IP* адресе јер за њих није потребна табела претраживања пошто се шифрују коришћењем локалног тајног кључа. Међутим, претраживање је неопходно за одредишну *IPv6* адресу, која се мора пронаћи на основу одредишне *IPv4* адресе. Аутори *PINOT* су претпоставили да се користи одређена врста пресретања *DNS* саобраћаја (енгл. *DNS snooping*), са пресретањем и мапирањем *A* и *AAAA* записа на мрежном уређају, како би се могла креирати одговарајућа одредишна *IPv6* адреса. Међутим, за уобичајене мрежне уређаје, овакав процес пресретања *DNS* саобраћаја не делује тривијално и притом доводи до губитка перформанси. Размена криптографских кључева у *PINOT* систему није потребна у општем случају. Уколико пакети излазе и улазе преко исте тачке мреже и ако не постоји више улазних тачака у мрежу, кључеви су локални за мрежни уређај.

Нажалост, системи за очување приватности као што су *PINOT* и *SPINE*, који користе

транслацију из *IPv4* у *IPv6* адресу, данас не могу обезбедити пуну интернет повезаност. У тренутку писања ове докторске дисертације, оквирно само једна трећина свих аутономних система на интернету подржава *IPv6* [70]. Штавише, када *IPv6* буде у потпуности усвојен и постане доминантан *IP* протокол на интернету, примена истог оваквог приступа за *IPv6* адресе и транспортни слој биће изазовна, уколико не и немогућа. *PINOT* и *SPINE* користе чињеницу да су *IPv6* адресе дуже од *IPv4* адреса, што омогућава складиштење шифрованог текста *IPv4* адресе и случајне једнократне вредности у склопу *IPv6* адресе. Међутим, када се случајна једнократна вредност дода *IPv6* адреси, резултујући шифровани текст биће дужи од доступног простора у *IPv6* адреси. Поставља се питање где би се вишак бита шифрованог текста могао складиштити: или у новом заглављу протокола или коришћењем операције која чува стање.

Табела 2.1: Поређење постојећих решења по кључним карактеристикама

Карактеристика	<i>TOR</i>	<i>HORNET</i>	<i>TARANET</i>	<i>LAP</i>	<i>Dovetail</i>	<i>PHI</i>	<i>PANEL</i>	<i>AHP</i>	<i>SPINE</i>	<i>PINOT</i>	<i>LISPP</i>
Нема слојевитог шифровања	○	○	○	●	●	●	●	●	●	●	●
Нема шифровања корисног дела пакета	○	○	○	●	●	○	●	●	●	●	●
Занемарљив утицај на перформансе	○	○	○	●	●	○	●	●	●	●	●
Транспарентност за корисника	○	○	○	○	○	○	●	●	●	●	●
Стање сесије није у пакету	○	○	○	○	○	○	●	◐	◐	◐	●
Стање сесије се не чува на уређајима	●	●	●	●	●	●	○	○	●	●	●
Компатибилност са постојећим мрежама	●	○	○	○	○	●	●	●	●	●	●
Подршка и за <i>IPv4</i> и за <i>IPv6</i>	●	○	○	○	○	○	●	○	○	○	●
Локализована примењивост	●	○	◐	◐	◐	○	●	◐	◐	●	●
Гаранција анонимности	●	●	●	◐	◐	◐	◐	◐	◐	◐	◐
Анонимност и изворишта и одредишта	●	●	●	○	●	●	○	○	●	○	○
Нема цурења информација о путањи	●	●	●	○	○	●	●	○	○	○	○
Поверење у први чвор није неопходно	●	●	●	○	●	●	○	○	○	○	○
Извориште одређује путању пакета	●	●	●	○	●	○	○	○	○	○	○

Карактеристике постојећих решења су приказане у оквиру табеле 2.1. Постојећа решења, која су предложила шифровање критичних поља заглавља пакета користећи класичне алгоритме за блоковско шифровање, показала су да такав приступ успешно прикрива *IP* адресе корисника. Овим је потврђена полазна хипотеза да је успостављање корелације између пакета посматраног корисника могуће отежати замагљивањем делова заглавља пакета. Међутим, блоковске шифре захтевају или операције које чувају стање или превођење из *IPv4* у *IPv6* пакете, што доводи до значајних проблема у имплементацији и коришћењу, као што је претходно објашњено. Може се закључити да важи и полазна хипотеза да постојећи механизми заштите приватности на мрежном слоју имају проблеме у погледу перформанси услед интензивне примене алгоритама шифровања или чувања стања током рада.

У оквиру табеле 2.1 јасно се истичу два суштински различита приступа у погледу обезбеђивања приватности на мрежном слоју. Први приступ тежи да обезбеди истовремену анонимност и изворишта и одредишта, да сакрије целу путању пакета и да избегне наметање поверења у први чвор. Наведени строги захтеви намећу и сложену имплементацију, због чега се овај приступ колоквијално може назвати *тежким* приступом. Његови представници, као што су *Tor*, *HORNET* и *TARANET*, пружају висок ниво заштите и сврставају се у системе са јаким гаранцијама приватности. Карактеристике ових система су јасно уочљиве у доњем левом углу табеле 2.1. Насупрот томе, други приступ има мање строге захтеве што значајно поједностављује имплементацију, те се стога може назвати *лаким* приступом. Његови представници, попут *AHP*, *SPINE* и *PINOT*, генерално пружају нижи ниво заштите приватности и сврставају се у системе са релаксираним гаранцијама приватности. Карактеристике ових система груписане су у горњем десном углу табеле 2.1. Кључне предности лаког приступа огледају се у транспарентности за корисника, занемарљивом утицају на перформансе и могућности тренутне примене без захтевних интервенција на постојећој архитектури интернета. За лаки приступ постоје разумне претпоставке и реални сценарији коришћења, као што је, на пример, случај универзитетске мреже која штити приватност својих запослених и студената. Управо због практичне применљивости и могућности непосредне имплементације у савременим мрежним окружењима, у овој дисертацији је као правац истраживања изабран лаки приступ односно системи са релаксираним гаранцијама приватности.

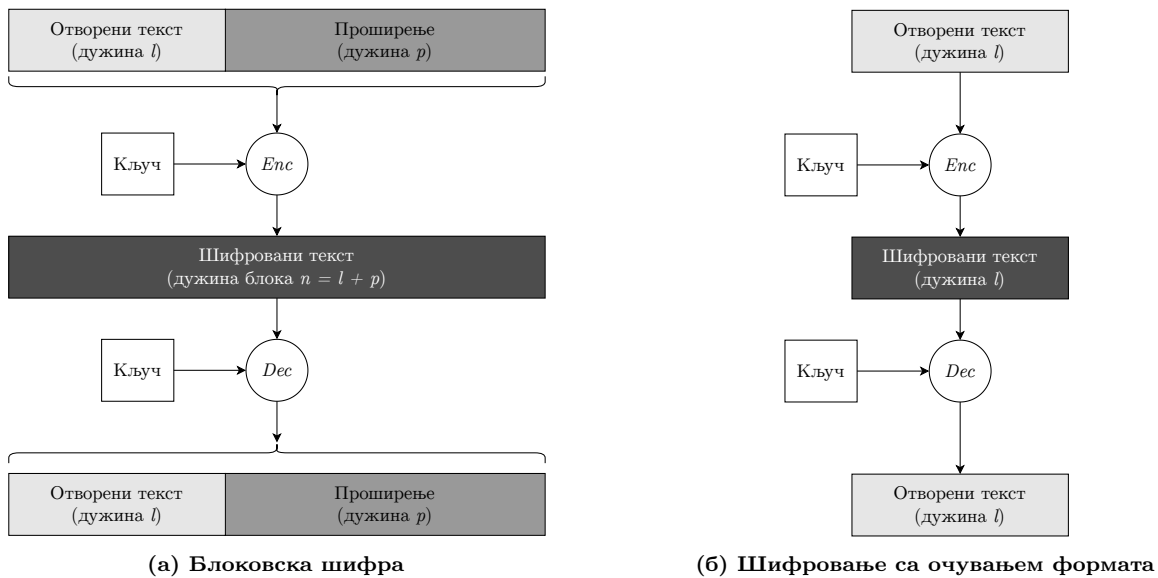
Ова докторска дисертација истражује могућност коришћења шифровања са очувањем формата за замагљивање поља заглавља пакета из чега проистиче и новопредложени *LISPP* систем са релаксираним гаранцијама приватности. *LISPP* систем не чува стање и ефикасно шифрује податке о мрежном току користећи *FPE*, чиме замагљује изворишну *IPv4* или *IPv6* адресу од посматрача на интернету. Овим приступом се онемогућава профилисање корисника, без повећања величине пакета или измештања информација у друга заглавља. *LISPP* систем је пројектован за приступачне *SmartNIC* уређаје у три различите имплементације, као и за *eBPF* технологију како би се што лакше могао применити захваљујући избегавању зависности од наменских мрежних уређаја. Све имплементације показују перформансе адекватне за различите примене што ће бити детаљније изложено у наредним поглављима.

3. Шифровање са очувањем формата

Ова докторска дисертација предлаже систем без чувања стања за заштиту приватности на мрежном слоју, који шифровањем замагљује само неопходне делове заглавља пакета ради заштите приватности корисника. Овакав приступ употребом добро познатих симетричних алгоритама, као што је на пример Напредни стандард за шифровање (енгл. *advanced encryption standard, AES*), није изводљив јер су поља заглавља пакета обично краћа и нису поравната са величином блока код блоковских шифара нити са границом бајтова код проточних шифара (енгл. *stream ciphers*). На пример, ако се 12-битни отворени текст шифрује користећи *AES-128* неопходно је дати отворени текст проширити до величине блока од 128 бита (обично додавањем нула или случајних података), а резултат шифровања биће 128-битни шифровани текст. Да би се реконструисао иницијални 12-битни отворени текст, шифровани текст од 128 бита негде мора бити сачуван у целини ради накнадног успешног дешифровања и уклањања података додатих као проширење зарад поравнања на величину блока; овај процес приказан је на слици 3.1а. Ако је 12-битни отворени текст заправо поље заглавља пакета или само један његов део, на пример хост део *IP* адресе са маском /20, који треба проширити и затим шифровати, чување шифрованог текста захтева додатни простор за складиштење, што може подразумевати потребу за увођењем нових заглавља или протокола. Претходно излагање је у складу са полазном хипотезом да употреба блоковских алгоритама шифровања у циљу замагљивања делова заглавља пакета није прикладна јер није могуће постићи довољно фину грануларност замагљивања, што доводи до нарушавања стандардног формата тих заглавља или до потребе за сложеним праћењем стања активних сесија. Због тога ова докторска дисертација истражује могућност примене *FPE* у циљу заштите приватности на мрежном слоју. *FPE* врши шифровање отвореног текста произвољне дужине, без проширивања, омогућавајући тиме замену поља заглавља произвољног протокола шифрованим текстом исте величине. Принцип рада *FPE*, као контраст у односу на блоковске шифре, приказан је на слици 3.1б.

FPE је тип шифровања који у склопу шифрованог текста чува дужину и формат отвореног текста. Отворени и шифровани текст имају исту дужину и сачињени су од истог скупа симбола. Један од првих алгоритама са подршком за променљиву дужину улаза посматрано на нивоу бита и исти формат излаза био је *Hasty Pudding Cipher (HPC)* [71], један од кандидата на *AES* такмичењу. *HPC* алгоритам није прошао у касније фазе *AES* такмичења због своје сложене и неуобичајене структуре. Као директна последица тога, изостала је детаљнија анализа *HPC* алгоритма од стране криптографске заједнице, па његова отпорност на различите нападе није детаљно позната.

Први формално стандардизовани *FPE* алгоритми су *FF1* и *FF3-1* алгоритми [72] препоручени 2016. године од стране америчког Националног института за стандарде и технологију (енгл. *National Institute of Standards and Technology, NIST*), као и *FEA-1* и *FEA-2* алгоритми [73] стандардизовани 2015. године од стране корејског Удружења за телекомуникационе технологије (енгл. *Telecommunications Technology Association, TTA*) под окриљем Корејске агенције за технологију и стандарде (енгл. *Korean Agency for Technology and Standards, KATS*). Сви наведени *FPE* алгоритми имају врло сличну Фајстелову (енгл. *Feistel*) структуру са параметром модификације. Фајстелова структура је добро познати



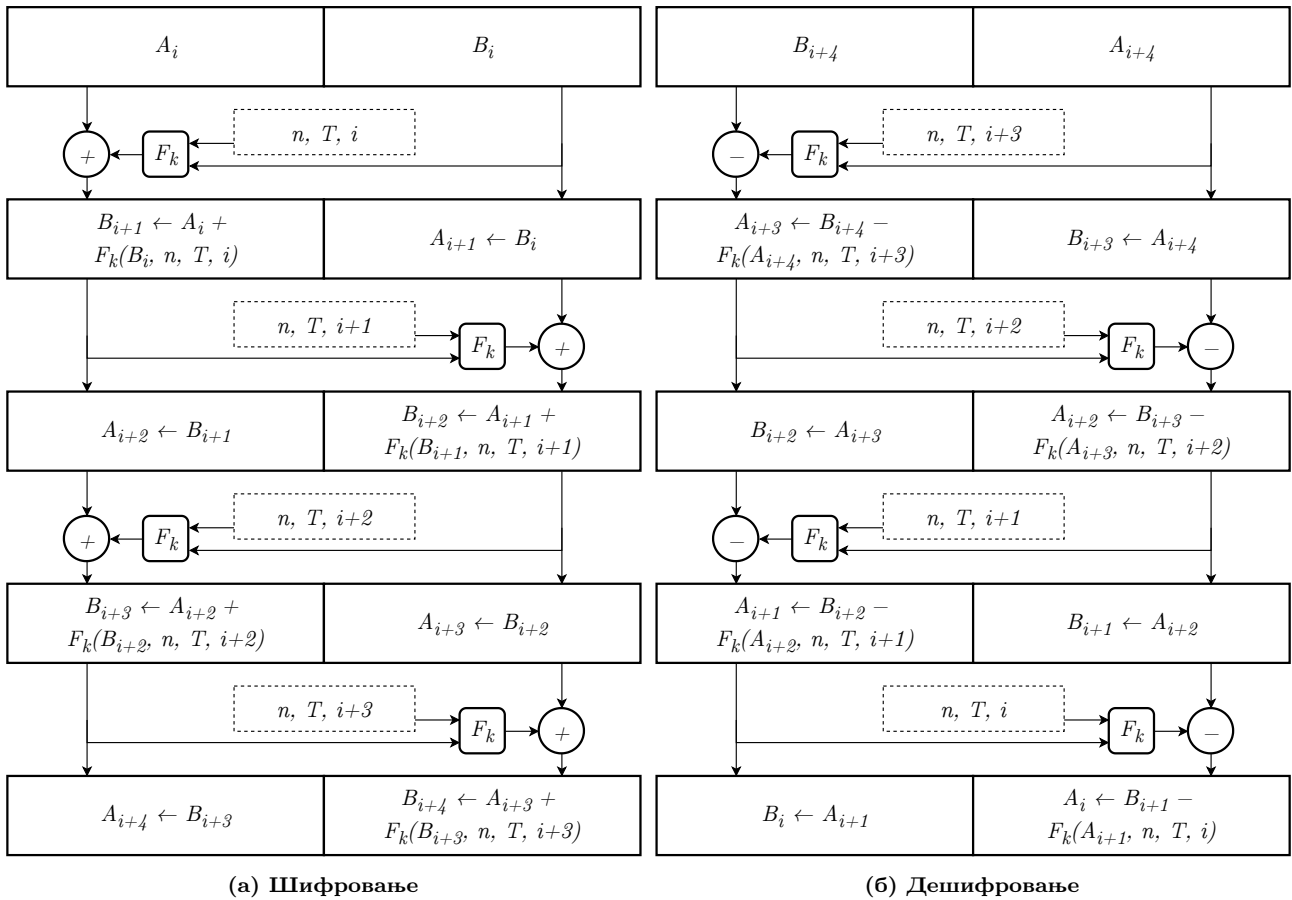
Слика 3.1: Разлика између блоковске шифре и шифровања са очувањем формата

основни блок за пројектовање симетричних алгоритама познат још из периода пре усвајања Стандарда за шифровање података (енгл. *Data Encryption Standard, DES*). Ово подразумева да се улазни податак раздваја на два дела, функција рунде (енгл. *round function*) се примењује на један део како би се затим помоћу добијеног резултата модификовао други део, док се на крају сваке рунде замењују улоге ова два дела пред почетак наредне рунде. Подржане псеудослучајне функције (енгл. *pseudorandom function*) за функцију рунде се разликују између *NIST* препоруке и *KATS* стандарда, што ће бити описано у наредним подсекцијама. Параметар модификације представља додатну компоненту криптографског алгорита, која се користи као улаз процеса шифровања и дешифровања. Параметар модификације не мора нужно бити тајан. Параметар модификације се користи да повећа варијабилност улазних података, јер у случају *FPE* улазни отворени текстови могу бити кратки и сачињени од ограниченог скупа симбола. Сви наведени *FPE* алгоритми користе параметар модификације управо ради уношења додатне варијабилности потребне због неретко кратких отворених текстова [74, 75]. Препоручује се шифровање што мањег броја отворених текстова са једном истом вредношћу параметра модификације. Међутим, промена вредности параметра модификације потенцијално може имати негативан утицај зависно од контекста употребе. На пример, промена вредности параметра модификације у контексту шифровања *IP* адреса може изазвати прекид постојећих сесија, због тога што ће *IP* адреса крајње тачке комуникације бити измењена, те се промена вредности параметра модификације мора обављати у пажљиво дефинисаним тренуцима.

3.1 *NIST FPE* препоруке

FF1 и *FF3-1* су симетрични алгоритми шифровања засновани на Фајстеловој структури са параметром модификације. У основи сваке рунде налази се одобрена блоковска шифра коришћена као функција рунде, са ознаком F_K , за генерисање псеудослучајне вредности. Слика 3.2 приказује четири Фајстелове рунде *FF1* односно *FF3-1* шифровања и дешифровања. Улазни отворени текст је подељен на два дела (A_i и B_i). Ова два дела су једнаке дужине ако је број карактера отвореног текста паран, или се разликују у дужини за један карактер ако је број карактера отвореног текста непаран. У контексту обраде мрежних пакета, један карактер одговара једном биту, јер су поља заглавља пакета бинарне речи. Други део (B_i) се копира у први део наредне рунде (A_{i+1}), док се први део додаје на излаз

функције рунде F_K . Улазне вредности функције рунде су један део претходног блока, број рунде i , параметар модификације T и дужина отвореног текста n .



Слика 3.2: Четири фајстелове рунде алгоритама $FF1$ и $FF3-1$

Функција рунде F_K , према *NIST* препорукама, мора бити нека од одобрених блоковских шифара са тајним кључем K , а у овом тренутку само *AES-128* блоковска шифра одговара том профилу. Међутим, постоје имплементације које одступају од *NIST* препорука и за функцију рунде користе и друге једноставније алгоритме од *AES* блоковске шифре [76]. $FF1$ и $FF3-1$ не користе блоковску шифру односно *AES* директно за шифровање и дешифровање података, већ помоћу њега само генеришу псеудослучајну вредност која се затим скраћује на потребан број бита и додаје половини отвореног текста. Ово имплицира да се унутар шифровања и дешифровања користи искључиво *AES* шифровање, што донекле поједностављује имплементацију ових алгоритама. Са друге стране, $FF3-1$ и $FF1$ се састоје од осам и десет рунди, респективно, при чему се у свакој рунди врши једно или више *AES* шифровања 128-битног блока, што стандардне имплементације $FF3-1$ и $FF1$ алгоритама чини оквирно упоредивим са шифровањем тачно 1024 бита (осам блокова) и најмање 2560 бита (двадесет блокова) помоћу *AES*, респективно, и то може представљати изазов за перформансе. Међутим, поглавље 6 показује да чак и чиста софтверска имплементација $FF3-1$ алгоритама може имати пропусни опсег на нивоу брзине везе на *SmartNIC* уређају или остварити проток пакета од 2,97 Mpps помоћу *eBPF* технологије.

Спецификација $FF1$ и $FF3-1$ алгоритама наводи неколико параметара који одређују њихово понашање. Алфавет (енгл. *alphabet*) је скуп два или више симбола односно карактера који се користе за представљање отвореног и шифрованог текста. База (енгл. *base*) је број карактера у посматраном алфавету и означава се још и као основа (енгл. *radix*). На пример, основа за бинарни отворени текст је 2, а основа за енглески отворени текст сачињен искључиво од малих слова је 26. Нумерал (енгл. *numeral*) за дату основу јесте ненегативни

цео број мањи од посматране основе. Број нумерала односно дужина (енгл. *length*) отвореног текста уз његову основу дефинишу величину домена отвореног текста као $radix^{length}$. На пример, величина домена за бинарни отворени текст дужине 14 нумерала је 2^{14} , а величина домена за енглески отворени текст сачињен искључиво од малих слова дужине 7 нумерала је 26^7 . Поред неколицине наведених параметара, спецификација такође наводи неколико синтаксних нотација, оператора и функција које су потребне за дефинисање алгоритама.

- 0^s , за задати број бита s , јесте низ бита вредности нула дужине s . На пример, за задати број бита важи $0^4 = 0, 0, 0, 0$.
- $[x]^s$, за задати број бајтова s и ненегативни цео број x мањи од 256^s , јесте представа x као низа бајтова дужине s . На пример, за задати број бајтова и ненегативни цео број важи $[13]^3 = 0x00, 0x00, 0x0D$.
- $X || Y$, за задате низове нумерала X и Y , јесте конкатенација X и Y . На пример, за задате низове нумерала важи $1, 2 || 3, 4, 5 = 1, 2, 3, 4, 5$.
- $BYTESLEN(X)$, за задати низ бајтова X , јесте број бајтова у X . На пример, за задати низ бајтова важи $BYTESLEN(0xB9, 0xAC) = 2$.
- $LEN(X)$, за задати низ нумерала односно бита X , јесте број нумерала односно бита у X . На пример, за задати низ нумерала важи $LEN(5, 2, 9) = 3$.
- $NUM(X)$, за задати низ бита X , јесте број који X представља, када се бити вреднују у редоследу опадајуће тежине. На пример, за задати низ бита важи $NUM(1, 1, 1, 0) = 14$.
- $NUM_{radix}(X)$, за задати низ нумерала X и основу $radix$, јесте број који X представља у основи $radix$ када се нумерали вреднују у редоследу опадајуће тежине. На пример, за задати низ нумерала и основу важи $NUM_4(2, 0, 2, 3) = 139$.
- $STR_{radix}^m(x)$, за задату основу $radix$, број нумерала m и ненегативни цео број x мањи од $radix^m$, јесте представа x као низа нумерала дужине m у основи $radix$ када се нумерали вреднују у редоследу опадајуће тежине. На пример, за задату основу, број нумерала и ненегативни цео број важи $STR_4^6(139) = 0, 0, 2, 0, 2, 3$.
- $CIPH_K(X)$, за задати улазни блок X и кључ K , јесте излаз функције рунде.
- $REV(X)$, за задати низ нумерала X , јесте низ нумерала из X у обрнутом редоследу. На пример, за задати низ нумерала у основи 10 важи $REV(1, 2, 3, 4) = 4, 3, 2, 1$.
- $REVB(X)$, за задати низ бајтова X , јесте низ бајтова из X у обрнутом редоследу. На пример, за задати низ бајтова важи $REVB(0xB9, 0xAC, 0x74) = 0x74, 0xAC, 0xB9$.

3.1.1 *FF1* алгоритам

FF1 алгоритам заснива се на *FFX* режиму рада (енгл. *mode of operation*) за шифровање са очувањем формата чија је спецификација поднета *NIST* у циљу стандардизације [77, 78]. *FFX* улазни параметри су отворени текст X , параметар модификације T и криптографски кључ K . Поред улазних параметара, *FFX* има и конфигурационе параметре зарад постизања веће флексибилности и прилагодљивости. *FFX* конфигурациони параметри су број Фајстелових рунди, жељени ниво неравнотеже Фајстелове мреже и функција рунде. Вредности конфигурационих параметара морају бити фиксне током читавог животног века криптографског кључа, а дефинисањем њихових вредности заправо се инстанцира један конкретан алгоритам *FFX* режима рада. *FFX* наводи пример два скупа конфигурационих

параметара за шифровање бинарних и децималних отворених текстова на основу којих се инстанцирају шифре *FFX-A2* и *FFX-A10* респективно. У оквиру спецификације, *NIST* препоручује тачно један скуп конфигурационих параметара чиме се суштински за *FF1* алгоритам елиминише поменута флексибилност и прилагодљивост. У оквиру *NIST* препоруке за *FF1* алгоритам, постоји тачно десет Фајстелових рунди, користи се максимално балансирана Фајстелова мрежа, док је за функцију рунде изабран *AES* алгоритам. Према томе, *NIST* је направио одступање само по питању броја Фајстелових рунди у односу на *FFX-A2* и *FFX-A10* скупове конфигурационих параметара који у зависности од величине отвореног текста предлажу и до тридесетшест рунди.

Алгоритам 3.1 *FF1* шифровање

```

1: procedure FF1-ENCRYPT( $K, T, X$ )
2:    $(n, t) \leftarrow (\text{LEN}(X), \text{BYTELEN}(T))$ 
3:    $(u, v) \leftarrow (\text{floor}(n/2), n - \text{floor}(n/2))$ 
4:    $(A, B) \leftarrow (X[0..(u-1)], X[u..(n-1)])$ 
5:    $(b, d) \leftarrow (\text{ceil}(\text{ceil}(v * \log_2(\text{radix}))/8), 4 * \text{ceil}(b/4) + 4)$ 
6:    $P \leftarrow [1]^1 \parallel [2]^1 \parallel [1]^1 \parallel [\text{radix}]^3 \parallel [10]^1 \parallel [u \bmod 256]^1 \parallel [n]^4 \parallel [t]^4$ 
7:   for  $i \leftarrow 0, 9$  do
8:      $Q \leftarrow T \parallel [0]^{(-t-b-1) \bmod 16} \parallel [i]^1 \parallel [\text{NUM}_{\text{radix}}(B)]^b$ 
9:      $R \leftarrow \text{CBC-CIPH}_K(P \parallel Q)$ 
10:     $S \leftarrow R$ 
11:    for  $j \leftarrow 1, \text{ceil}(d/16) - 1$  do
12:       $S \leftarrow S \parallel \text{CIPH}_K(R \oplus [j]^{16})$ 
13:    end for
14:    if  $i \bmod 2 = 0$  then
15:       $m \leftarrow u$ 
16:    else
17:       $m \leftarrow v$ 
18:    end if
19:     $c \leftarrow (\text{NUM}_{\text{radix}}(A) + \text{NUM}(S[0..(d * 8 - 1)])) \bmod \text{radix}^m$ 
20:     $C \leftarrow \text{STR}_{\text{radix}}^m(c)$ 
21:     $(A, B) \leftarrow (B, C)$ 
22:  end for
23:  return  $A \parallel B$ 
24: end procedure

```

Дефиниција *FF1* шифровања дата је унутар алгоритма 3.1. У склопу припреме за десет Фајстелових рунди, врши се раздвајање улазног низа нумерала X на два дела односно поднизове нумерала A и B у кораку 4, као и формирање низа бита односно блока P ширине 128 бита у кораку 6 као иницијалног блока за уланчавање шифрованих блокова (енгл. *cipher block chaining, CBC*). У склопу сваке рунде *FF1* шифровања, прво се формира низ бита Q ширине дељиве са 128 бита у кораку 8, енковањем параметра модификације T , броја рунде i и подниза нумерала B претвореног у цео број на ширини од b бајтова. Блок R ширине 128 бита добија се у кораку 9, применом псеудослучајне функције у виду CBC-CIPH_K на конкатенацију блока P и претходно добијеног низа бита Q . Низ бајтова S минималне дужине d елемената добија се у кораку 12, проширивањем претходно добијеног блока R до потребног броја елемената. Цео број c добија се у кораку 19 сабирањем вредности d елемената низа бајтова S са представом низа нумерала A у основи radix и одсецањем тако добијеног збира израчунавањем остатка при дељењу са radix^m . У кораку 20, цео број c се претвара у низ нумерала C у основи radix на ширини од m нумерала. Крај сваке

рунде представља припрему за наредну рунду односно смештање низа нумерала B у низ нумерала A и смештање претходно добијеног низа нумерала C у низ нумерала B . Након десет Фајстелових рунди, крајњи резултат $FF1$ шифровања добија се конкатенацијом низова нумерала A и B . Дефиниција $FF1$ дешифровања дата је унутар алгоритма 3.2.

Алгоритам 3.2 $FF1$ дешифровање

```

1: procedure FF1-DECRYPT( $K, T, X$ )
2:    $(n, t) \leftarrow (\text{LEN}(X), \text{BYTELEN}(T))$ 
3:    $(u, v) \leftarrow (\text{floor}(n/2), n - \text{floor}(n/2))$ 
4:    $(A, B) \leftarrow (X[0..(u-1)], X[u..(n-1)])$ 
5:    $(b, d) \leftarrow (\text{ceil}(\text{ceil}(v * \log_2(\text{radix}))/8), 4 * \text{ceil}(b/4) + 4)$ 
6:    $P \leftarrow [1]^1 \parallel [2]^1 \parallel [1]^1 \parallel [\text{radix}]^3 \parallel [10]^1 \parallel [u \bmod 256]^1 \parallel [n]^4 \parallel [t]^4$ 
7:   for  $i \leftarrow 9, 0$  do
8:      $Q \leftarrow T \parallel [0]^{(-t-b-1) \bmod 16} \parallel [i]^1 \parallel [\text{NUM}_{\text{radix}}(A)]^b$ 
9:      $R \leftarrow \text{CBC-CIPH}_K(P \parallel Q)$ 
10:     $S \leftarrow R$ 
11:    for  $j \leftarrow 1, \text{ceil}(d/16) - 1$  do
12:       $S \leftarrow S \parallel \text{CIPH}_K(R \oplus [j]^{16})$ 
13:    end for
14:    if  $i \bmod 2 = 0$  then
15:       $m \leftarrow u$ 
16:    else
17:       $m \leftarrow v$ 
18:    end if
19:     $c \leftarrow (\text{NUM}_{\text{radix}}(B) - \text{NUM}(S[0..(d * 8 - 1)])) \bmod \text{radix}^m$ 
20:     $C \leftarrow \text{STR}_{\text{radix}}^m(c)$ 
21:     $(A, B) \leftarrow (C, A)$ 
22:  end for
23:  return  $A \parallel B$ 
24: end procedure

```

$FF1$ пружа флексибилност у погледу дужине параметра модификације. Његова употреба код $FF1$ алгоритма је опциона у смислу да је прихватљив и параметар модификације дужине нула. Ограничење за параметар модификације постоји само по питању његове максималне дужине.

$FF1$ захтева да минимална величина домена отвореног текста буде најмање 1 милион. Према томе, за бинарне улазе сваки отворени текст који је једнак или дужи од 20 бита испуњава овај услов спецификације алгоритма. За $LISPP$, ово ограничење суштински значи да је /28 најдужа мрежна маска, која се може користити као улаз, чиме се гарантује захтевана минимална величина домена отвореног текста: 4 бита за хост део IP адресе и 16-битни изворишни порт. Међутим, тако мала мрежа са само 14 крајњих уређаја сама по себи потенцијално уноси другачији вид ризика по приватност корисника. Пошто је број корисника који користе мрежу мали, већа је вероватноћа да дође до напада споредним каналима (енгл. *side-channel attack*) који анализирају активност корисника у одређеном периоду. Бољи резултати се постижу код већих мрежа са краћим мрежним маскама и више корисника; јача анонимност се постиже уколико се корисник скрива у што већем анонимном скупу [79]. Мрежна маска највеће дужине /24 може се зато сматрати препоруком.

Друго ограничење $FF1$ је максимална дужина отвореног текста, која мора бити мања од 2^{32} карактера независно од основе. Ово ограничење превазилази сваку примену у домену шифровања поља заглавља пакета за више редова величина. Занимљиво је приметити

корак 9 алгоритма 3.1, као и корак 9 алгоритма 3.2, из којих се види да се *CBC* примењује на читав параметар модификације T и половину улазног низа нумерала након представљања у виду броја у одговарајућој основи на ширини од b бајтова. Како параметар модификације може бити произвољне дужине, а b директно зависи од дужине отвореног текста чији је максимум 2^{32} , ово сугерише да је у свакој *FF1* рунди могуће извршити већи број *AES* шифровања. У кораку 12 алгоритма 3.1, као и кораку 12 алгоритма 3.2, може се приметити $\text{ceil}(d/16) - 1$ додатних *AES* шифровања која се извршавају у свакој рунди. Ово сугерише да су перформансе *FF1* алгоритма у великој мери под утицајем дужине отвореног текста и параметра модификације од којих зависи број извршених *AES* шифровања у склопу сваке рунде.

3.1.2 *FF3-1* алгоритам

FF3-1 алгоритам заснива се на блоковској шифри *BC*, променљиве ширине блока, интерно коришћене од стране алгоритма *BPS* [80]. Шифра *BC* је конфигурабилна по питању ширине блока и основе улаза. *NIST* је модификовао *BC* шифру у извесној мери како би се побољшала њена сигурност. Величина параметра модификације код *FF3-1* алгоритма је смањена са 64 бита на 56 бита, што је проузроковало и измену начина његове поделе на леви и десни део. У оквиру спецификације, *NIST* препоручује искључиво *AES* алгоритам за функцију рунде, иако *BC* шифра подржава и друге алгоритме као што је *SHA* хеш функција. *FF3-1* алгоритам је ревидиран након што је у иницијалној верзији пронађен сигурносни пропуст [81]. Линеарна криптоанализа *FPE* алгоритма [82] открила је да су напади на *FF3-1* временски захтевнији у односу на друге алгоритме, у смислу броја операција шифровања, чиме се истиче његова безбедност. Коначно, могућност *FF3-1* алгоритма да шифрује бинарне речи дужине од 20 до 192 бита учинила га је посебно погодним за шифровање издвојених поља произвољних протокола што је искоришћено за потребе развоја *LISPP* система.

Алгоритам 3.3 *FF3-1* шифровање

```

1: procedure FF3-1-ENCRYPT( $K, T, X$ )
2:    $n \leftarrow \text{LEN}(X)$ 
3:    $(u, v) \leftarrow (\text{ceil}(n/2), n - \text{ceil}(n/2))$ 
4:    $(A, B) \leftarrow (X[0..(u-1)], X[u..(n-1)])$ 
5:    $(Tl, Tr) \leftarrow (T[0..27] \parallel 0^4, T[32..55] \parallel T[28..31] \parallel 0^4)$ 
6:   for  $i \leftarrow 0, 7$  do
7:     if  $i \bmod 2 = 0$  then
8:        $(m, W) \leftarrow (u, Tr)$ 
9:     else
10:       $(m, W) \leftarrow (v, Tl)$ 
11:    end if
12:     $P \leftarrow W \oplus [i]^4 \parallel [\text{NUM}_{radix}(\text{REV}(B))]^{12}$ 
13:     $S \leftarrow \text{REVB}(\text{CIPH}_{\text{REVB}(K)}(\text{REVB}(P)))$ 
14:     $c \leftarrow (\text{NUM}_{radix}(\text{REV}(A)) + \text{NUM}(S)) \bmod radix^m$ 
15:     $C \leftarrow \text{REV}(\text{STR}_{radix}^m(c))$ 
16:     $(A, B) \leftarrow (B, C)$ 
17:  end for
18:  return  $A \parallel B$ 
19: end procedure

```

FF3-1 вреднује нумерале у редоследу растуће тежине, што је конвенција супротна у

односу на *FF1* алгоритам. Управо из тог разлога, дефиниција *FF3-1* алгоритма садржи позиве функција *REV* и *REVB*. Дефиниција *FF3-1* шифровања дата је унутар алгоритма 3.3. У склопу припреме за осам Фајстелових рунди, у корацима 4 и 5, врши се раздвајање улазног низа нумерала X на два дела односно поднизове нумерала A и B , као и раздвајање параметра модификације T на два дела односно леви и десни параметар модификације Tl и Tr . У склопу сваке рунде *FF3-1* шифровања, прво се формира низ бита односно блок P ширине 128 бита у кораку 12, применом логичке операције ексклузивно ИЛИ на број рунде i и леви односно десни параметар модификације зависно од парности броја рунде, да би се затим извршила конкатенација тако добијеног резултата са представом низа нумерала B у основи *radix* на ширини од 12 бајтова. Блок S ширине 128 бита добија се у кораку 13, шифровањем претходно формираног блока P применом *AES* алгоритма под кључем K . Цео број c добија се у кораку 14 сабирањем вредности блока S са представом низа нумерала A у основи *radix* и одсецањем тако добијеног збира израчунавањем остатка при дељењу са $radix^m$. У кораку 15, цео број c се претвара у низ нумерала C у основи *radix* на ширини од m нумерала. Крај сваке рунде представља припрему за наредну рунду односно смештање низа нумерала B у низ нумерала A и смештање претходно добијеног низа нумерала C у низ нумерала B . Након осам Фајстелових рунди, крајњи резултат *FF3-1* шифровања добија се конкатенацијом низова нумерала A и B . Дефиниција *FF3-1* дешифровања дата је унутар алгоритма 3.4.

Алгоритам 3.4 *FF3-1* дешифровање

```

1: procedure FF3-1-DECRYPT( $K, T, X$ )
2:    $n \leftarrow \text{LEN}(X)$ 
3:    $(u, v) \leftarrow (\text{ceil}(n/2), n - \text{ceil}(n/2))$ 
4:    $(A, B) \leftarrow (X[0..(u-1)], X[u..(n-1)])$ 
5:    $(Tl, Tr) \leftarrow (T[0..27] \parallel 0^4, T[32..55] \parallel T[28..31] \parallel 0^4)$ 
6:   for  $i \leftarrow 7, 0$  do
7:     if  $i \bmod 2 = 0$  then
8:        $(m, W) \leftarrow (u, Tr)$ 
9:     else
10:       $(m, W) \leftarrow (v, Tl)$ 
11:    end if
12:     $P \leftarrow W \oplus [i]^4 \parallel [\text{NUM}_{radix}(\text{REV}(A))]^{12}$ 
13:     $S \leftarrow \text{REVB}(\text{CIPH}_{\text{REVB}(K)}(\text{REVB}(P)))$ 
14:     $c \leftarrow (\text{NUM}_{radix}(\text{REV}(B)) - \text{NUM}(S)) \bmod radix^m$ 
15:     $C \leftarrow \text{REV}(\text{STR}_{radix}^m(c))$ 
16:     $(A, B) \leftarrow (C, A)$ 
17:  end for
18:  return  $A \parallel B$ 
19: end procedure

```

FF3-1 захтева да минимална величина домена отвореног текста буде најмање 1 милион. Према томе, за бинарне улазе сваки отворени текст који је једнак или дужи од 20 бита испуњава овај услов спецификације алгоритма. У оквиру *LISPP* система, услов за минималну дужину отвореног текста је свакако испуњен имајући у виду дискусију у склопу подсекције 3.1.1 о анонимности скривањем у што већем анонимном скупу. У случају максималне препоручене дужине мрежне маске /24, дужина отвореног текста је 24 бита укупно: 8 бита за хост део *IP* адресе и 16-битни изворишни порт, што је у збиру довољно за испуњење захтева *FF3-1* спецификације.

Друго ограничење *FF3-1* је максимална дужина отвореног текста, која мора бити

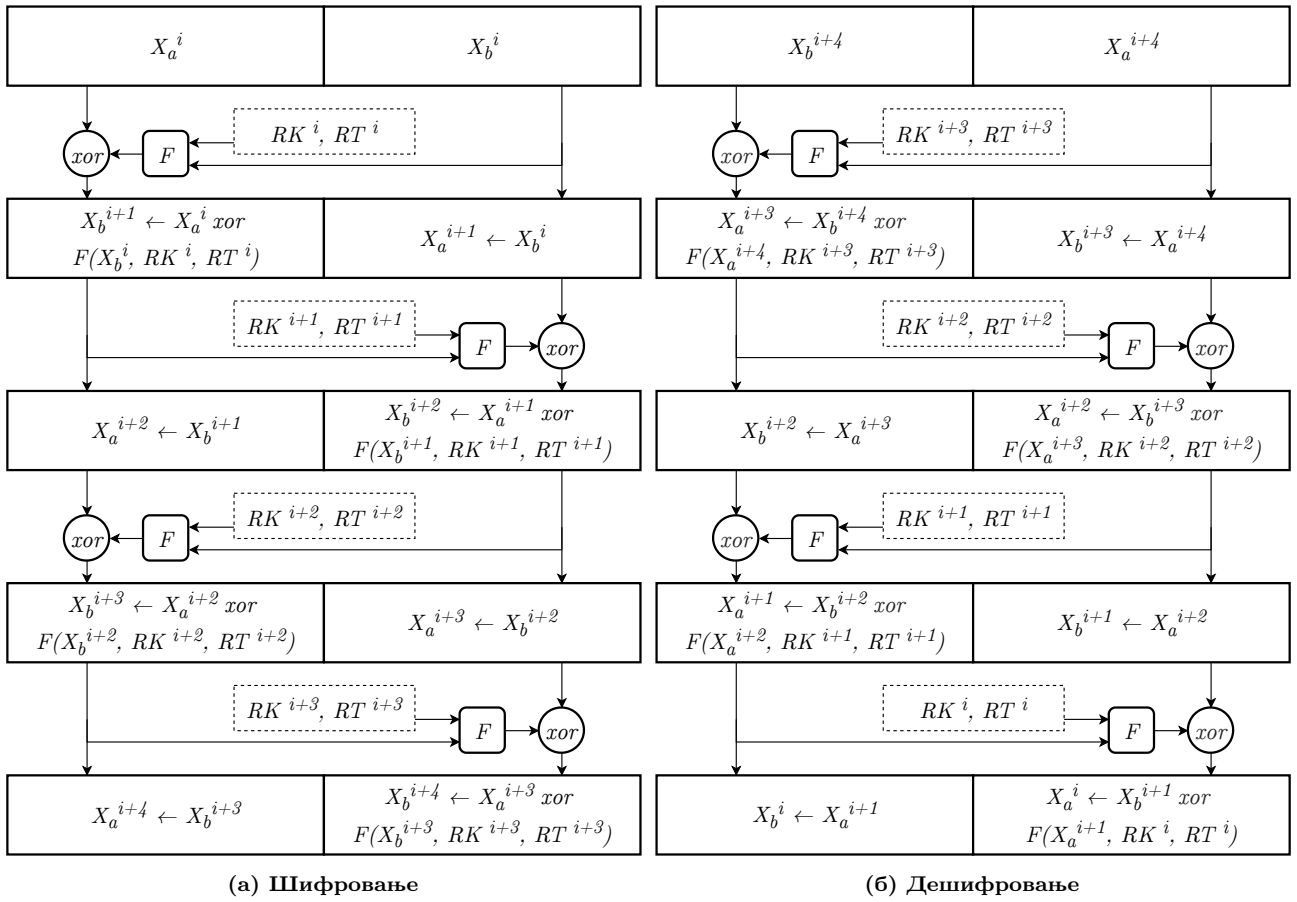
мања од $2 * \log_{radix}(2^{96})$ карактера. Према томе, за бинарне улазе максимална дужина отвореног текста је 192 бита, што је довољно за већину могућих примена над пољима заглавља пакета и за *IPv4* и за *IPv6*. У контексту шифровања поља заглавља пакета помоћу *LISPP* система, ово подразумева да се изворишни порт и целокупна *IPv6* адреса могу шифровати помоћу *FF3-1* без достизања теоријског ограничења алгоритма. Занимљиво је приметити корак 13 алгоритма 3.3, као и корак 13 алгоритма 3.4, из којих се види да се излаз *AES* алгоритма суштински примењује на половину отвореног текста, која може имати максималну дужину од 96 бита. Ово сугерише да је у свакој *FF3-1* рунди довољно извршити само једно *AES* шифровање у свим случајевима, без обзира на величину отвореног текста из опсега подржаних величина. Перформансе *FF3-1* алгоритма стога би требало да буду приближно исте за било коју дозвољену дужину отвореног текста. Експериментална евалуација спроведена у склопу ове докторске дисертације показала је да су перформансе система заиста биле исте без обзира на дужину маске *IP* адресе. *FF3-1* постиже већи проток јер има осам рунди, две мање у односу на *FF1*, а притом у свакој рунди има мањи број *AES* шифровања. Пошто би проток пакета требало да буде што већи, у оквиру докторске дисертације фокус је стављен управо на *FF3-1*.

3.2 *KATS FPE* стандарди

Непосредно пре првих активности америчког *NIST* на пољу стандардизације алгоритама за шифровање са очувањем формата, значајне кораке предузела су и друга национална тела, пре свега корејски *TTA*. У овој секцији приказани су стандарди дефинисани од стране овог тела, а који су усвојени под окриљем *KATS*. Реч је о алгоритмима *FEA-1* и *FEA-2* (енгл. *Fast Encryption Algorithm*) који деле исти фундаментални архитектонски приступ као и претходно описане *NIST* препоруке, ослањајући се на Фајстелову структуру са параметром модификације како би се обезбедило очување формата шифрованог податка. Међутим, кључна тачка дивергенције између ова два приступа огледа се у моделу саме функције рунде. Док су алгоритми *FF1* и *FF3-1* конципирани тако да се ослањају на робусност и проверену сигурност постојећег *AES* алгоритма, пројектанти *FEA* фамилије ставили су примарни фокус на рачунарску ефикасност и брзину извршавања. Сходно томе, уместо позивања сложене стандардне блоковске шифре, као што је *AES* алгоритам, *FEA* алгоритми користе наменски развијене, лаке блоковске шифре унутар својих рунди, што им омогућава постизање бољих перформанси у окружењима где су рачунарски ресурси ограничени или је ниска латенција од критичног значаја.

У основи сваке рунде *FEA-1* и *FEA-2* алгоритама налази се наменски развијена блоковска шифра коришћена као функција рунде, са ознаком F , за генерисање псеудослучајне вредности. Слика 3.3 приказује четири Фајстелове рунде *FEA-1* односно *FEA-2* шифровања и дешифровања. Улазни отворени текст је подељен на два дела (X_a^i и X_b^i). Ова два дела су једнаке дужине ако је број карактера отвореног текста паран, или се разликују у дужини за један карактер ако је број карактера отвореног текста непаран. Други део (X_b^i) се копира у први део наредне рунде (X_a^{i+1}), док први део представља један операнд операције логичко ексклузивно ИЛИ, а други операнд је излаз функције рунде F . Улазне вредности функције рунде су један део претходног блока, криптографски кључ за посматрану рунду RK^i добијен експанзијом кључа и параметар модификације за посматрану рунду RT^i добијен експанзијом параметра модификације.

Алгоритми *FEA-1* и *FEA-2* су базирани на фамилијама наменских блоковских шифара са параметром модификације који подржавају различите ширине блока, при чему се разликују два типа *type1* и *type2*, респективно. Наменске блоковске шифре са параметром модификације су изабране уместо модова рада постојећих стандардних алгоритама како



Слика 3.3: Четири фајстелове рунде алгоритама *FEA-1* и *FEA-2*

би се постигле боље перформансе. Сваки од ова два алгорита суштински представља дефиницију фамилије блоковских шифара са параметром модификације уз подршку за ширину блока од 8 до 128 бита. Алгоритми *FEA-1* и *FEA-2* су неколико пута бржи од *FF1*. Иако *FEA-1* и *FEA-2* спадају у FPE алгоритме, њихов алфабет формално чине нуле и јединице односно њихов домен представља скуп природних бројева. Ово није проблем зато што се било који алфабет може једноставно бијективно пресликати на скуп природних бројева. Максимална величина домена отвореног текста је 2^{128} што значи да је максимална ширина отвореног текста 128 бита. Поред неколицине наведених параметара, спецификација такође наводи неколико синтаксних нотација, оператора и функција које су потребне за дефинисање алгоритама.

- 0^s , за задати број бита s , јесте низ бита вредности нула дужине s . На пример, за задати број бита важи $0^4 = 0, 0, 0, 0$.
- $X || Y$, за задате низове бита X и Y , јесте конкатенација X и Y . На пример, за задате низове бита важи $1, 0 || 0, 0, 1 = 1, 0, 0, 0, 1$.
- $LEN(X)$, за задати низ бита X , јесте број бита у X . На пример, за задати низ бита важи $LEN(1, 0, 0, 0, 1) = 5$.
- $RC_{type, LEN(K), i}$ представља константу рунде ширине 64 бита, коришћену за наведену рунду i , ширину криптографског кључа $LEN(K)$ и тип $type$ наменске блоковске шифре са параметром модификације.
- $SBox(X)$, за задати низ бита X ширине 64 бита, јесте низ бита исте ширине добијен супституцијом помоћу осам паралелних *SBox* ширине 8 бита чији је садржај дат у

оквиру табеле 3.1, што представља стандардни супституциони блок.

- $M(X)$, за задати низ бита X ширине 64 бита, јесте низ бита исте ширине добијен множењем са 8×8 *MDS* матрицом над пољем Галоа $GF(2^8)$ која је приказана у оквиру табеле 3.2.

Табела 3.1: *FEA SBox*

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	62	31	70	8e	bc	30	9c	78	e0	5c	ce	bb	42	ac	b8	df
1	29	e7	86	5f	ee	ba	3f	87	c0	36	c3	14	7c	ec	73	da
2	57	72	f6	77	98	3b	c5	c4	4c	52	81	20	15	97	26	fc
3	8b	3c	af	6e	c8	7e	f0	40	24	a1	b1	54	ff	ad	51	bd
4	c1	13	41	b5	6b	94	63	d6	de	6f	89	d2	a9	d4	17	38
5	a5	f2	e3	db	47	66	ed	cb	4e	d5	05	60	8c	06	92	a3
6	be	68	56	a7	80	32	fa	6c	8f	88	d9	50	0a	21	3d	75
7	71	01	e5	7a	c6	b9	82	64	d1	00	7d	2b	a0	1a	5e	f5
8	35	90	2f	2a	83	49	5a	a8	d8	8d	46	96	dc	b0	c9	dd
9	cd	65	44	c7	43	67	55	eb	e1	9d	34	74	b3	4a	ca	d7
a	79	bf	f7	99	6a	2d	ef	85	e2	5d	fe	11	0f	19	cc	e4
b	58	09	8a	1b	6d	91	9f	4b	61	2c	2e	cf	27	10	18	b7
c	1d	0c	9b	39	7f	d3	84	a4	f9	76	33	f4	f3	d0	07	0e
d	22	1f	fd	25	12	08	1e	4d	b6	b4	53	37	e8	b2	9e	93
e	02	e9	f1	3a	0b	fb	45	69	ea	f8	c2	1c	04	59	03	48
f	16	a2	4f	3e	9a	23	aa	ae	5b	e6	95	ab	7b	0d	28	a6

Табела 3.2: *FEA* 8×8 *MDS* матрица над пољем Галоа $GF(2^8)$

$$\begin{pmatrix} 28 & 1a & 7b & 78 & c3 & d0 & 42 & 40 \\ 1a & 7b & 78 & c3 & d0 & 42 & 40 & 28 \\ 7b & 78 & c3 & d0 & 42 & 40 & 28 & 1a \\ 78 & c3 & d0 & 42 & 40 & 28 & 1a & 7b \\ c3 & d0 & 42 & 40 & 28 & 1a & 7b & 78 \\ d0 & 42 & 40 & 28 & 1a & 7b & 78 & c3 \\ 42 & 40 & 28 & 1a & 7b & 78 & c3 & d0 \\ 40 & 28 & 1a & 7b & 78 & c3 & d0 & 42 \end{pmatrix}$$

Процедура експанзије кључа, чија је дефиниција дата у виду алгоритма 3.5, прихвата криптографски кључ K и ширину улаза n . Како *FEA* алгоритми подржавају различите ширине за криптографски кључ, неопходно је прво извршити његово проширивање до фиксне ширине од 256 бита у кораку 2. Експанзија кључа се у потпуности реализује кроз $r/2$ итерација, где r представља укупан број рунди *FEA* алгоритма. Свака итерација користи властиту константу рунде $RC_{type, LEN(K), j}$ чија вредност зависи од итерације j , ширине криптографског кључа $LEN(K)$ и типа *type* наменске блоковске шифре са параметром модификације. У корацима 5 и 6 сваке итерације врши се супституција помоћу осам паралелних *SBox* ширине 8 бита чији је садржај дат у оквиру табеле 3.1 и пермутација множењем са 8×8 *MDS* матрицом над пољем Галоа $GF(2^8)$ која је приказана у оквиру табеле 3.2. На крају сваке појединачне итерације, генеришу се два кључа ширине 128 бита за две узаступне рунде што се види у кораку 11.

Алгоритам 3.5 *FEA* експанзија кључа

```
1: procedure FEA-KEYSCHEDULE( $K, n$ )
2:    $K \leftarrow K \parallel 0^{256-\text{LEN}(K)}$ 
3:    $(K_a, K_b, K_c, K_d) \leftarrow (K[0..63], K[64..127], K[128..191], K[192..255])$ 
4:   for  $j \leftarrow 1, \text{ceil}(r/2)$  do
5:      $X \leftarrow \text{M}(\text{SBox}(K_a \oplus K_c \oplus \text{RC}_{\text{type}, \text{LEN}(K), j}))$ 
6:      $Y \leftarrow \text{M}(\text{SBox}(K_b \oplus K_d \oplus X \oplus n))$ 
7:      $X \leftarrow X \oplus Y$ 
8:      $(K_a, K_b, K_c, K_d) \leftarrow (X \oplus K_a, Y \oplus K_b, X \oplus K_c, Y \oplus K_d)$ 
9:      $K_c \leftarrow K_c \oplus K_d$ 
10:     $K_d \leftarrow K_d \oplus K_c$ 
11:     $(RK^{2*j-1}, RK^{2*j}) \leftarrow (K_a \parallel K_b, K_c \parallel K_d)$ 
12:  end for
13:  return  $(RK^1, RK^2, \dots, RK^r)$ 
14: end procedure
```

Функција рунде, чија је дефиниција дата у виду алгоритма 3.6, прихвата половину улазног податка X , кључ рунде RK и параметар модификације рунде RT . Функција рунде представља композицију три различите операције при чему се друга операција примењује два пута узастопно. Прва операција Tw у кораку 3 јесте проширивање улаза функције рунде до ширине од 64 бита и додавање параметра модификације рунде логичком операцијом ексклузивног ИЛИ. Друга операција KSP у корацима 4 и 5 састоји се од слоја K за додавање половине кључа рунде, слоја S за супституцију и слоја P за пермутацију односно дифузију. Слој за супституцију је реализован помоћу осам паралелних *SBox* ширине 8 бита чији је садржај дат у оквиру табеле 3.1. Слој за пермутацију је реализован као множење са 8×8 *MDS* матрицом над пољем Галоа $GF(2^8)$ која је приказана у оквиру табеле 3.2. Трећа операција Tr у кораку 6 врши скраћивање добијене вредности на потребну ширину. Према томе, дефиниција функције рунде је $Tw \circ KSP \circ KSP \circ Tr$. На основу овога се јасно види да је сама функција рунде заснована на супституционо-пермутационој мрежи, што је једна од најчешће коришћених архитектура симетричних алгоритама поред Фајстелове структуре. Функција рунде подржава ширину улаза од 4 до 64 бита, што је једнако управо половини улаза *FEA* алгоритама која се доводи на улаз функције рунде.

Алгоритам 3.6 *FEA* функција рунде

```
1: procedure FEA-ROUNDFUNCTION( $X, RK, RT$ )
2:    $(RK_a, RK_b) \leftarrow (RK[0..63], RK[64..127])$ 
3:    $Y \leftarrow (X \parallel 0^{64-\text{LEN}(X)}) \oplus RT$ 
4:    $Y \leftarrow \text{M}(\text{SBox}(Y \oplus RK_a))$ 
5:    $Y \leftarrow \text{M}(\text{SBox}(Y \oplus RK_b))$ 
6:   return  $Y[0..(\text{LEN}(X) - 1)]$ 
7: end procedure
```

Операција *FEA* шифровања, чија је дефиниција дата у виду алгоритма 3.7, прихвата криптографски кључ K , параметар модификације T и отворени текст P . Како би за сваку од r рунди имали одговарајући кључ рунде и параметар модификације рунде, неопходно је извршити експанзију кључа у кораку 4 и одмах затим експанзију параметра модификације у кораку 5. У свакој од r рунди, имајући у виду да *FEA* алгоритам има Фајстелову структуру, примењује се функција рунде на један део улазног податка X_b^i , кључ рунде RK^i и параметар модификације рунде RT^i , да би се затим помоћу добијеног резултата

модификовао други део улазног податка X_a^i логичком операцијом ексклузивно ИЛИ у кораку 8, док се пред почетак наредне рунде замењују улоге ова два дела улазног податка. Независно од тога да ли је реч о $FEA-1$ или $FEA-2$ алгоритму односно да ли је реч о типу $type1$ или типу $type2$ структура операције шифровања је иста. Разлика између $FEA-1$ и $FEA-2$ алгоритма огледа се у укупном броју рунди, вредностима константи рунде и начину на који се врши експанзија параметра модификације. Укупан број рунди, вредности константи рунде и детаљи експанзије параметра модификације у случају $FEA-1$ и $FEA-2$ алгоритмима биће дати у подсекцијама 3.2.1 и 3.2.2 респективно.

Алгоритам 3.7 FEA шифровање

```

1: procedure FEA-ENCRYPTION( $K, T, P$ )
2:    $n \leftarrow \text{LEN}(P)$ 
3:    $(n_1, n_2) \leftarrow (\text{ceil}(n/2), \text{floor}(n/2))$ 
4:    $(RK^1, RK^2, \dots, RK^r) \leftarrow \text{FEA-KeySchedule}(K, n)$ 
5:    $(RT^1, RT^2, \dots, RT^r) \leftarrow \text{FEA-TweakSchedule}(T, n_1, n_2)$ 
6:    $(X_a^1, X_b^1) \leftarrow (P[0..(n_1 - 1)], P[n_1..(n_1 + n_2 - 1)])$ 
7:   for  $i \leftarrow 1, r$  do
8:      $X_b^{i+1} \leftarrow X_a^i \oplus \text{FEA-RoundFunction}(X_b^i, RK^i, RT^i)$ 
9:      $X_a^{i+1} \leftarrow X_b^i$ 
10:  end for
11:   $C \leftarrow X_b^{r+1} || X_a^{r+1}$ 
12:  return  $C$ 
13: end procedure

```

3.2.1 $FEA-1$ алгоритам

Фамилија алгоритма типа $type1$ односно $FEA-1$ подржава кључеве ширине 128, 192 и 256 бита. Зависно од ширине кључа број рунди мора бити 12, 14 и 16, респективно. Експанзија кључа се реализује у $r/2$ итерација, где је r укупан број рунди $FEA-1$ алгоритма, тако да је број константи рунде дупло мањи од броја рунди. Вредности константи рунде, означене одговарајућим бројем рунде i , дате су у табели 3.3 зависно од ширине криптографског кључа.

Табела 3.3: $FEA-1$ константе рунде за различите дужине криптографског кључа

i	$\text{LEN}(K) = 128$	$\text{LEN}(K) = 192$	$\text{LEN}(K) = 256$
1	71366fbd8eef2e7d	d2f928b5c6c08b51	8f1c67da8e609269
2	9063ff208a85d13f	4cbe190cdcc2962c	9b705f1835e0cdcd
3	fdb54b3c9a86cb08	d0a2a85f772c8a07	6bf524a08a50a621
4	f2ea772be55e4de0	e3fb1d49f5932802	6b3c821900adab39
5	7c8814f95b9f8d0b	047117eee8007dfe	1f0eb84f4de6881c
6	eb21fbffccbb8df5	4390e40073a64c7d	887fba6319cbf504
7		ee9fab45168ddadc	51547790dd0b8145
8			ad7c1f118ca88090

Трећи диференцијатор алгоритма типа $type1$ односно $FEA-1$, поред броја рунди и вредности константи рунде, јесте параметар модификације и његова експанзија. Параметар модификације има ширину од $128 - n$ бита, где је n дужина отвореног текста, и дели се на два дела T_L и T_R ширине $\text{floor}(n/2)$ и $\text{ceil}(n/2)$ бита, респективно, на основу којих се

зависно од броја рунде i формира параметар модификације рунде. Процедура експанзије параметра модификације, чија је дефиниција дата у виду алгоритма 3.8, прихвата параметар модификације T и дужине половина отвореног текста n_1 и n_2 .

Алгоритам 3.8 *FEA-1* експанзија параметра модификације

```

1: procedure FEA-TWEAKSCHEDULE( $T, n_1, n_2$ )
2:    $(T_L, T_R) \leftarrow (0^{n_2} \parallel T[0..(64 - n_2 - 1)], 0^{n_1} \parallel T[(64 - n_2)..(128 - n_2 - n_1 - 1)])$ 
3:   for  $i \leftarrow 1, r$  do
4:     if  $i \bmod 2 = 0$  then
5:        $RT^i \leftarrow T_R$ 
6:     else
7:        $RT^i \leftarrow T_L$ 
8:     end if
9:   end for
10:  return  $(RT^1, RT^2, \dots, RT^r)$ 
11: end procedure

```

3.2.2 *FEA-2* алгоритам

Фамилија алгоритама типа *type2* односно *FEA-2* такође подржава кључеве ширине 128, 192 и 256 бита. Зависно од ширине кључа број рунди мора бити 18, 21 и 24, респективно. Експанзија кључа се реализује у $r/2$ итерација, где је r укупан број рунди *FEA-2* алгоритма, тако да је број константи рунде дупло мањи од броја рунди. Вредности константи рунде, означене одговарајућим бројем рунде i , дате су у табели 3.4 зависно од ширине криптографског кључа.

Табела 3.4: *FEA-2* константе рунде за различите дужине криптографског кључа

i	LEN(K) = 128	LEN(K) = 192	LEN(K) = 256
1	c9e3b39803f2f6af	a4198d55053b7cb5	93c7673007e5ed5e
2	40f343267298b62d	be1442d9b7e08df0	81e6864ce5316c5b
3	8a0d175b8baafa2b	3d97eeea5149358c	141a2eb71755f457
4	e7b876206debac98	aa9782d20cc69850	cf70ec40dbd75930
5	559552fb4afa1b10	5071f733039a8ed5	ab2aa5f695f43621
6	ed2eae35c1382144	625c15071ea7bca1	da5d5c6b82704288
7	27573b291169b825	cf37d8f11024c664	4eae765222d3704a
8	3e96ca16224ae8c5	86d094e21e74d0a5	7d2d942c4495d18a
9	1acbda11317c387e	47df6e91fc91754b	3597b42262f870fd
10		1f0b2f23b88200e7	73d53787626cc076
11		29816e82b43e6464	4adf41d8ecafee96
12			e59d0f633aca9195

Трећи диференцијатор алгоритма типа *type2* односно *FEA-2*, поред броја рунди и вредности константи рунде, јесте параметар модификације и његова експанзија. Параметар модификације има фиксну ширину од 128 бита и дели се на два дела T_L и T_R ширине 64 бита на основу којих се зависно од броја рунде i формира параметар модификације рунде. Процедура експанзије параметра модификације, чија је дефиниција дата у виду алгоритма 3.9, прихвата параметар модификације T и дужине половина отвореног текста n_1 и n_2 .

Алгоритам 3.9 *FEA-2* експанзија параметра модификације

```
1: procedure FEA-TWEAKSCHEDULE( $T, n_1, n_2$ )
2:    $(T_L, T_R) \leftarrow (T[0..63], T[64..127])$ 
3:   for  $i \leftarrow 1, r$  do
4:     if  $i \bmod 3 = 0$  then
5:        $RT^i \leftarrow T_R$ 
6:     else if  $i \bmod 3 = 1$  then
7:        $RT^i \leftarrow 0^{64}$ 
8:     else
9:        $RT^i \leftarrow T_L$ 
10:    end if
11:  end for
12:  return  $(RT^1, RT^2, \dots, RT^r)$ 
13: end procedure
```

3.3 Одабир алгоритма

Најважнија компонента *LISPP* система је управо *FPE* алгоритам. Процес одабира конкретног *FPE* алгоритма ограничен је једино максималном подржаном величином отвореног текста, која мора бити довољно велика за шифровање свих делова заглавља пакета од интереса. По питању осталих критеријума нема формалних ограничења, тако да постоји слобода приликом избора *FPE* алгоритма. За потребе имплементације *LISPP* система одабран је *FF3-1* алгоритам из неколико разлога наведених у наставку. Кредитилитет који *NIST* ужива на пољу криптографије имао је изванредан допринос приликом избора *FPE* алгоритма. *FF3-1* алгоритам има мањи број рунди и генерално мањи број операција *AES* шифровања по свакој рунди, у поређењу са *FF1* алгоритмом такође препорученим од стране *NIST*, због чега је оправдано очекивати да има боље перформансе што га у домену обраде пакета чини погоднијим за постизање пропусног опсега на нивоу брзине везе.

FF3-1 алгоритам се може користити са произвољним алфабетом односно скупом карактера. Имајући у виду да *LISPP* систем врши обраду пакета, од значаја су искључиво бинарне вредности што донекле може олакшати имплементацију *FF3-1* алгоритма. У случају алфавета чија је основа једнака броју 2, што је случај за алфабет бинарних вредности, следеће базичне функције *FF3-1* алгоритма су поједностављене:

- $NUM(X)$, за задати низ бита X , јесте број који X представља, када се бити вреднују у редоследу опадајуће тежине, своди се на X .
- $NUM_{radix}(X)$, за задати низ нумерала X и основу $radix$, јесте број који X представља у основи $radix$ када се нумерали вреднују у редоследу опадајуће тежине, једнак је вредности X .
- $STR_{radix}^m(X)$, за задату основу $radix$, број нумерала m и ненегативни цео број x мањи од $radix^m$, јесте представа x као низа нумерала дужине m у основи $radix$ када се нумерали вреднују у редоследу опадајуће тежине, представља X на ширини од m бита.
- Операција израчунавања остатка при целобројном дељењу $X \bmod radix^m$ лако се имплементира операцијом битског маскирања $X \& ((1 \ll m) - 1)$.

Спецификације *FPE* алгоритама, иако релативно младе у поређењу са добро познатим симетричним криптографским алгоритмима блоковских шифара, већ су привукле велику

пажњу криптоаналитичара. Криптоанализа стандардизованих *FPE* алгоритама показала је да су напади на *FPE* алгоритме коришћењем диференцијалног разликовања (енгл. *differential distinguishers*) сложенији и захтевају више података при нападу на алгоритме из *NIST* препоруке него на алгоритме из *KATS* стандарда [83]. Стручњаци су уочили потенцијалне рањивости *FPE* шема, посебно указујући на смањење комплексности напада, нарочито када је дужина отвореног текста мала и у специфичним околностима, као што су оне у којима нападач зна вредност параметра модификације [82, 84]. У контексту *LISPP* система, ови проблеми су суштински спречени, јер дужине коришћених отворених текстова превазилазе оне за које постоје такве рањивости, а параметар модификације никада не напушта мрежни уређај, чиме је његова тајност загарантована. Према томе, напади описани у литератури нису примењиви на *LISPP*. Ипак, откривање оваквих проблема у тренутним алгоритмима требало би само да побољша њихове будуће верзије, док употреба *FPE* алгоритама у општем случају не би требало да буде угрожена. Оваквој констатацији иде у прилог чињеница да се активно појављују предлози нових *FPE* алгоритама, као што су *SPF* [85], *FAST* [86] и *RFPE* [87], који настоје да реше уочене проблеме и унапреде перформансе.

4. Архитектура *LISPP* система

У секцији 2.4 је изложена анализа постојећих решења за заштиту приватности на мрежном слоју, међу којима се посебно истичу *AHP*, *SPINE* и *PINOT*, засновани на идеји замагљивања *IP* адреса. Иако сваки од њих успешно скрива идентитет корисника од посматрача на мрежи у одређеним условима, ова решења пате од значајних функционалних ограничења која отежавају њихову широку примену. *AHP* систем захтева чување стања на мрежним уређајима ради одржавања мапирања између оригиналних и замагљених података, што директно утиче на скалабилност система и захтева значајне меморијске ресурсе. Са друге стране, *SPINE* и *PINOT* заобилазе проблем чувања стања ослањањем на већи адресни простор *IPv6* протокола за складиштење шифроване *IPv4* адресе, чиме постају некомпатибилни са изворним *IPv6* саобраћајем и зависе од доступности *IPv6* повезаности. Управо ови изазови представљају главну мотивацију за пројектовање система *LISPP*, који користи шифровање са очувањем формата како би обезбедио заштиту приватности без чувања стања, уз подршку за обе верзије *IP* адреса и очување изворне структуре заглавља пакета. *LISPP* је пројектован имајући у виду следеће особине:

- **Транспарентност.** Корисници из заштићене мреже не морају покретати никакву наменску апликацију. Они нису свесни да на путањи пакета постоји било какав систем за заштиту приватности (осим додатног минималног кашњења узрокованог обрадом пакета).
- **Стање се не чува током рада.** Мрежни уређај уопште не чува никакво стање; није потребно чувати мапирања између вредности отвореног и шифрованог текста нити било какве табеле. Рад без чувања стања додатно олакшава повезивање на већи број мрежа, јер нема потребе за синхронизацијом стања између свих улазних/излазних тачака.
- **Повезивање на већи број мрежа.** Претпоставимо да заштићена мрежа има више улазних/излазних тачака и да путање пакета нису симетричне у смеру уласка и изласка. У том случају, *LISPP* треба применити на свим улазним/излазним тачкама, при чему је неопходно разменити само криптографски материјал (криптографске кључеве и параметре модификације, као што је описано у секцији 3.1) између улазних/излазних тачака. Примена *LISPP* система на свим улазним/излазним тачкама лако се постиже коришћењем било ког механизма за размену кључева или постојећих криптографских канала између крајњих тачака као што је на пример *IPsec*.
- **Једноставна промена подешавања.** *LISPP* се може конфигурисати да штити било који порт *TCP* или *UDP* протокола. Потребно је само дефинисати одговарајући филтер за упаривање како би се одабрали пакети над којима ће се извршити замагљивање поља заглавља пакета.
- **Независност од протокола.** *LISPP* истовремено ради и са *IPv4* и са *IPv6* без потребе за било каквим изменама протокола мрежног слоја.

- **Правна усаглашеност.** Оператори мрежа заштићених помоћу *LISPP* система лако могу реконструисати право порекло пакета у случају легитимног захтева законских органа.

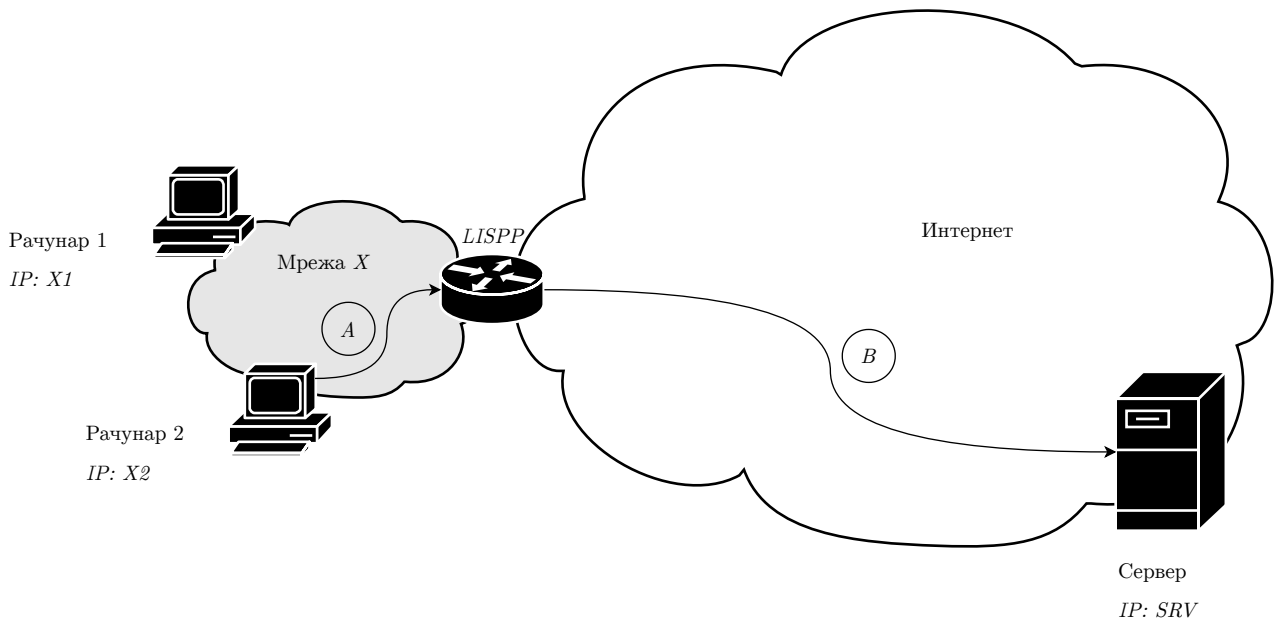
4.1 Принцип функционисања

IP адреса се састоји од два нераздвојна дела: мрежног дела, који дефинише локацију уређаја на интернету (аутономни систем), и хост дела, који идентификује тачног пошиљаоца или примаоца пакета у тој мрежи. Пошто је информација о локацији потребна за правилно прослеђивање пакета, *IP* адресе се обично шаљу незаштићене односно непромењене. Неки од постојећих механизма за заштиту примењују или замену адреса уз њихово чување као интерно стање, као што је случај код пресликавања мрежне адресе (енгл. *network address translation, NAT*), или потпуно шифровање пакета као код *Tor* или *IPsec*. Међутим, такви приступи нису увек скалабилни за опште обрасце коришћења интернета.

LISPP обрађује пакете на граници односно ивици мреже. Шифрује хост део изворишне *IP* адресе и изворишни порт¹ у пакетима који излазе из заштићене мреже, а дешифрује их у супротном смеру. У смеру изласка пакета из заштићене мреже, хост део оригиналне изворишне *IP* адресе и изворишни порт замењују се њиховим шифрованим вредностима (означени као *C1* и *C2* — шифровани текст на слици 4.1). Мрежни део *IP* адресе остаје непромењен, чиме се обезбеђује правилно прослеђивање пакета назад ка заштићеној мрежи. У смеру уласка пакета у заштићену мрежу, врши се дешифровање истим кључем, враћајући адресе и портове на њихове оригиналне вредности.

На овај начин, када корисник из заштићене мреже комуницира са спољним уређајима, спољни уређаји могу знати само локацију корисника односно мрежни део његове *IP* адресе, али не и тачну односно читаву оригиналну изворишну *IP* адресу корисника. Избор вредности изворишног порта из опсега привремених (енгл. *ephemeral*) изворишних портова на псеудослучајан начин је препоручена пракса још од 2011. године [88, 89]. Захваљујући томе, за један исти хост део изворишне *IP* адресе, шифровани текст би требало да буде другачији у свакој сесији, због велике вероватноће да изворишни порт добије нову вредност у свакој наредној *TCP* или *UDP* сесији. Сваки пут када корисник из заштићене мреже приступи истом спољном серверу, корисник ће изгледати као да има другачију *IP* адресу са великом вероватноћом, што се обезбеђује употребом криптографских алгоритама. Овакво понашање спречава одредиште или посматрача на било којој тачки између заштићене мреже и одредишта да прати понашање било ког специфичног корисника у заштићеној мрежи на мрежном нивоу, јер ће његове мрежне сесије изгледати као да долазе од различитих *IP* адреса. Понашање *LISPP* система је слично пресликавању порт адреса (енгл. *port address translation, PAT*), јер на улазној/излазној тачки мреже мења изворишну *IP* адресу и изворишни порт. Међутим, за разлику од *PAT*, који мапира скуп приватних *IP* адреса на једну или мањи број јавних *IP* адреса, *LISPP* врши бијекцију скупа јавних *IP* адреса на исти тај скуп *IP* адреса. Такође, за разлику од *PAT*, *LISPP* не чува стање, што значи да се мапирања између отвореног и шифрованог текста за хост део *IP* адреса и портова не морају чувати на мрежном уређају, јер се мапирање реализује коришћењем шифровања односно дешифровања. Из овог кратког описа је јасно да *LISPP* не тежи да замени нити да буде алтернатива *Tor* систему, јер представља једну тачку која замагљује изворишне адресе и портове. Ипак, постоји неколико јасних случајева употребе, као што је описано у наставку овог поглавља, у којима *LISPP* може заштитити приватност корисника.

¹Теоретски је изводљиво шифровати и додатна поља заглавља која не утичу на прослеђивање пакета, попут идентификатора трансакције код *DNS* упита, у циљу повећања ентропије улаза. При томе је неопходно осигурати да се не прекорачи максимална дозвољена дужина улаза *FPE* алгоритама.

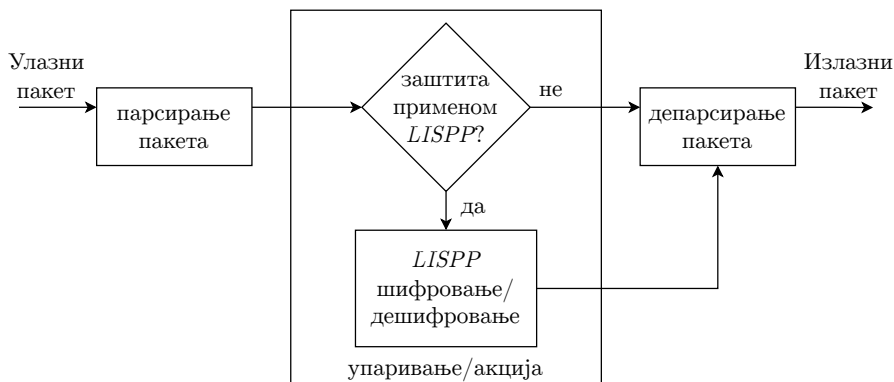


	Src IP		Dst IP		Src Port		Dst Port	
A	X	2	SRV		1025		443	
B	X	C1	SRV		C2		443	

Слика 4.1: Модификација поља заглавља пакета на ивици мреже применом LISPP система

4.2 Обрада пакета

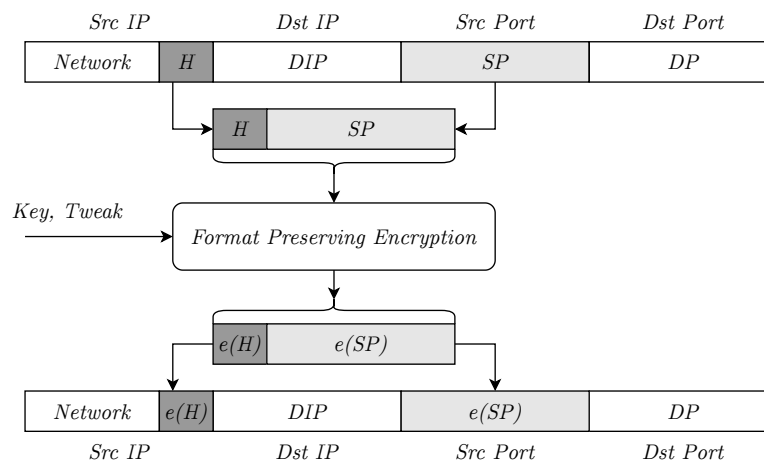
У оба смера, након парсирања пакета и провере контролне суме (енгл. *checksum*), LISPP филтрира пакете који ће бити обрађени што је приказано на слици 4.2. LISPP се може користити за заштиту било ког TCP или UDP пакета, јер за формирање отвореног текста користи изворишни порт у смеру изласка пакета из заштићене мреже. Међутим, могуће је програмирати филтер за упаривање (енгл. *match filter*) тако да у корак за шифровање/дешифровање шаље искључиво пакете са неком задатом вредношћу порта, док сви остали пакети пролазе кроз систем непромењени. Према томе, могуће је циљати само одређени подскуп саобраћаја од интереса као што је, на пример, TCP 443 за веб сигурност (енгл. *transport layer security, TLS*) или UDP 53 за DNS упите и одговоре.



Слика 4.2: Дијаграм LISPP обраде пакета

4.3 Шифровање поља заглавља пакета

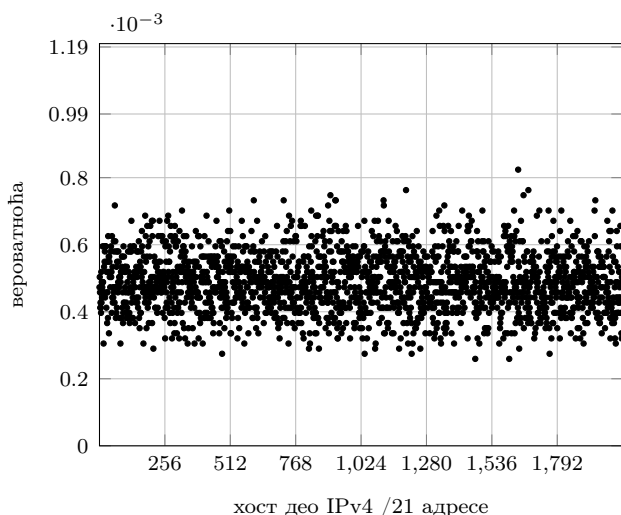
Слика 4.3 приказује како *LISPP* систем шифрује поља заглавља пакета користећи *FPE*. Хост део изворишне *IP* адресе и изворишни порт се надовезују и затим шифрују користећи тајни кључ и параметар модификације. Пошто се користи *FPE*, шифровањем n бита отвореног текста добија се тачно n бита шифрованог текста, без обзира колико бита n заиста представља. Захваљујући томе, без обзира на величину мрежне маске и верзију *IP* протокола, могуће је добити реверзибилно односно инјективно 1-1 пресликавање између парова (изворишна *IP*, изворишни порт) и ($e(\text{изворишна } IP)$, $e(\text{изворишни порт})$), где је са $e()$ означена операција шифровања. На тај начин могуће је постићи у потпуности транспарентан рад, без чувања стања, у оба смера.



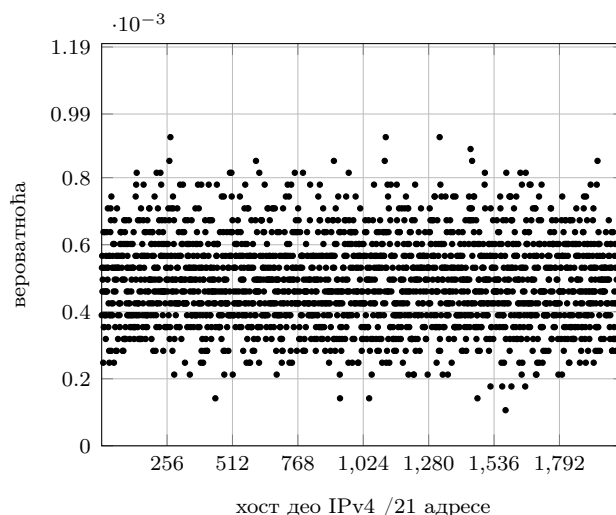
Слика 4.3: Шифровање изворишне адресе и изворишног порта применом *LISPP* система

Илустрација ефекта замагљивања адреса помоћу *LISPP* дата је на слици 4.4. Ова слика приказује емпиријску вероватноћу појављивања различитих вредности шифрованог текста за хост део изворишне *IP* адресе, добијених за једну исту изворишну *IP* адресу са маском /21 уз варирање изворишног порта у посматраном опсегу, све за исту вредност криптографског кључа и параметра модификације. Ово илуструје пример када би корисник у институцији попут Електротехничког факултета Универзитета у Београду, чији главни адресни опсег 147.91.8.0/21 има маску управо те дужине, користио *LISPP* ради замагљивања адреса. Слика 4.4а односи се на опсег вредности свих могућих изворишних портова од 0 до 65 535, док се слика 4.4б односи на опсег вредности привремених изворишних портова од 32 768 до 60 999 коришћених од стране *Linux* оперативног система. Визуелна инспекција показује да *LISPP* постиже равномерну расподелу шифрованих изворишних *IP* адреса преко свих могућих 2048 вредности 11-битног хост дела *IP* адресе. Овим је потврђена полазна хипотеза да се *FPE* алгоритми могу ефикасно користити за заштиту приватности на мрежном слоју и спречавање профилисања корисника.

FPE алгоритми захтевају тајни криптографски кључ и параметар модификације, описан у поглављу 3, за шифровање и дешифровање поља заглавља. Овај криптографски материјал може се генерисати директно на мрежном уређају коришћењем псеудослучајне вредности, изведене из семена (енгл. *seed*) задатог од стране корисника или преузете из неког извора псеудослучајних вредности на самом мрежном уређају. У случају постојања само једне улазне тачке у заштићену мрежу, криптографски кључ и параметар модификације не морају напуштати тај мрежни уређај, јер се и шифровање и дешифровање обављају на истом том уређају. Међутим, када заштићена мрежа има више улазних тачака и асиметричне улазне и излазне путање пакета, сви уређаји на граници мреже морају користити исти криптографски материјал. У том случају, један гранични мрежни уређај генерише криптографски кључ и



(а) сви изворишни портови



(б) привремени изворишни портови

Слика 4.4: Емпиријска вероватноћа појављивања различитих вредности хост дела *IP* адресе коришћењем *LISPP* система за само једну *IP* адресу са маском /21 и за опсег вредности изворишног порта

параметар модификације, док их сви остали мрежни уређаји добијају преко неке безбедне конекције, на пример *TLS* или *IPsec*. У било ком од ових случајева, параметар модификације се не шаље заједно са шифрованим пољима заглавља, што додатно ојачава безбедност предложеног решења.

Пошто криптографски материјал има ограничен радни век, криптографски кључ и параметар модификације морају се периодично мењати, на пример дневно, или након обраде одређеног броја пакета. У тренуцима када се кључ и/или параметар модификације мењају, мрежни токови који су активни у том периоду биће прекинати, јер су поља заглавља пакета у смеру изласка из заштићене мреже била шифрована старим кључем, док ће у смеру уласка у заштићену мрежу бити дешифрована новим кључем; ово ће резултовати погрешним *IP* адресама и бројевима портова за тај мрежни ток. Овакво понашање у прелазном периоду, иако краткотрајно, ремети рад мреже и мора се планирати у мирним периодима рада мреже. Реч је о компромису између вишег степена заштите приватности података, постигнутог кроз чешће промене криптографског кључа и параметра модификације, са једне стране и поузданог рада мреже са друге стране.

4.4 Модел претњи

LISPP представља механизам заштите приватности на мрежном слоју и као такав не штити приватност на апликативном нивоу. Као и *PINOT*, он прикрива *IP* адресе корисника и идентификаторе мрежних токова од сервера и посредних мрежа током приступа услугама на интернету. *LISPP* имплицитно намеће поверење у локалног мрежног оператора и не прикрива детаље конекција/токова од њега, нити од мрежних уређаја који врше шифровање. Независан посматрач између заштићене мреже и одредешта, као и само одредеште, могу открити стварну *IP* адресу корисника само прибављањем криптографског кључа или парова шифрованог и отвореног текста у дослуху са злонамерним мрежним оператором корисника, или „разбијањем” криптографског алгоритма, што је у тренутку писања ове докторске дисертације рачунарски тешко изводљиво.

4.5 Случајеви коришћења

LISPP систем је примењив и пожељан у свим случајевима у којима *ISP* додељује јавну *IP* адресу кориснику. Потенцијално, *ISP* може користити механизме попут доделе привремених *IPv6* адреса [90], без чувања стања, ради периодичне доделе и мењања привремених *IP* адреса. Међутим, према подразумеваним подешавањима овај период траје један дан, што нападачу даје довољно велики временски оквир за анализу понашања корисника и унакрсно повезивање овог понашања са другим изворима приватних података. Постоји доказ да је, упркос коришћењу таквих механизма, и даље нарушена приватност [8]. Захваљујући обезбеђеној насумичности *IP* адреса на нивоу сваке нове сесије, *LISPP* спречава могућност праћења корисника на мрежном слоју.

Једна јасна примена *LISPP* система је ублажавање претње од цурења приватних информација и профилисања корисника од стране јавних *DNS* разрешивача као што су *Google Public DNS* (8.8.8.8), *Quad9* (9.9.9.9), *Cloudflare* (1.1.1.1) и слични. *DNS* разрешивачи примају скуп симболичких назива домена веб страница, које корисник посећује, без обзира на форму транспорта тих назива домена до разрешивача: директно у отвореном тексту, шифровано применом *HTTPS* или *DNS* преко *TLS* (енгл. *DNS over TLS, DoT*). Стога, за корисника којем је додељена одређена *IP* адреса, *DNS* разрешивачи могу прикупљати информације о његовим интересовањима и посећеним веб страницама. Постоје покушаји да се називи домена сакрију од јавних *DNS* разрешивача шифровањем и енкапсулацијом у регуларне *DNS* упите, а затим преусмеравањем ка другом разрешивачу [91]. С друге стране, варирањем изворне *IP* адресе за сваки *DNS* упит корисника, *LISPP* успешно онемогућава такво профилисање, при чему је систем знатно једноставнији од постојећих решења. *ISP* може понудити *LISPP* као додатну услугу заштите приватности, која на мрежном слоју спречава треће стране (спољне веб странице и сервисе) на интернету да прате кориснике и анализирају њихово понашање.

Пример како *LISPP* замагљује извор *DNS* упита дат је на слици 4.5. Ова слика приказује осам узастопних *DNS* упита са једног истог рачунара у локалној мрежи са маском /21, као и скуп адреса и портова у које је *LISPP* променио оригиналне податке. Оператору *DNS* разрешивача је тешко, ако не и немогуће, да утврди са којим тачно уређајем заиста комуницира у било ком тренутку. Ефекат коришћења *LISPP* система био би исти за било који други мрежни протокол: *HTTPS*, *SSH*, *FTP*, итд.

<i>Src IP</i>	<i>Src Port</i>	<i>LISPP</i>	<i>LISPP Src Port</i>	<i>LISPP Src IP</i>
147.91.12.136	58119	↔	147.91.15.114	62299
147.91.12.136	44383	↔	147.91.10.233	6352
147.91.12.136	35450	↔	147.91.12.74	53732
147.91.12.136	58776	↔	147.91.10.72	4875
147.91.12.136	44346	↔	147.91.11.199	63754
147.91.12.136	43597	↔	147.91.12.122	13501
147.91.12.136	51655	↔	147.91.11.27	24254
147.91.12.136	56926	↔	147.91.9.118	44452

Слика 4.5: Шифровање 8 узастопних *DNS* упита који долазе од уређаја са *IP* адресом 147.91.12.136/21 применом *LISPP* система

Поред тога, због стално растућег броја сајбер-безбедносних претњи и потешкоћа у идентификовању нападача када напад долази иза операторског *NAT* (енгл. *carrier-grade NAT, CGNAT*) уређаја, недавно су се појавили подстицаји за прописивање обавезе чувања

метаподатака који омогућавају мапирање између корисника и *IP* адресе [92]. Уколико се такве регулације усвоје, *LISPP* их се може лако придржавати и истовремено сачувати приватност корисника у односу на треће стране. За разлику од великих логова *NAT* мапирања, у случају *LISPP* потребно је чувати само криптографски материјал, коришћен за *FPE* током његовог радног века, како би се могла реконструисати стварна *IP* адреса корисника на захтев законских органа.

Још један споредни ефекат коришћења *LISPP* система јесте значајно отежано скенирање портова (енгл. *port scan*) уређаја у заштићеној мрежи споља. Претпоставимо да нападач скенира читав опсег портова, са циљем проналажења отворених портова, за тачно једну одредишну адресу у заштићеној мрежи. У том случају, пакети за претраживање портова ће након дешифровања бити расути по читавом *IP* адресном сегменту, као што је приказано на слици 4.4, „погађајући” различите уређаје на портovima који у највећем броју случајева уопште нису они које је нападач циљао, чиме се анализа знатно отежава за спољашњег посматрача. Даље, честом променом криптографског кључа и/или параметра модификације, скенирани отисак (енгл. *footprint*) ће се у потпуности променити, чинећи анализу и нападе још тежим. *LISPP* се зато може сматрати и једним од алата и техника за стратегију померајуће мете (енгл. *moving target defence strategy*) [93].

Услед шифровања *LISPP* системом, слично као у случају отежаног скенирања портова, постоји изазов приступа неком серверу када се иницијатор сесије налази изван заштићене мреже. Поред могућности измештања сервера у демилитаризовану зону, што је данас уобичајена пракса, или организацијом адресирања и доделом посебног опсега адреса оним рачунарима којима је потребан приступ споља, приступ серверу који је унутар заштићене мреже инициран споља је могуће обезбедити дефинисањем статичког правила за филтрирање саобраћаја који *LISPP* треба да шифрује, што је илустровано на слици 4.2. Приступ датом серверу је омогућен јер ће *LISPP* пропуштати хост део његове *IP* адресе и порт без шифровања у оба смера: при уласку и изласку из заштићене мреже. Међутим, овим се нарушава бијективно пресликавање, које је иначе гарантовано алгоритмом шифровања, јер се хост део *IP* адресе и порт датог сервера пресликавају у исту вредност као регуларни шифровани текст за неки други пар хост дела *IP* адресе и порт из посматране заштићене мреже. Из тог разлога, *LISPP* ће тај други пар хост дела *IP* адресе и порт пресликати у шифровани текст у који би се иначе пресликао хост део *IP* адресе и порт датог сервера.

4.6 Промена криптографског материјала

Промену криптографског материјала би требало вршити када нема ниједне активне сесије, како не би дошло до њиховог прекида. У супротном неминовно долази до прекида активних сесија приликом промене криптографског материјала, јер се са новим криптографским кључем и параметром модификације мења и функција пресликавања изворишне *IP* адресе и порта, што значи да пакети текуће сесије више не би били исправно дешифровани при повратку. Поља заглавља пакета у смеру изласка из заштићене мреже су шифрована користећи стари криптографски материјал, док се у смеру уласка у заштићену мрежу дешифрују новим криптографским материјалом; ово ће резултовати погрешним *IP* адресама и бројевима портова за ту сесију. Стога, тренутак промене криптографског материјала представља изазов јер није једноставно детерминистички утврдити када тачно у систему нема ниједне активне сесије, што би био идеалан тренутак за ажурирање криптографског материјала. Овај проблем је усложњен чињеницом да број и трајање активних сесија зависе од специфичног случаја коришћења мреже и навика корисника, што отежава дефинисање универзалног правила за безбедну промену криптографског материјала. Дискусија по овом питању у случају *DNS* сервиса изложена је у оквиру секције 7.3.

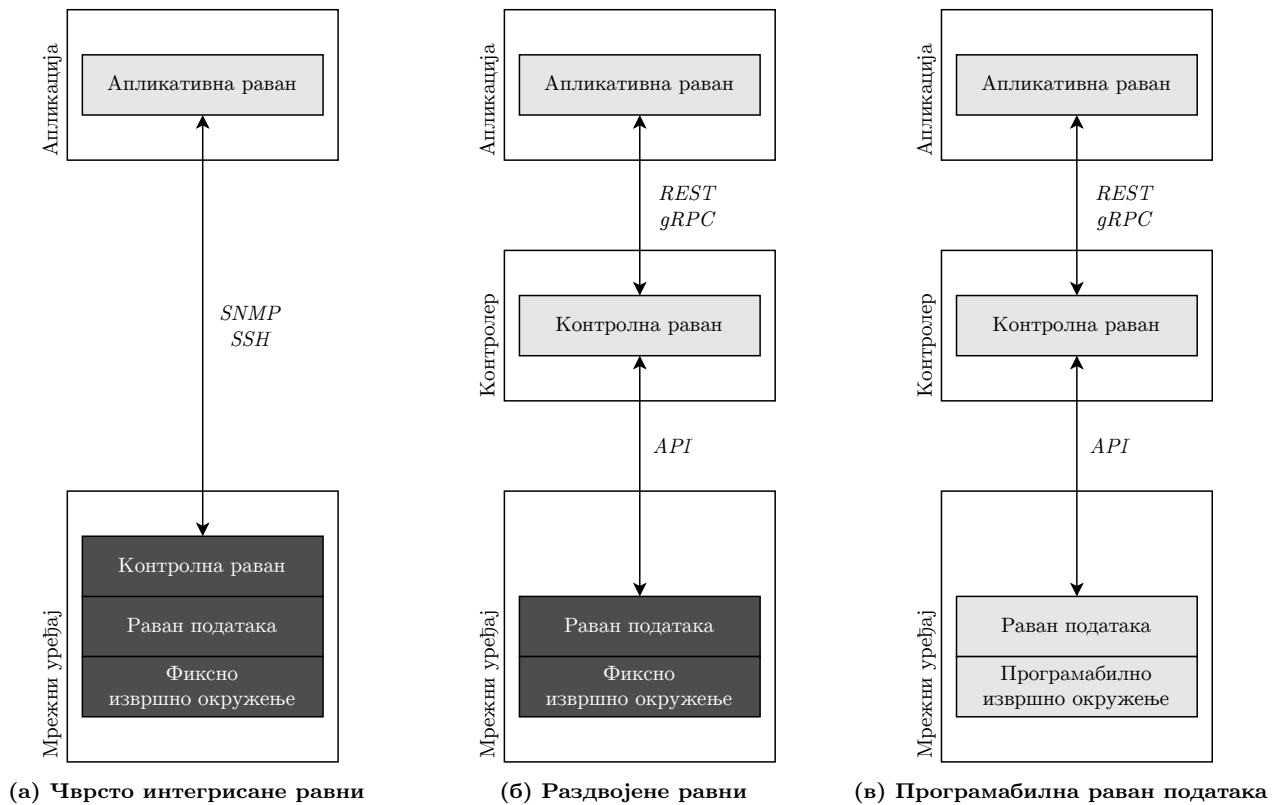
5. Имплементација *LISPP* система

Ово поглавље бави се практичном реализацијом *LISPP* система, представљајући различите приступе који омогућавају његову интеграцију у актуелну мрежну инфраструктуру. Основни предуслов за успешну имплементацију система попут *LISPP* је способност мрежног уређаја да ефикасно пресреће мрежни саобраћај и директно утиче на садржај пакета у реалном времену. У наставку су прво размотрене технологије које омогућавају поменути манипулацију пакетима кроз концепт програмабилних мрежа, након чега су детаљно описане две верзије имплементације *LISPP* система: хардверска, заснована на *Netronome SmartNIC* уређају и *P4* и *Micro-C* језику, која тежи максималним перформансама, и софтверска, заснована на *eBPF* технологији, погодна за флексибилну примену у виду виртуелизоване мрежне функције.

5.1 Модификација садржаја пакета

Традиционални мрежни уређаји обрађују пакете користећи алгоритме контролне равни (енгл. *control plane*) и равни података (енгл. *data plane*) што је илустровано на слици 5.1а. Програмер није у могућности да промени уграђене алгоритме контролне равни, већ може једино да конфигурише традиционални мрежни уређај и то искључиво кроз промену параметара уграђених алгоритама контролне равни. Приликом пројектовања *LISPP* система један од примарних циљева била је његова компатибилност са постојећим мрежама. Постизање овог циља олакшано је захваљујући развоју области софтверски дефинисаних мрежа (енгл. *Software-Defined Networks, SDN*) у последњих десетак година [94, 95]. Значај *SDN* огледа се у пруженој флексибилности по питању дефинисања начина обраде и прослеђивања пакета на мрежним уређајима.

За разлику од традиционалних мрежних уређаја, које карактерише ограничена конфигурабилност услед ослањања на наменске чипове са фиксно дефинисаним функцијама, *SDN* са друге стране подразумева постојање програмабилних мрежних уређаја чиме је омогућено управљање понашањем читаве мреже у погледу обраде и прослеђивања пакета. Једно од главних начела и првих корака у развоју *SDN* јесте раздвајање контролне равни и равни података видљиво на слици 5.1б, које су код традиционалних мрежних уређаја чврсто интегрисане [96]. Контролна раван је имплементирана у софтверу и најчешће је реализована кроз централизованог контролера. Задатак контролне равни у општем случају јесте одређивање путање пакета кроз мрежу користећи информације о топологији мреже, а на основу политике дефинисане од стране администратора дате мреже. Након одређивања путање пакета, контролна раван комуницира са равни података како би задала правила за прослеђивање пакета. Комуникација контролне равни са равни података се обавља кроз експлицитно дефинисани апликативни програмски интерфејс (енгл. *Application Programming Interface, API*) преко одговарајућег комуникационог протокола. Најпознатији протокол за комуникацију контролне равни са равни података је *OpenFlow* протокол [97, 98], чија је стандардизација у великој мери потпомогла реализацију *SDN* концепта. Раван података представља део мрежних уређаја задужен за прослеђивање пакета на основу правила дефи-



Слика 5.1: Еволуција мрежних уређаја

нисаних од стране контролне равни. Правила су углавном задата у формату појединачних улаза табеле прослеђивања или у општем случају табеле за упаривање и акције (енгл. *match-action tables*) датог уређаја.

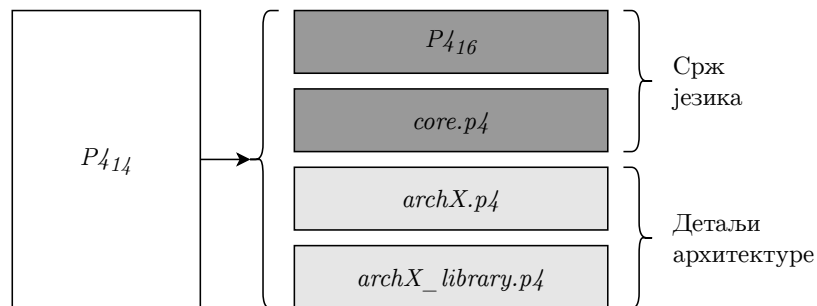
Поред могућности прослеђивања пакета искључиво на основу пуких правила, равни података је временом такође постала програмабилна сама по себи што је приказано на слици 5.1в, захваљујући чему може директно обављати и друге функције као што су модификација заглавља пакета, филтрирање пакета, формирање статистике мрежног саобраћаја итд. Програмабилност равни података омогућава дефинисање начина обраде и прослеђивања пакета на мрежним уређајима на скоро произвољан начин. Овим се избегава уоквиреност у коначан скуп операција предефинисаних протокола доступних код традиционалних мрежних уређаја. Брзина обраде и прослеђивања пакета притом остаје на нивоу брзине везе или приближно на нивоу брзине везе, јер равни података може функционисати у потпуности независно, елиминишући самим тим потребу за сталном комуникацијом између контролне равни и равни података. Ово омогућава лаку имплементацију нових протокола и алгоритама, као што је случај са *LISPP* системом. Управо ради постизања обраде пакета на нивоу брзине везе, *LISPP* систем је имплементиран у целости у оквиру равни података, што је изводљиво захваљујући програмабилности равни података.

Постоје два практична приступа програмирању равни података мрежних уређаја. Први приступ се заснива на коришћењу општенаменских програмских језика као што је *C* програмски језик, док други приступ подразумева употребу релативно младих доменски специфичних језика (енгл. *Domain-Specific Language, DSL*) као што је *P4* језик [99, 100]. Карактеристике ова два приступа су у великој мери супротне, при чему предности једног приступа ефективно осликавају недостатке другог и обрнуто. Општенаменски програмски језици имају развијен екосистем, што подразумева постојање опробаних ланаца алата за превођење (енгл. *toolchain*), зрелих радних окружења (енгл. *framework*), корисних алата за тестирање, као и велики број искусних програмера. Општенаменски програмски језици пружају и велику флексибилност у погледу начина обраде пакета, али зато захтевају детаљно

познавање архитектуре мрежног уређаја. Доменски специфични језици за програмирање равни података мрежних уређаја су наменски пројектовани, тако да одлично осликавају могућности циљне платформе и олакшавају посао програмеру захваљујући доступним конструктима језика прикладним управо за домен програмирања мрежних уређаја. Доменски специфични језици апстрахују циљну платформу до извесне мере, омогућавајући преносивост имплементације на већи број различитих уређаја исте архитектуре. У наставку докторске дисертације, кроз дискусију о имплементацији *LISPP* система, биће изложено више детаља о предностима и манама ова два приступа.

5.1.1 *P4* језик

Тренутно најраспрострањенији начин програмирања равни података мрежних уређаја јесте помоћу *P4* језика [101]. *P4* (*Programming Protocol-independent Packet Processors*) је доменски специфичан и хардверски независан језик за програмирање мрежних уређаја, помоћу којег програмер може да дефинише произвољан начин прослеђивања пакета на мрежним уређајима. Постоје две стандардизоване верзије језика, *P4₁₄* [102] и *P4₁₆* [103], које се у великој мери међусобно разликују што је приказано на слици 5.2 и описано детаљније у наставку. *P4₁₄* дијалект омогућава дефинисање алгоритама равни података помоћу добро познатих императивних конструката, али поред тога има и уграђене специјализоване декларативне конструкте за често коришћене функционалности равни података као што су бројачи, израчунавање контролне суме итд. Услед постојања наведених декларативних конструката, *P4₁₄* дијалект обухвата велики број кључних речи и има стрму криву учења (енгл. *steep learning curve*). *P4₁₄* дијалект имплицитно намеће једну *P4* архитектуру циљног уређаја и нема подршку за увођење дефиниција нових *P4* архитектура разноликих циљних уређаја. Поред свега тога, *P4₁₄* дијалект има још недостатака од којих су најбитнији слаба типизираност, мноштво конструката ниског нивоа апстракције и изостанак јасне подршке за модуле, што све резултује нелагодним искуством приликом писања програма.

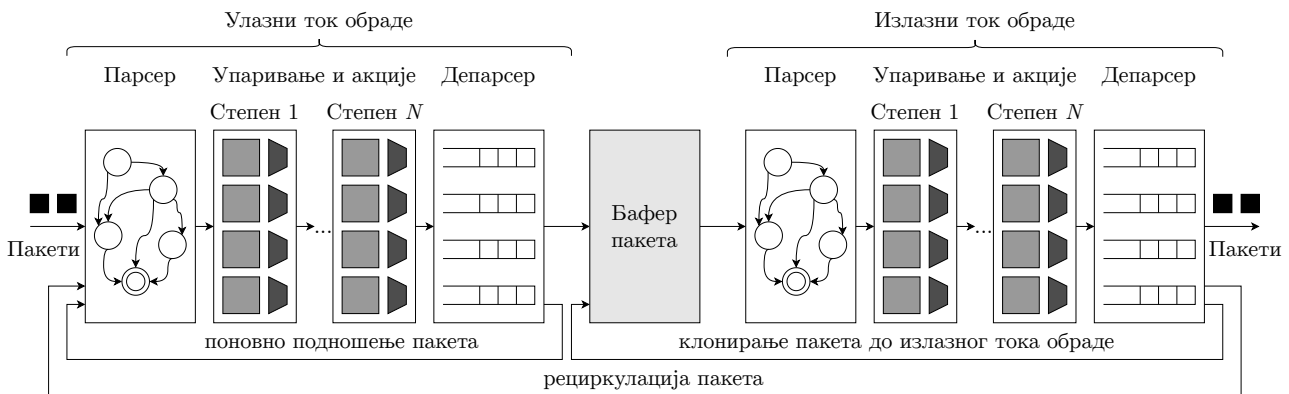


Слика 5.2: Разлика између *P4₁₄* и *P4₁₆* дијалеката

P4₁₆ дијалект је значајно измењен у односу на *P4₁₄* дијалект, увођењем великог броја повратно некомпатибилних измена, са циљем отклањања споменутих недостатака. Избачени су у потпуности специјализовани декларативни конструкти, наметнута је строга типизираност и уведено је неколико механизма за дефинисање модула. Највећи искорак *P4₁₆* дијалекта јесте подршка за увођење нових *P4* архитектура, што је постигнуто раздвајањем сржи језика и детаља *P4* архитектуре. Задатак *P4* архитектуре јесте премошћавање јаза између сржи језика и циљног уређаја, кроз дефинисање структуре односно навођење доступних ресурса и функционалности програмабилне равни података циљног уређаја, чинећи то искључиво на високом нивоу апстракције и без улажења у детаље конкретне архитектуре циљног уређаја. Структура *P4* архитектуре издвојена је у засебан *P4₁₆* опис, док су функције дате *P4* архитектуре доступне унутар пратеће *P4₁₆* библиотеке. Срж језика је даље подељена на мали скуп конструката језика и стандардну библиотеку коришћену од

стране већине програма. Због свих наведених предности, данас је у практичној употреби примарно P_{416} дијалект, док се P_{414} дијалект користи искључиво у застарелим пројектима. Управо због тога, у наставку дисертације биће разматран једино P_{416} дијалект, а свуда где се спомиње P_4 језик подразумева се управо P_{416} дијалект.

Архитектура преносивог свича (енгл. *Portable Switch Architecture, PSA*) [104, 105] дефинише структуру тока обраде пакета кроз већи број програмабилних блокова и компоненти фиксне намене, као што је илустровано на слици 5.3. Обрада пристиглих пакета почиње са програмабилним парсером, реализованим као коначни аутомат, чији је задатак препознавање, екстракција и валидација поља заглавља пакета. Програмабилност парсера се огледа у подршци за увођење произвољног броја стања у процес парсирања и могућности дефинисања заглавља пакета, како за добро познате, тако и за нестандартне протоколе. По окончању парсирања, пакет се прослеђује у *улазни ток обраде за упаривање и акције* (енгл. *ingress match-action pipeline*), где се кроз низ од N фаза врши обрада пакета заснована на експлицитно задатим правилима. Свака фаза у току обраде за упаривање и акције располаже меморијским ресурсима у виду већег броја табела за упаривање и процесорским ресурсима у виду већег броја једноставних аритметичко-логичких јединица (енгл. *Arithmetic Logic Units, ALU*) за извршавање акција. Улази наведених табела се попуњавају од стране контролне равни, при чему сваки улаз садржи кључ за упаривање, придружену акцију и додатне пратеће податке. Уколико се за кандидата, формираног на основу вредности изабраних поља заглавља и метаподатака пакета, пронађе иста вредност кључа за упаривање у неком улазу табеле, извршава се акција придружена управо том улазу табеле. Извршавање акције може за ефекат имати модификацију пакета, ажурирање стања чуваног унутар мрежног уређаја и усмеравање пакета ка жељеном одредишту. *PSA* се стога заснива на парадигми *упаривања и акција*, где се вредности изабраних поља заглавља и метаподатака пакета упарују са кључевима улаза табела, са циљем одабира одговарајуће придружене акције за извршавање. Следећи блок у низу приликом обраде пакета је улазни депарсер чија је улога реконструкција односно серијализација пакета.

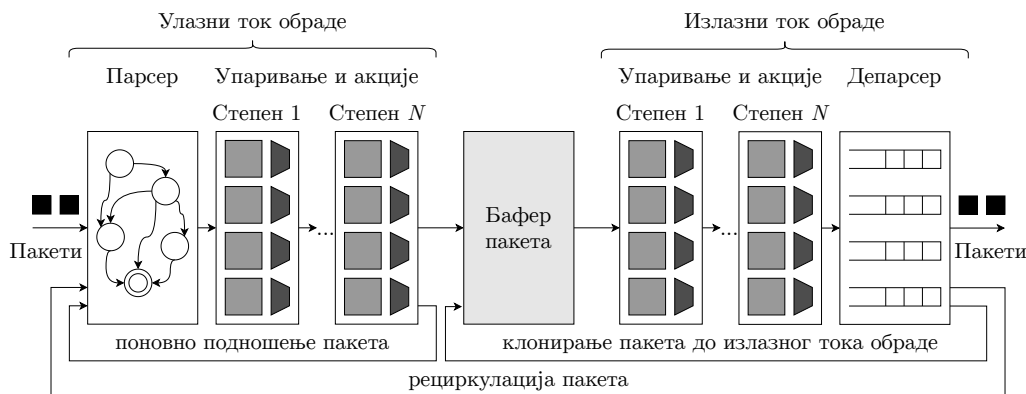


Слика 5.3: *PSA P4* архитектура

Претходно наведена три програмабилна блока, тачније улазни парсер, улазни ток обраде за упаривање и акције и улазни депарсер, заједно чине сегмент имена *улазни ток обраде* пакета. Након проласка кроз улазни ток обраде, пакет може бити поново поднет назад на почетак улазног тока обраде или прослеђен даље ка баферу пакета. Бафер пакета представља једину компоненту фиксне намене у оквиру *PSA* и задужен је за привремено складиштење и управљање редовима чекања. Напуштањем бафера, пакет улази у *излазни ток обраде* пакета сачињен од излазног парсера, излазног тока обраде за упаривање и акције и излазног депарсера, чије су функције аналогне њиховим парњацима из улазног дела. Након проласка кроз излазни ток обраде, пакет може бити усмерен ка одредишном интерфејсу, прослеђен централном процесору на даљу обраду, али може

бити подвргнут и операцији клонирања или рециркулације. Клонирање пакета омогућава креирање дупликата посматраног пакета уз слање дупликата на почетак излазног тока обраде, док рециркулација пакета подразумева враћање посматраног пакета на почетак улазног тока обраде ради додатног процесирања.

PSA пружа апстрактни модел чијим се варијацијама могу дефинисати различите конкретне *P4* архитектуре. Варијације се могу огледати у броју и врсти доступних блокова, присутности парсера и депарсера у току обраде или одабиру компоненти фиксне намене. *v1model* [106, 107] представља једну од варијација теоријског модела дефинисаног у оквиру *PSA*, сачињену од улазног парсера, улазног тока обраде за упаривање и акције, бафера пакета, излазног тока обраде за упаривање и акције и излазног депарсера, као што је приказано на слици 5.4.



Слика 5.4: *v1model P4* архитектура

v1model дефинише постојање компоненти фиксне намене за пријем односно слање пакета, смештених непосредно пре улазног парсера односно непосредно након излазног депарсера. Интеракција између програмабилних блокова и компоненти фиксне намене реализује се посредством механизма размене уграђених метаподатака (енгл. *intrinsic metadata*). Сваки пакет поред експлицитно дефинисаних метаподатака, чије је значење одређено од стране програмера, поседује и уграђене метаподатке са специфичним значењем у контексту појединачних компоненти фиксне намене. Уграђени метаподаци обухватају информације попут улазног и излазног интерфејса, величине пакета, временских тренутака итд. Програмабилни блокови могу идентификовати улазни интерфејс посматраног пакета читањем одговарајућег поља уграђених метаподатака које је претходно иницијализовано од стране компоненте фиксне намене за пријем пакета на адекватан начин. Слично, програмабилни блокови могу задати излазни интерфејс посматраног пакета уписом жељене вредности у одговарајуће поље уграђених метаподатака које компонента фиксне намене за слање пакета консултује приликом избора излазног интерфејса. *v1model* дефинише и постојање компоненти фиксне намене за израчунавање контролне суме, смештених непосредно након улазног парсера и непосредно пре излазног депарсера. Овим је омогућена верификација интегритета пакета при пријему и ажурирање контролне суме при слању, што је од кључног значаја за исправно функционисање мрежних протокола након модификације заглавља пакета у оквиру програмабилних блокова.

Дефинисање жељеног начина обраде и прослеђивања пакета, као и његово мапирање на програмабилне блокове одабране *P4* архитектуре, реализује се употребом конструката *P4* језика описаних у наставку и илустрованих листингом 5.1 на примеру *v1model* архитектуре. Парсер представља специфичан програмабилни блок *P4* архитектуре за чију дефиницију се користи `parser` конструкт односно парсер блок намењен за моделовање коначних аутомата. У оквиру парсер блока дефинишу се стања и правила транзиције између њих, која су условљена вредностима екстрахованих поља заглавља пакета. Извршавање парсера увек

започиње у стандардном експлицитно дефинисаном почетном стању `start`. У зависности од исхода парсирања, коначни аутомат завршава рад у једном од два стандардна имплицитно дефинисана завршна стања: `accept`, које означава успешно парсирање и прослеђивање пакета до наредног програмабилног блока, или `reject`, које представља грешку приликом парсирања и доводи до одбацавања пакета.

За опис начина обраде пакета у осталим програмабилним блоковима, као што су улазни ток обраде пакета, излазни ток обраде пакета и депарсер, користи се `control` конструкт односно контролни блок. У потпису контролног блока налази се листа улазно/излазних параметара неопходних зарад размене заглавља и метаподатака пакета између посматраног контролног блока и осталих делова *P4* програма. Контролни блок обавезно садржи тачно један `apply` блок за дефиницију алгоритма обраде пакета од стране посматраног програмабилног блока, а поред тога опционо може садржати и дефиниције ресурса као што су `table` табеле за упаривање или `action` акције за извршавање, или инстанцирање екстерних елемената у виду регистара, бројача, компоненти за израчунавање контролне суме и генерисање псеудослучајних бројева, итд. Унутар `apply` блока специфицира се ток извршавања (енгл. *control flow*) програма, укључујући редослед примене табела и условно извршавање акција. *P4* језик подржава стандардне механизме за контролу тока извршавања програма, као што су `if-else` условно гранање и `switch` вишеструко гранање, као и позиве функција и екстерних метода, операције за руковање подацима, укључујући конкатенацију, индексирање и одсецање ниски бита.

Битна карактеристика *P4* језика, која га разликује од већине општенаменских програмских језика, јесте одсуство подршке за петље и рекурзију. Ово ограничење је намерно уведено како би се гарантовало да се обрада сваког пакета завршава у коначном и предвидивом времену, што је неопходан услов за постизање детерминистичких перформанси и пропусног опсега на нивоу брзине везе. Слично `apply` блоку, акције такође дефинишу ток извршавања програма енкапсулирајући логику за модификацију садржаја пакета и метаподатака. Док `apply` блок оркестрира целокупан процес обраде пакета од стране једног програмабилног блока, акције дефинишу конкретне операције извршаване као резултат упаривања у табелама. Табеле за упаривање представљају основни механизам за доношење одлука у *P4* програмима. Дефиниција табеле специфицира њено име, структуру, скуп свих дозвољених акција, подразумевану акцију, спецификацију кључа за упаривање и врсту упаривања. Улази табеле могу бити задати у виду непроменљивих података у оквиру дефиниције табеле, али најчешће се улази табеле попуњавају од стране контролне равни. Приликом дефинисања табеле могу се навести и додатни атрибути као што су величина табела изражена у максималном броју складиштених улаза.

P4 језик је статички типизиран језик са подршком за разнолике типове података корисне у домену програмирања равни података. Од основних типова података доступне су логичке вредности `bool` и целобројни типови података задате фиксне величине у које спадају означени цели бројеви `int` и неозначени цели бројеви `bit` звани често и ниске бита. С обзиром да раван података мрежних уређаја често захтева рад са пољима заглавља и метаподатака чија величина није поравната на границе бајтова, за величину целобројних типова података подржана је грануларност до нивоа појединачног бита, тако да, на пример, могу бити дефинисане променљиве типа података `int<3>`, `bit<14>` и `bit<96>`. Уз ниске бита фиксне величине, доступне су и ниске бита променљиве величине `varbit`, што омогућава дефинисање променљивих чија је величина одређена тек приликом извршавања *P4* програма.

Поред основних типова података, подржани су и изведени (енгл. *derived*) типови података. Набрајање (енгл. *enumeration*) је тривијални представник изведених типова података услед могућности експлицитног навођења интерне представе за елементе набрајања. Осим набрајања, *P4* језик подржава и изведене типове као што су торке (енгл. *tuples*), структуре

```

1 #include <core.p4>
2 #include <v1model.p4>
3
4 struct MyHeaders_t {
5 }
6
7 struct MyMetadata_t {
8 }
9
10 parser MyParser(
11     packet_in packet,
12     out MyHeaders_t hdrs,
13     inout MyMetadata_t meta,
14     inout standard_metadata_t standard_metadata)
15 {
16     state start { /* packet.extract() ... */ transition my_state_1; }
17     state my_state_1 { transition my_state_2; }
18     state my_state_2 { transition accept; }
19 }
20
21 control MyVerifyChecksum(
22     inout MyHeaders_t hdrs,
23     inout MyMetadata_t meta)
24 { apply { /* ... */ } }
25
26 control MyIngress(
27     inout MyHeaders_t hdrs,
28     inout MyMetadata_t meta,
29     inout standard_metadata_t standard_metadata)
30 { apply { /* action a1() {}; table t1{}; ... */ } }
31
32 control MyEgress(
33     inout MyHeaders_t hdrs,
34     inout MyMetadata_t meta,
35     inout standard_metadata_t standard_metadata)
36 { apply { /* action a2() {}; table t2{}; ... */ } }
37
38 control MyComputeChecksum(
39     inout MyHeaders_t hdrs,
40     inout MyMetadata_t meta)
41 { apply { /* update_checksum(); ... */ } }
42
43 control MyDeparser(
44     packet_out packet,
45     in MyHeaders_t hdrs)
46 { apply { /* packet.emit(); ... */ } }
47
48 V1Switch(
49     MyParser(), MyVerifyChecksum(), MyIngress(),
50     MyEgress(), MyComputeChecksum(), MyDeparser()
51 ) main;

```

Листинг 5.1: Структура *P4* програма за *v1model* архитектуру

(енгл. *structs*), заглавља (енгл. *headers*) и стекови заглавља (енгл. *header stacks*). Торка омогућава груписање већег броја елемената произвољних типова података у једну логичку целину. Елементима торке приступа се индексирањем помоћу позиције жељеног елемента унутар саме торке. Структура омогућава груписање већег броја чланова произвољних типова података у једну логичку целину. Члановима структуре приступа се преко имена жељеног члана додељеног у склопу дефиниције структуре. Значај торки и структура долази до изражаја приликом дефинисања сложенијих типова података као што су метаподаци пакета. Заглавље се користи за представљање заглавља различитих протокола, због чега може садржати искључиво поља која директно или индиректно могу бити серијализована у целини, а најчешће су то ниске бита фиксне величине `bit` и ниске бита променљиве величине `varbit`. Заглавље увек поседује и имплицитно логичко поље валидности чија вредност указује да ли је посматрано заглавље присутно унутар посматраног пакета. Поље валидности унутар заглавља поставља парсер приликом екстракције заглавља, чита депарсер зарад доношења одлуке да ли посматрано заглавље треба уградити у реконструисани пакет, али вредношћу поља валидности може се експлицитно руковати путем функција `isValid`, `setValid` и `setInvalid`. *P4* језик не пружа ниједну дефиницију заглавља, чак ни за добро познате протоколе, већ свако неопходно заглавље мора бити експлицитно дефинисано. Стек заглавља омогућава дефинисање секвенце заглавља истог типа, што је корисно приликом рада са протоколима који дозвољавају уградњу више понављајућих заглавља истог типа, као што је случај са мултипротоколским комутирањем на основу ознака (енгл. *multiprotocol label switching, MPLS*).

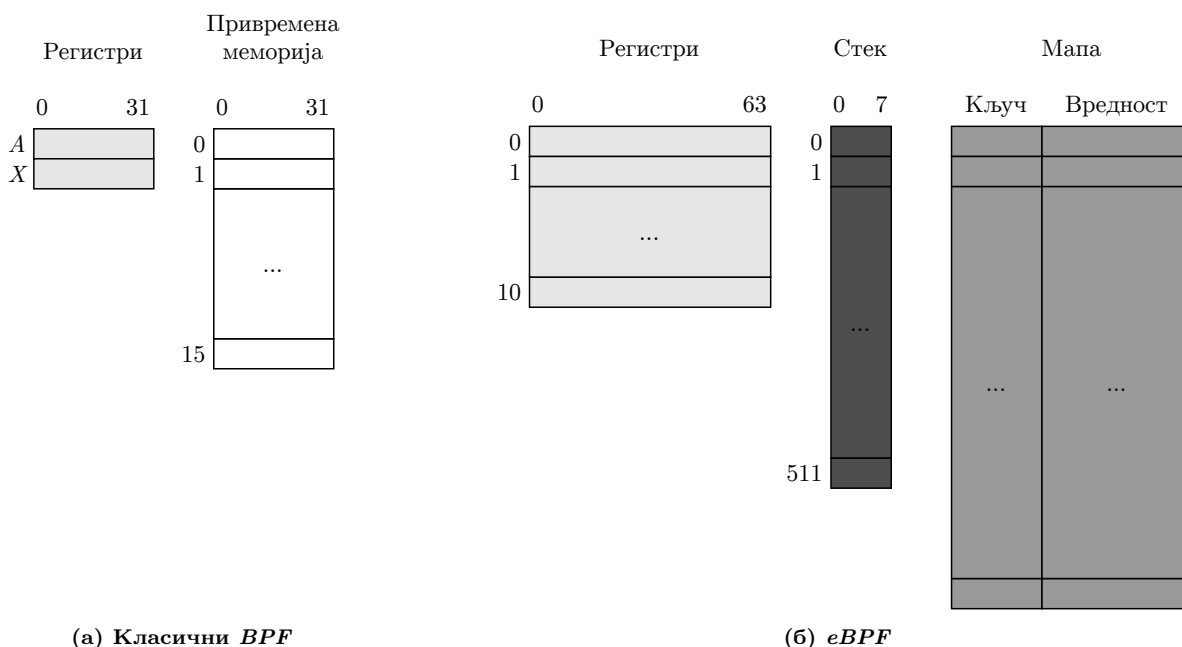
P4 програм, написан за неку *P4* архитектуру, дефинише алгоритам извршаван од стране програмабилних блокова, као и њихову интеракцију са компонентама фиксне намене. Симбиоза *P4* програма и компоненти фиксне намене чини целовиту имплементацију равни података за циљни уређај. *P4* преводилац, обезбеђен од стране произвођача циљног уређаја, преводи *P4* програм у извршни код прилагођен свим могућим специфичностима конкретне архитектуре посматраног циљног уређаја. Како би се избегла нежељена ситуација да сваки произвођач циљног уређаја мора имплементирати *P4* преводилац од нуле, *P4* преводилац се састоји из два дела: *P4 front-end* преводилац и *P4 back-end* преводилац. *P4 front-end* преводилац је независан од циљне архитектуре, а задужен је за анализу и проверу исправности *P4* програма, као и за генерисање *P4* међурепрезентације (енгл. *intermediate representation, IR*). *P4 back-end* преводилац је специфичан и уско везан за циљни уређај, а задужен је за превођење *P4* међурепрезентације у извршни код прилагођен конкретној архитектури циљног уређаја. Овакав приступ омогућава произвођачима циљних уређаја да се фокусирају искључиво на развој *P4 back-end* преводиоца, док је *P4 front-end* преводилац у великој мери заједнички за све циљне уређаје. Захваљујући овоме, релативно лако се постиже преносивост *P4* програма међу свим циљним уређајима са подршком за исту *P4* архитектуру за коју је посматрани програм написан.

5.1.2 *eBPF* технологија

eBPF вуче порекло од Берклијевог филтера пакета (енгл. *Berkeley Packet Filter, BPF*) [108]. Једина функција класичног *BPF* је иницијално била ефикасно филтрирање пакета унутар језгра оперативног система, са циљем смањења броја пакета копираних из простора језгра у кориснички простор приликом надгледања мрежног саобраћаја. Уместо безусловног копирања сваког примљеног пакета из простора језгра у кориснички простор, класични *BPF* омогућава извршавање једноставних класичних *BPF* програма унутар језгра ради што ранијег одбацивања нежељених пакета. Класични *BPF* представља виртуелну машину специфичне намене са релативно једноставном архитектуром коју чине 32-битни акумулатор (*A*), 32-битни индексни регистар (*X*) и имплицитни програмски бројач (енгл. *program*

counter). Меморијски модел класичне *BPF* виртуелне машине базиран је на процесу филтрирања пакета што имплицира да инструкције за рад са подацима подразумевано приступају садржају тренутно посматраног пакета, али поред тога постоји још и помоћна привремена меморија (енгл. *scratch memory*) величине свега 64 бајта [109]. Инструкцијски сет класичне *BPF* виртуелне машине је довољно генерализован како би се могла изразити произвољна логика филтрирања пакета, док су појединачне инструкције једноставне таман толико да се њихово декодовање може обавити помоћу само једне наредбе вишеструког гранања што убрзава њихову интерпретацију од стране језгра.

Током година употребе, долази до проширења скупа функционалности и последично постепеног повећања домена примене, услед чега класична *BPF* виртуелна машина специфичне намене полако прераста у општенаменску *eBPF* виртуелну машину [110]. Најзначајније проширење функционалности јесте подршка за извршавање ширег спектра *eBPF* програма, који више не морају обављати искључиво филтрирање пакета на граници између простора језгра и корисничког простора, већ могу обављати произвољне задатке као вид реакције на појаву неког од многобројних догађаја унутар језгра оперативног система [111]. Додавање опције позивања помоћних функција језгра (енгл. *helper functions*) у великој мери је отворило врата нових могућности и олакшало писање *eBPF* програма, док је увођење апликативног програмског интерфејса *eBPF* мапа, за коришћење делова меморије језгра флексибилне величине, омогућило интеракцију програма из корисничког простора са *eBPF* програмом извршаваним у језгру [112]. Имплементација превођења у лету (енгл. *Just-In-Time, JIT*) на машинске инструкције процесора домаћина уместо интерпретације је осетно побољшала перформансе *BPF* виртуелне машине [113]. Поред тога, број програмски доступних регистара је повећан са два на једанаест (*R0-R10*), регистри су постали 64-битни, уведен је стек подразумеване величине 512 бајтова чиме је омогућено позивање функција, што је све илустровано на слици 5.5, а додате су и инструкције сличније машинским инструкцијама процесора домаћина.



Слика 5.5: Архитектуре класичне *BPF* и *eBPF* виртуелне машине

Инструкцијски сет (енгл. *instruction set architecture, ISA*) *eBPF* виртуелне машине [114] је релативно ограничен, без подршке за неке од уобичајених ресурса процесора домаћина као што су инструкције за рад са реалним бројевима, инструкције за рад са више података (енгл. *single instruction multiple data, SIMD*), периферије за криптографска израчунавања, итд.

Услед тога, *eBPF* програми не могу директно користити све ресурсе процесора домаћина, иако се извршавају унутар језгра. Директна последица овога јесте да, у неким специфичним ситуацијама, извршавање *eBPF* програма може бити спорије у односу на извршавање програма из корисничког простора који могу у потпуности искористити ресурсе процесора домаћина.

Извршавање *eBPF* програма под *eBPF* виртуелном машином постиже се типично кроз неколико корака описаних у наставку. *eBPF* програм прво мора бити учитан у меморију језгра оперативног система, што иницира програм из корисничког простора путем мулти-плексираног `bpf` системског позива, наводећи *eBPF* бајт код и аргумент `BPF_PROG_LOAD` за жељену команду. Задати *eBPF* програм не читава се у меморију језгра безусловно, већ се прво анализирају његова безбедносна својства коришћењем верификатора (енгл. *verifier*) [115, 116]. Уколико је *eBPF* програм безбедан, врши се његово превођење у лету на машинске инструкције процесора домаћина, након чега линкер врши повезивање са помоћним функцијама језгра, да би се коначно такав *eBPF* програм прочитао у меморију језгра. Затим програм из корисничког простора мора претходно учитани *eBPF* програм закачити на одговарајући прикључак (енгл. *hook*). Улога прикључка јесте излагање неког конкретног догађаја из језгра. Приликом сваке појаве датог догађаја прикључка, припремају се *eBPF* виртуелне машине у сврху извршавања сваког појединачног *eBPF* програма закаченог за посматрани прикључак. Током извршавања, *eBPF* програм може комуницирати са језгром користећи помоћне функције језгра за обављање предефинисаних услуга. Коначно, *eBPF* програм може бити откачен са прикључка или замењен новим, што корисницима омогућава њихово динамичко ажурирање.

eBPF програм захтева експлицитну верификацију јер током свог извршавања унутар језгра оперативног система заобилази његову уобичајену изолацију. Извршавањем у простору језгра ризикује се нарушавање сигурносних политика или чак крах оперативног система, али се истовремено избегавају и велики режијски трошкови узроковани учесталим системским позивима. Управо зато верификатор анализира *eBPF* програм, пре његовог читавања у меморију језгра, како би се осигурало да током извршавања не угрожава интегритет оперативног система. Верификатор проверава коректност приступа адресном простору и гаранцију завршетка извршавања у коначном времену као два најбитнија безбедносна својства.

Коректност приступа огледа се у писању и читању искључиво додељеног адресног простора, што верификатор проверава статичком анализом кроз праћење типова и вредности података. Овом техником верификатор може проверити директно неке од приступа адресном простору, док за друге проверава индиректно тиме што испитује да ли *eBPF* бајт код садржи проверу коректности приступа у складу са познатим опсегом додељених адреса. На пример, приликом приступа пакету, *eBPF* бајт код мора експлицитно употребити конструкт контроле тока извршавања, где се адреса наредног приступа пореди са границама пакета, како би се гарантовао приступ искључиво адресама унутар бафера коришћеног за смештање датог пакета, што је илустровано листингом 5.2.

Гаранција завршетка извршавања у коначном времену је неопходна, јер се *eBPF* програми извршавају све до краја на једној истој нити језгра покренутој у склопу припреме *eBPF* виртуелне машине, услед појаве догађаја прикључка. Уколико се *eBPF* програм не би зауставио у коначном времену, он би суштински блокирао дату нит језгра. Верификатор завршетак извршавања у коначном времену доказује представљајући *eBPF* програм у виду усмереног ацикличног графа, над којим се претрагом по дубини врши провера да ли се извршавање завршава у коначном броју извршених инструкција, за свако иницијално стање развијених (енгл. *unrolled*) конструката контроле тока извршавања са скоком уназад. Верификатор такође проверава да ли *eBPF* програм у сваком тренутку држи највише једну браву (енгл. *lock*) спречавајући тиме кружно блокирање (енгл. *deadlock*) од стране два или

```

1 void* data_end = (void*)(long)ctx->data_end;
2 void* data = (void*)(long)ctx->data;
3
4 struct ethhdr* eth_hdr = data;
5 if ((void*)(eth_hdr + 1) > data_end) {
6     return -1;
7 }
8
9 // pocev oдавde verifikator smatra bezbednim pristup eth_hdr poljima

```

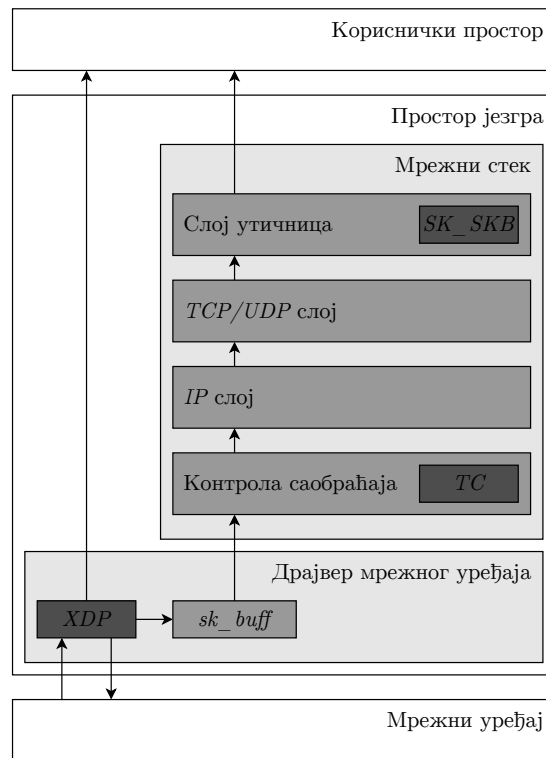
Листинг 5.2: Пример експлицитне провере коректности приступа адресном простору у циљу задовољавања критеријума *eBPF* верификатора

више *eBPF* програма. Ипак, треба имати у виду да верификатор обавља нетривијалан посао и да може садржати интерне пропусте и грешке [117], чиме се отвара могућност да пажљиво конструисани злонамерни *eBPF* програм заобиђе провере верификатора, започне извршавање и угрози интегритет језгра оперативног система.

У циљу избегавања проблема компатибилности због чврсте зависности од конкретне верзије језгра, *eBPF* програму је онемогућено позивање произвољних функција језгра оперативног система. *eBPF* програм интеракцију са језгром зато остварује позивајући искључиво наменске помоћне функције језгра, чији се апликативни програмски интерфејс сматра стабилним услед чега се постиже значајно боља преносивост између различитих верзија језгра. Тачан скуп доступних помоћних функција језгра није константан, већ зависи од конкретног прикључка на који је *eBPF* програм закачен. Зависно од конкретног прикључка на који је закачен, разликују се и прослеђени аргументи односно контекст датог *eBPF* програма. Језгро садржи велики број различитих прикључака, од којих су неки намењени за мрежне апликације, док су други намењени за безбедносне провере, надгледање и профилисање перформанси система, распоређивање процеса, складиштење података, итд.

Будући да је фокус ове дисертације мрежни домен, у наставку ће бити дат кратки преглед прикључака на путањи пакета илустрованој сликом 5.6. Први прикључак на путањи пакета назива се *XDP* јер се користи за брзу путању података (енгл. *express data path*, *XDP*). *XDP* прикључак пружа приступ пакетима директно на нивоу драјвера мрежног уређаја, што представља најранији тренутак односно најнижи могући ниво обраде пакета, услед чега омогућава ефикасну имплементацију мрежних функција. Следећи прикључак на путањи пакета назива се *TC* јер је компатибилан са подсистемом контроле саобраћаја (енгл. *traffic control*) у оквиру оперативног система. *TC* прикључак омогућава обраду пакета над инстанцом *sk_buff* структуре и пружа значајно шири скуп помоћних функција језгра везаних за мрежне сервисе. Последњи прикључак на путањи пакета назива се *SK_SKB*, а налази се након мрежног стека односно након комплетне *TCP* или *UDP* обраде пакета од стране језгра. *SK_SKB* прикључак представља догађај смештања пакета са долазне путање у пријемни ред посматране мрежне утичнице (енгл. *network socket*).

Приликом сваке појаве догађаја свог прикључка, *eBPF* програм се позива у оквиру нове *eBPF* виртуелне машине са истим почетним стањем, извршава до краја и по завршетку ослобађа све ресурсе. Услед тога, *eBPF* програм не може имплицитно одржавати стање апликације између различитих позива. Како би се превазишло ово ограничење, *eBPF* програм путем апликативног програмског интерфејса *eBPF* мапа може експлицитно приступати унапред алоцираној меморији изван своје *eBPF* виртуелне машине. *eBPF* мапа представља део меморије језгра, за складиштење елемената у виду парова кључ-вредност, са уграђеном подршком за атомичност вртењем ради закључавања (енгл. *spinlock*) у ци-



Слика 5.6: Прикључци за *eBPF* програме на путањи пакета

```

1 struct
2 {
3     __uint(type, BPF_MAP_TYPE_HASH);
4     __uint(max_entries, 1);
5     __type(key, uint32_t);
6     __type(value, uint32_t);
7 } primer_ebpf_map SEC(".maps");

```

Листинг 5.3: Пример дефиниције *eBPF* мапе

љу осигурања конзистентности у случају конкурентног приступа једном истом елементу. Животни век *eBPF* мапе одређен је бројањем њених референци и може гарантовано бити дужи од појединачног извршавања *eBPF* програма захваљујући трајном увећању броја референци помоћу експлицитног фиксирања (енгл. *pinning*) путем мултиплексираног `bpf` системског позива наводећи аргумент `BPF_OBJ_PIN` за жељену команду. Ово омогућава употребу *eBPF* мапа за комуникацију путем дељене меморије између *eBPF* програма и програма из корисничког простора, између различитих *eBPF* програма или између више позива једног истог *eBPF* програма. Приступ *eBPF* мапи, *eBPF* програм експлицитно постиже коришћењем `bpf_map_lookup_elem`, `bpf_map_update_elem` и `bpf_map_delete_elem` помоћних функција језгра, док програм из корисничког простора исту ствар постиже путем мултиплексираног `bpf` системског позива наводећи аргументе `BPF_MAP_LOOKUP_ELEM`, `BPF_MAP_UPDATE_ELEM` и `BPF_MAP_DELETE_ELEM` за жељену команду.

Постоји више типова *eBPF* мапа, различите семантике и ефикасности приступа, прикладних за одговарајуће случајеве коришћења. Неки од најистакнутијих типова *eBPF* мапа су низови, хеш мапе и прстенови (енгл. *ring buffers*). Низови обезбеђују уобичајени директан приступ елементима на основу индекса унутар фиксног броја преалоцираних улаза, хеш мапе омогућавају динамичку алокацију простора за нове елементе, док прстенови имплементирају семантику кружног бафера. Неки од типова *eBPF* мапа имају и

```

1 /* user accessible metadata for XDP packet hook
2  * new fields must be added to the end of this structure
3  */
4 struct xdp_md
5 {
6     __u32 data;
7     __u32 data_end;
8     __u32 data_meta;
9     /* Below access go through struct xdp_rxq_info */
10    __u32 ingress_ifindex; /* rxq->dev->ifindex */
11    __u32 rx_queue_index; /* rxq->queue_index */
12
13    __u32 egress_ifindex; /* txq->dev->ifindex */
14 };

```

Листинг 5.4: Дефиниција `xdp_md` структуре за представу контекста *XDP eBPF* програма

могућност засебног инстанцирања по процесору, тако да излажу засебни део меморије језгра сваком од процесора и тиме елиминишу проблеме међусобног искључења. Атрибути *eBPF* мапе укључују тип, максимални број елемената, величину кључа и величину вредности. *eBPF* мапа се ствара путем мултиплексираног `bpf` системског позива наводећи аргумент `BPF_MAP_CREATE` за жељену команду, али може бити и подразумевано створена навођењем њене дефиниције у оквиру изворног кода што је илустровано листингом 5.3.

5.1.3 XDP прикључак

XDP прикључак представља најранију тачку на путањи пакета, пре него што језгро оперативног система уопште започне било какву обраду пакета, за коју *eBPF* програм може бити закачен [118]. *eBPF* програм закачиће се за *XDP* прикључак са грануларношћу појединачног интерфејса, што омогућава његову селективну примену само на одређеним интерфејсима. *eBPF* програм закачен за *XDP* прикључак неког интерфејса у наставку дисертације биће скраћено називан *XDP eBPF* програм. Будући да барата тек пристиглим односно необрађеним пакетима и то непосредно након њиховог пријема, *XDP eBPF* програм може рано доносити одлуке о прослеђивању или одбацивању пакета уз минималне режијске трошкове. *XDP eBPF* програм може променити било који део података пакета, може чак и проширивати или смањивати пакет ради додавања или уклањања заглавља.

XDP eBPF програм започиње извршавање приликом пристизања пакета на његов интерфејс. Приликом сваког позива, *XDP eBPF* програму се прослеђује аргумент односно контекст представљен `xdp_md` структуром чија је дефиниција приказана на листингу 5.4. Директан приступ садржају пристиглог пакета могуће је остварити преко поља `xdp_md` структуре и то `data` показивача на почетак садржаја пакета и `data_end` показивача на крај садржаја пакета. Кроз поља `xdp_md` структуре могуће је приступити и другим подацима пристиглог пакета, као што су метаподаци предвиђени за наредне кораке обраде пакета `data_meta`, индекс изворног интерфејса `ingress_ifindex` и индекс пријемног реда `rx_queue_index`.

Постоји више различитих режима рада који су семантички транспарентни за сам *XDP eBPF* програм, али имају значајан утицај на перформансе његовог извршавања. *XDP offload*, *XDP native* и *XDP generic* су три доступна режима рада наведена у редоследу опадајућих перформанси извршавања. *XDP offload* режим рада подразумева измештање извршавања *XDP eBPF* програма директно на мрежни уређај, чиме се централни процесор у потпуности ослобађа оптерећења обраде пакета. Мрежни уређај мора имати подршку за

XDP offload режим рада, док *XDP eBPF* програм не сме користити помоћне функције језгра, а приликом његовог учитавања *XDP offload* режим рада мора бити експлицитно захтеван. *XDP native* режим рада подразумева извршавање програма у оквиру самог драјвера мрежног уређаја, непосредно након пријема пакета и пре него што пакет уопште стигне до језгра оперативног система. Драјвер мрежног уређаја мора имати подршку за *XDP native* режим рада, што је случај код драјвера као што су *virtio_net*, *mlx**, *ixgbe** и *vmxnet3* [119]. *XDP generic* режим рада представља резервну опцију (енгл. *fallback option*) предвиђену за мрежне уређаје без икакве подршке за извршавање *XDP eBPF* програма. Оперативни систем у *XDP generic* режиму рада емулира извршавање *XDP eBPF* програма, што омогућава његово извршавање независно од мрежног уређаја и његовог драјвера, уз неизбежне пенале по питању перформанси.

По завршетку извршавања, *XDP eBPF* програм мора одредити судбину пакета специфицирањем жељене акције, што се постиже избором једне од доступних целобројних повратних вредности описаних у наставку:

- `XDP_ABORTED` за одбацивање пакета уз пријављивање грешке.
- `XDP_DROP` за тихо одбацивање пакета.
- `XDP_PASS` за пропуштање пакета на обраду од стране мрежног стека језгра.
- `XDP_TX` за прослеђивање пакета назад преко изворног интерфејса.
- `XDP_REDIRECT` за преусмеравање пакета на неку другу дестинацију.

Прве четири акције представљају просту повратну вредност, без потребе за додатним аргументом, са јасно дефинисаним исходом. Последња акција, за преусмеравање пакета на неку другу дестинацију, захтева додатни аргумент којим се прецизира конкретна дестинација преусмеравања што може бити неки од интерфејса, неки други процесор зарад додатне обраде или специјална *AF_XDP* мрежна утичница корисничког простора. Прецизирање конкретне дестинације преусмеравања постиже се позивом `bpf_redirect` помоћне функције уз навођење жељеног интерфејса као аргумента или `bpf_redirect_map` помоћне функције уз навођење *eBPF* мапе у оквиру које треба пронаћи жељену дестинацију. Преусмеравање пакета позивом `bpf_redirect_map` помоћне функције сматра се препорученим приступом јер обезбеђује боље перформансе и притом још омогућава динамичко одређивање дестинације преусмеравања на основу садржаја *eBPF* мапе, што пружа већу флексибилност у односу на експлицитно навођење жељеног интерфејса.

5.2 Имплементација за *Netronome SmartNIC* уређај

Избор циљне платформе за имплементацију *LISPP* система је направљен под утицајем неколико фактора. Први фактор јесте могућност постизања пропусног опсега на нивоу брзине везе и што мањег додатног кашњења пакета у оба смера, ради минимизације степена деградације квалитета постојећих сервиса услед додатне обраде пакета. Други фактор представља флексибилност при дефинисању начина прослеђивања и обраде пакета, што омогућава једноставну промену политике у погледу селекције обрађиваних пакета, као и имплементацију нових *FPE* алгоритама уместо изабраног *FF3-1* алгорита. Трећи фактор се огледа у жељи за што нижом ценом имплементације и краћим временом до стављања у употребу (енгл. *deploy to production*), како би се постигла што распрострањенија примена у постојећим мрежама. Узимајући у обзир наведене факторе, као циљна платформа за имплементацију *LISPP* система одабран је *SmartNIC* уређај, конкретно, паметна мрежна картица из серије *Netronome Agilio CX* са програмабилним процесорским ресурсима.

Предност *SmartNIC* уређаја јесте то што произвољно сложену обраду пакета може вршити непосредно при интерфејсу, без прекидања централног процесора, чиме директно побољшава перформансе обраде пакета и смањује целокупно оптерећење.

Имплементација је прилагођена *Netronome Agilio CX 2 × 10GbE NFP-4000* серији *SmartNIC* уређаја [120]. Реч је о програмабилној мрежној картици са два 10 Gb/s интерфејса која може да се програмира помоћу *P4* језика и *Micro-C* језика. Примарна намена *SmartNIC* уређаја је потпуно растерећење централног процесора уз ефикасну обраду токова у домену *SDN* и виртуелних мрежних функција (енгл. *network functions virtualization, NFV*). Састоји се од 12 кластера (енгл. *cluster*) односно острва различитих архитектура. Острва се могу грубо поделити у две категорије зависно од намене садржаних језгара за обраду токова (енгл. *Flow Processor Cores, FPC*), познатих и као микромодули (енгл. *Microengines, ME*).

У прву категорију спадају острва која садрже искључиво вишенитне (енгл. *multi-threaded*) *ME* намењене за обраду пакета. У сваком *ME* постоји осам кооперативних нити, при чему се у сваком тренутку може извршавати највише једна нит, од којих свака има свој властити скуп од 32 општенаменских регистара ширине 32 бита. *ME* је представник харвардске архитектуре (енгл. *harvard architecture*): сваки има своју одвојену меморију за смештање извршног кода под називом *Code Store* и своју меморију за локалне податке под називом *Local Memory*. Поред сопствене *Local Memory*, која се најчешће користи за чување података потребних приликом обраде већине пакета, *ME* има приступ до још четири врсте меморија. Величина и време приступа сваком од типова меморија, изражено у бројевима циклуса, дати су у табели 5.1 [121]. *Cluster Local Scratch* је предвиђен за чување података потребних приликом обраде већине пакета, као и за смештање мањих табела. *Cluster Target Memory* служи за смештање заглавља пакета и за координацију *ME* са другим подсистемима. *Internal Memory* чува садржај пакета и табеле средње величине. *External Memory* чува велике табеле.

Табела 5.1: Типови меморија код *Netronome Agilio CX*

Врста меморије	Величина	Време приступа (циклуси)
<i>Code Store (CS)</i>	8 k инструкција	1
<i>Local Memory (LM)</i>	4 kB	1-3
<i>Cluster Local Scratch (CLS)</i>	64 kB	20-50
<i>Cluster Target Memory (CTM)</i>	256 kB	50-100
<i>Internal Memory (IMEM)</i>	4 MB	150-250
<i>External Memory (EMEM)</i>	3 MB + 2 GB RAM	150-500

Друга категорија укључује острва која садрже акцелераторе (*ILA, PCIe, Crypto, ARM*) и вишенитне *ME* који управљају тим акцелераторима. *SmartNIC* уређај коришћен у склопу експерименталне евалуације не садржи ниједно острво са криптографским акцелератором. Због тога је *FF3-1* алгоритам у потпуности имплементиран у софтверу. *Netronome Agilio CX* картице без криптографских акцелератора користе померачки регистар са линеарном повратном спрегом (енгл. *Linear Feedback Shift Register, LFSR*) за генерисање псеудослучајног броја, који програм извршаван на *ME* може користити као *FF3-1* кључ и параметар модификације. *LFSR* се може иницијализовати псеудослучајним семеном добијеним из временске ознаке (енгл. *timestamp*) и неке вредности коју дефинише корисник.

Програмеру није неопходно познавање посебних структура података специфичних за процесор мрежних токова (енгл. *Network Flow Processor, NFP*). *P4* преводилац аутоматски мапира различите делове *P4* програма на унутрашње ресурсе *NFP*. *P4 front-end* преводилац најпре преводи *P4* програм у међуреферентацију. *Netronome P4 back-end* преводилац затим трансформише *IR* у *Micro-C* програм, који се потом може превести и повезати (енгл. *link*)

да би се генерисао *NFP* извршни код помоћу преводиоца за мрежне токове (енгл. *network flow C compiler*) [122]. Извршни код генерисан из *P4* кода учитава се на већи број *ME* јединица, од којих свака може независно да обрађује пакете у складу са кодом за обраду пакета написаним на *P4* језику. Зарад остваривања преносивости имплементације на већи број *P4* уређаја односно циљева (енгл. *targets*), у оквиру ове докторске дисертације истражена је и преносива имплементација искључиво у *P4* језику. *Netronome* подржава извршавање *P4* програма написаних за *v1model* архитектуру, а таква имплементација би теоретски требало да буде преносива на било који *P4* циљ са *v1model* архитектуром, од којих је један и симулатор *Behavioral Model (BMv2)* [123]. Коришћени су алати из *Netronome SDK* верзије *6.1-preview*, што је прва верзија која има подршку за *P4-16* језик. Изворни код свих *LISPP* имплементација описаних у наредним подсекцијама је доступан у саставу једног истог репозиторијума [124].

5.2.1 Имплементација у целости на *P4* језику

Највећи изазови при имплементацији *FF3-1* алгоритма користећи *P4* језик проистичу из ограничења која директно намеће *P4* језик [125]. Како је реч о доменски специфичном језику за програмирање равни података мрежних уређаја, *P4* језик не садржи конструкт петљи (енгл. *loop construct*) у жељи да се тиме обезбеди предвидиво време извршавања програма. Непостојање конструкта петљи у оквиру *P4* језика представља озбиљан изазов у имплементацији симетричних криптографских алгоритама који се састоје из већег броја рунди ради постизања конфузије и дифузије података. Одмотавање петљи (енгл. *loop unrolling*) на нивоу читавог алгоритма није одржива опција због ограничења наметнутог величином *Code Store*, где се програмски код смешта. Ограничење наметнуто величином *Code Store* посебно долази до изражаја због архитектуралног лимита специфицираног у *Netronome SDK* документацији, према којем је дозвољено дефинисати највише 256 акција, притом уз очекивано повећање *Code Store* заузећа са дефинисањем сваке нове акције. За свако позивање акције, *Netronome P4 back-end* преводилац дефинише нову *Micro-C* функцију која обезбеђује контекст односно аргументе и затим позива дату акцију. Дефинисање нове *Micro-C* функције за свако позивање акције доводи до осетног повећања величине програмског кода. Штедња *Code Store* простора постаје још важнија када се узме у обзир да *Netronome P4 front-end* преводилац нема подршку за *P4* функције.

Због свега претходно наведеног, ограничење у виду непостојања конструкта петље било је могуће превазићи једино поновним враћањем (енгл. *resubmit*) пакета на почетак *улазног тока обраде* (енгл. *ingress pipeline*) за сваку рунду *FF3-1* алгоритма. Уместо реплицирања изворног кода за једну целу *FF3-1* рунду већи број пута, позивањем *resubmit* екстерне функције из *v1model* архитектуре пакет се поново враћа на почетак улазног тока обраде, започињући тиме још једну рунду *FF3-1* алгоритма што је листингом А.1 илустровано на упрошћен начин. Унутрашње стање *FF3-1* алгоритма морало је бити сачувано у метаподацима пакета, који надживљавају поновно враћање пакета на почетак улазног тока обраде, како би било очувано између рунди.

За имплементацију *AES* шифровања искоришћен је приступ заснован на предшифрованим табелама за претраживање (енгл. *scrambled lookup tables*) [126]. Предност овог приступа је што користи унапред проширени *AES* кључ помоћу наменских табела чије генерисање и примена су приказани на листингу А.2. Захваљујући овом приступу читаво *AES* шифровање могло би се спровести током само једног проласка пакета кроз улазни ток обраде, али би то захтевало истовремено постојање чак 160 табела за упаривање и акције услед изостанка подршке за низове у *P4* језику. Наиме, сваки од шеснаест бајтова стања *AES* алгоритма захтева засебну табелу, из разлога што се једна иста табела не може применити више пута током једног проласка пакета кроз улазни ток обраде. Додатно, свих шеснаест

табела морају бити реплициране за сваку од десет рунди *AES* алгоритма, што укупно даје претходно поменутих 160 табела. Овако велики број табела негативно утиче на кашњење [127], посебно имајући у виду да *P4 back-end* преводилац смешта све табеле у екстерну меморију, која има најдуже време приступа.

Захтевани број табела премашује *Netronome* ограничење по питању броја табела за упаривање и акције које се користе у оквиру улазног тока обраде *P4* програма [128]. Због тога је број табела морао бити смањен на свега пет: четири различите за стандардне *AES* рунде и једну за последњу *AES* рунду. Смањење броја табела имплицира увођење додатних поновних враћања пакета на почетак улазног тока обраде ради успешне имплементације *AES* шифровања. Примитивна имплементација би пропущтала пакет кроз улазни ток обраде једном за сваки од шеснаест бајтова стања *AES* алгоритма, и тако за сваку рунду *AES* алгоритма. Такав наивни приступ захтева чак $8 \times (1 + 9 \times 16 + 1 \times 16 + 1) = 1296$ поновних подношења пакета за сваки долазећи пакет. Са занемарљиво сложенијом имплементацијом, у којој се све четири различите табеле за стандардне *AES* рунде примењују у једном проласку кроз улазни ток обраде, могуће је смањити број поновних враћања пакета на $8 \times (1 + 9 \times 4 + 1 \times 16 + 1) = 432$.

5.2.2 Имплементација са *AES* алгоритмом на *Micro-C* језику

Велики број поновних враћања пакета доводи до значајног смањења пропусног опсега, као што је приказано у подсекцији 6.1.1. Из тог разлога, ради побољшања пропусног опсега, покушана је замена делова *P4* кода *Micro-C* кодом. Програмски језик *Micro-C* је најефикаснији начин за програмирање *Netronome Agilio SmartNIC* уређаја, јер пружа подршку за експлицитно коришћење структура података специфичних за *NFP* архитектуру [129]. Програмирање у *Micro-C* језику за *NFP* се помало разликује од програмирања у стандардном *C* језику за уређај са регуларним оперативним системом, јер су *NFP* структуре података и меморије специфичне за *NFP* архитектуру. Први корак на путу замене делова *P4* кода са *Micro-C* кодом као заокружених целина, јесте да се целокупно *AES* шифровање пребаци на *Micro-C* језик, јер *AES* шифровање представља најсложенији део *FF3-1* алгоритма, а притом на неким *SmartNIC* уређајима може бити имплементирано и коришћењем хардверског акцелератора.

Велика пажња приликом имплементације је посвећена избору коришћених типова података због специфичности *Netronome Agilio* архитектуре. Преводилац подржава 8-битне и 16-битне типове података и њихове одговарајуће показиваче, али уз потенцијални губитак у перформансама. У оквиру документације наведена је препорука да 8-битне и 16-битне типове података не би требало користити, јер приступ вредностима мањим од 32 бита (64 бита за интерну или екстерну меморију) обично подразумева додатне операције за издвајање одговарајућих бајтова из 32-битних или 64-битних меморијских речи. Приступ преко показивача на 8-битне и 16-битне типове може такође захтевати поравнање података у току извршавања, што је још неефикасније. Стога нису коришћени ни 8-битни ни 16-битни типови података, нити показивачи на исте. Стање *AES* алгоритма је дефинисано као 128-битна структура са четири поља 32-битног типа података као што је приказано у оквиру листинга А.3. Дата структура се преноси искључиво по вредности, како би се избегли поменути пенали приликом коришћења показивача.

Зарад повећања пропусног опсега, имплементација је и даље заснована на предшифрованим табелама за претраживање. Алгоритам је знатно убрзан предрачунањем дела унутрашњих операција које *AES* алгоритам извршава и чувањем тих резултата у табелама за претраживање [130]. Пошто је садржај *P4* табела за претраживање непроменљив, оне не морају бити локалне за појединачне нити. Дељењем ових табела између већег броја нити постиже се боља искоришћеност меморијског простора. Стога су табеле експлицитно

означене као заједничке користећи `shared` модификатор и смештене у доступне меморије са што мањим кашњењем што је приказано на листингу А.4. Само једна од четири табеле за стандардне *AES* рунде додељена је најбржој *Local Memory*, јер услед просипања вредности регистара (енгл. *register spilling*) нема више слободног простора у *Local Memory*. Преостале табеле за стандардне *AES* рунде и табела за последњу *AES* рунду смештене су у нешто спорију *Cluster Local Scratch* меморију. Преводиоцу се за већину функција сугерише уграђивање директно на месту позива (енгл. *inlining*) у циљу убрзања извршавања. Пренос већине аргумената по вредности захтева се кроз општенаменске регистре користећи `gp_reg` модификатор ради што ефикаснијег преноса аргумената.

Премошћавање јаза између *Micro-C* језика и *P4* језика постигнуто је механизмом екстерних *P4* функција. Дефинисање екстерне *P4* функције на *Micro-C* језику захтева укључивање `pif_plugin.h` заглавља унутар којег су садржане неопходне дефиниције за интеграцију са *P4* доменом. Идентификатори екстерних *P4* функција морају као префикс имати `pif_plugin_` на *Micro-C* језику. Овакав услов приликом именовања екстерних *P4* функција је неопходан како би линкер био у могућности да на исправан начин изврши повезивање њихове дефиниције са позивом из *P4* домена где је дати префикс природно изостављен.

Свака екстерна *P4* функција, зарад приступа парсираним заглављима пакета и упареним подацима, мора имати параметре типа `EXTRACTED_HEADERS_T` и `MATCH_DATA_T`, док повратна вредност функције мора бити целобројни идентификатор акције за извршавање над посматраним пакетом. У таквој поставци, један од начина да се из *P4* домена проследи додатни улазни аргументи до екстерне *P4* функције и врате резултати из екстерне *P4* функције назад до *P4* домена јесте помоћу метаподатака парсираних заглавља пакета односно комуникацијом путем дељене меморије у виду `EXTRACTED_HEADERS` структуре. Унутар *Micro-C* функције, прослеђене додатне улазне аргументе могуће је дохватити из метаподатака коришћењем `pif_plugin_meta_get_` функција, док је враћање резултата назад могуће постићи постављањем вредности у метаподатке коришћењем `pif_plugin_meta_set_` функција. Ланац алата аутоматски генерише засебне `pif_plugin_meta_get_` и `pif_plugin_meta_set_` функције за сваки од елемената метаподатака на основу њихових дефиниција из структуре *P4* метаподатака. Овај механизам илустрован је листингом А.5, где је приказана дефиниција екстерне *P4* функције за шифровање помоћу *AES* алгоритма. Имплементација *AES* шифровања на *Micro-C* језику резултовала је око 45 пута већим пропусним опсегом у поређењу са имплементацијом у целости на *P4* језику, што је детаљније описано у поглављу 6.

5.2.3 Имплементација са *FF3-1* алгоритмом на *Micro-C* језику

Коначно, целокупна имплементација *FF3-1* алгоритма је пребачена са *P4-16* на *Micro-C* језик, тако да су само парсирање и филтрирање пакета остали на *P4* коду. Исти принципи кодирања, које намеће *Netronome Agilio* архитектура, коришћени су за *FF3-1* имплементацију на *Micro-C* језику. Поред тога, специфична операција ротирања бита (енгл. *bit-reversal*) код *FF3-1* алгоритма реализована је коришћењем табела за претраживање, као што је илустрована листингом А.6, уместо маскирањем и померањем бита. Ова имплементација је резултовала пропусним опсегом на нивоу брзине везе.

5.3 Имплементација за *eVPF* технологију

Ослањање на наменске мрежне уређаје за потребе имплементације *LISPP* система може представљати значајну препреку за његову ширу примену, првенствено из економских и

практичних разлога. Насупрот томе, имплементација *LISPP* система прилагођена стандардним извршним окружењима значајно олакшава усвајање од стране мрежних оператора, јер елиминише потребу за комплексним изменама инфраструктуре и додатним улагањима. Наивна идеја обраде сваког пакета у корисничком простору (енгл. *user space*) стандардних оперативних система, јесте примамљива због једноставности стављања у употребу, али лако се може показати као неефикасна у пракси. Високи режијски трошкови, узроковани учесталим системским позивима и преласком између корисничког простора и простора језгра (енгл. *kernel space*) оперативног система за сваки појединачни пакет, чине овај приступ неодрживим за мреже са протоком великог броја пакета. Сходно томе, имплементација *LISPP* система унутар самог језгра намеће се као императив за постизање задовољавајућих перформанси.

Имплементација *LISPP* система унутар језгра оперативног система теоријски се може постићи директном изменом његовог изворног кода. Међутим, овакав приступ је у пракси често неприхватљив јер захтева поновно превођење језгра што би као нетривијалан процес одвратило већину мрежних оператора. Алтернатива у виду модула језгра (енгл. *kernel modules*) омогућава динамичко проширење функционалности без поновног превођења језгра, али са собом носи инхерентне безбедносне ризике и извесне проблеме. Будући да модули језгра имају приступ меморији и ресурсима језгра оперативног система, свака намерна или ненамерна грешка у оквиру модула језгра може довести до нестабилности или потпуног краха оперативног система, што ствара оправдану резервисаност према њиховој употреби у продукционим окружењима. Модули језгра су чврсто везани за конкретну верзију језгра, што имплицира да ажурирање језгра оперативног система често са собом повлачи прилагођавање изворног кода модула језгра и његово поновно превођење ради очувања компатибилности. Додатно, програмирање модула језгра захтева напредно познавање унутрашњих механизма оперативног система, што представља додатну баријеру за програмере.

Као одговор на наведене изазове, за имплементацију *LISPP* система одабрана је *eBPF* технологија [131] за модификацију, интеракцију и програмирање језгра оперативног система у време извршавања. *eBPF* технологија обухвата једноставну општенаменску виртуелну машину унутар језгра оперативног система са гаранцијом безбедности захваљујући изолацији (енгл. *sandboxing*) датих виртуелних машина и строгом процесу верификације кода пре читавања у меморију језгра. У комбинацији са *XDP* прикључком, остварује неупоредиво боље перформансе обраде пакета у поређењу са обрадом пакета посредством мрежног стека у оквиру језгра. *XDP* прикључак омогућава пресретање и руковање пакетима на најранијој могућој тачки на путању обраде пакета односно одмах по пријему пакета од стране интерфејса. Тиме се заобилази већи део мрежног стека у оквиру језгра, што значајно смањује кашњење пакета и повећава пропусни опсег. Према томе, имплементација *LISPP* система заснована на *eBPF* технологији и *XDP* прикључку представља компромисно решење између обраде пакета на наменским мрежним уређајима и посредством мрежног стека у оквиру језгра оперативног система.

5.3.1 Имплементација помоћу *XDP*

У циљу постизања што бољих перформанси обраде пакета, имплементација *LISPP* система реализована је у виду *XDP eBPF* програма. Неопходна су два независна *XDP eBPF* програма, при чему је први одговоран за шифровање излазних пакета, док други обавља дешифровање пакета при повратку. Улазна тачка *XDP eBPF* програма, за шифровање излазних пакета од стране *LISPP* система, дефинисана је функцијом `lispp_if_encrypt` чији је изворни код приказан унутар листинга Б.1. Функција `lispp_if_encrypt` означена је `SEC("xdp/lispp-if-encrypt")` макроом неопходним ради закачињања посматраног *eBPF*

програма на жељени *XDP* прикључак у оквиру језгра оперативног система. Посматрани `SEC(name)` макро се експандује у `__attribute__((section(name), used))` директиву *clang* преводиоца за додељивање атрибута функцијама. Ефекат доделе `section(name)` атрибута функцији јесте смештање генерисаног објектног кода посматране функције у секцију извршног и повезивог формата (енгл. *executable and linkable format, ELF*) задатог имена `name`.

Приликом учитавања *eBPF* програма у меморију језгра, задато име *ELF* секције користи се управо за избор прикључка на који ће *eBPF* програм бити закачен [132]. Имена *ELF* секција нису део никаквог стандарда нити *eBPF* конвенције, већ их тумачи *bpftool* алат односно *libbpf* библиотека коришћена за учитавање *eBPF* програма у језгро оперативног система. *bpftool* алат је део репозиторијума изворног кода *Linux* језгра, а може се користити за учитавање и дебаговање *eBPF* програма, као и за стварање и руковање *eBPF* мапама. *libbpf* библиотека је такође део репозиторијума изворног кода *Linux* језгра, а имплементира једноставан апликативни програмски интерфејс као омотач око мултиплексаног *bpf* системског позива.

Функција `lispp_if_encrypt` обрађује пристигли излазни пакет представљен контекстом у виду `xdr_md` структуре. Први корак обраде пакета јесте конверзија типа поља `data_end` и `data` из `xdr_md` структуре у показиваче на крај и почетак садржаја пакета. Показивач на крај садржаја пакета је од изузетног значаја, јер се захваљујући поређењу са њим приликом сваког наредног приступа садржају пакета испуњава коректност приступа адресном простору у складу са захтевима верификатора. Показивач на почетак садржаја пакета користи се за приступ садржају пакета зарад парсирања заглавља. С обзиром да се заглавља парсирају једно за другим, потребно је одржавати и показивач на текућу позицију унутар садржаја пакета до које се стигло након последњег парсираног заглавља, за шта се користи `next_hdr` променљива.

Парсирање почиње од *Ethernet* заглавља позивом `parse_ethhdr` функције чија је имплементација дата у виду листинга Б.6, Б.5 и Б.4. Повратна вредност `parse_ethhdr` функције указује да ли је парсирање *Ethernet* заглавља успешно извршено враћајући тип протокола садржаног унутар *Ethernet* оквира. Уколико је парсирање *Ethernet* заглавља успешно извршено, наставља се даље са парсирањем *IPv4* заглавља позивом `parse_iphdr` функције чија је имплементација дата у виду листинга Б.7. Повратна вредност `parse_iphdr` функције указује да ли је парсирање *IPv4* заглавља успешно извршено враћајући тип протокола садржаног унутар *IPv4* пакета. Уколико је парсирање *IPv4* заглавља успешно извршено, наставља се даље са парсирањем *TCP* заглавља позивом `parse_tcphdr` функције чија је имплементација дата у виду листинга Б.8. Повратна вредност `parse_tcphdr` функције указује да ли је парсирање *TCP* заглавља успешно извршено враћајући величину *TCP* заглавља. Уколико је парсирање *TCP* заглавља успешно извршено, коначно се долази до сржи *LISPP* система односно замагљивања дела изворишне *IPv4* адресе и читавог изворишног порта. У случају да парсирање било ког од претходно наведених заглавља није успешно извршено, пакет се без икаквих измена пропушта даље на уобичајену обраду од стране мрежног стека језгра навођењем `XDP_PASS` за повратну вредност `lispp_if_encrypt` функције.

Срж *LISPP* система за обраду *IPv4* пакета приказана је на листингу Б.2. Зависно од архитектуре домаћина, а узимајући у обзир да *XDP eBPF* програм барата пакетима тек пристиглим са мреже, може постојати потреба да се изврши конверзија мрежног формата у домаћинов формат за свако поље заглавља пакета величине веће од једног бајта. Мрежни формат заснива се на *big-endian* редоследу смештања бајтова за вредности величине веће од једног бајта, док већина савремених архитектура домаћина користи *little-endian* редослед смештања бајтова. Описана конверзија је скоро увек потребна тако да *libbpf* библиотека садржи `bpf_ntohs` и `bpf_ntohl` макро дефиниције за конверзију 16-битних `short` вредности и 32-битних `long` вредности из мрежног формата у стварни домаћинов формат.

После успешног парсирања свих претходно наведених заглавља, издвајају се део изворишне *IPv4* адресе и читав изворишни порт на основу чега се формира отворени текст односно улаз *FF3-1* алгоритма. Над тако формираним отвореним текстом се извршава *FF3-1* алгоритам позивом функције `ff3_1_encrypt`, а добијени шифровани текст се користи за ажурирање дела изворишне *IPv4* адресе и читавог изворишног порта у одговарајућим заглављима пакета. Функција `ff3_1_encrypt` је имплементирана на исти начин као и у случају *Netronome* имплементације *FF3-1* алгоритма на *Micro-C* језику, укључујући употребу предшифрованих табела за претраживање, али уз незнатне измене услед једноставније меморијске хијерархије.

Како би се очувао интегритет пакета након ажурирања дела изворишне *IPv4* адресе и читавог изворишног порта, неопходно је израчунати нове контролне суме за *IPv4* и *TCP* заглавља. Постоје `bpf_13_csum_replace` и `bpf_14_csum_replace` помоћне функције језгра за поновно израчунавање контролних сума протокола трећег и четвртог слоја *OSI* модела, али оне нису доступне у случају *XDP* прикључка зато што као аргумент захтевају `sk_buff` структуру која се инстанцира тек непосредно пре *TC* прикључка. Зато се контролна сума *IPv4* заглавља израчунава позивом `csum_ipv4` функције чија је имплементација приказана листингом Б.9, док се контролна сума *TCP* заглавља израчунава инкрементално, ради постизања што бољих перформанси, позивом `csum_tcp_incremental_ipv4` функције чија је имплементација приказана листингом Б.10.

Након шифровања дела изворишне *IPv4* адресе и читавог изворишног порта, као и поновног израчунавања контролне суме, пакет се преусмерава на одговарајући излазни интерфејс користећи `bpf_redirect_map` помоћну функцију језгра. Неопходно је навести `decrypt_if_idx eBPF` мапу чија је дефиниција доступна кроз листинг Б.3, као аргумент `bpf_redirect_map` функције. Садржај `decrypt_if_idx eBPF` мапе може се попуњавати из корисничког простора, на пример, кроз командну линију извршавањем наредбе `bpftool map update name decrypt_if_idx key <key> value <value>`, приликом иницијализације *LISPP* система, где се као вредност користи индекс интерфејса на који је закачен *XDP eBPF* програм за дешифровање пакета.

Евалуација овог и свих претходно описаних система дата је у наредном поглављу. Прво су изложени резултати за *Netronome* имплементацију, са посебним освртом на случај смањења броја рунди *AES* алгоритма. Након тога, приказани су резултати за *eBPF* имплементацију.

6. Евалуација *LISPP* система

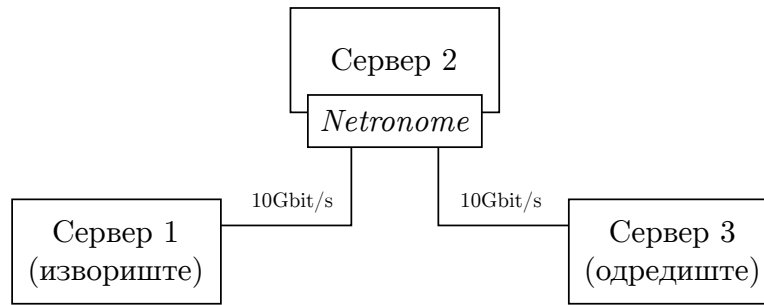
У оквиру експерименталне евалуације перформанси коришћена су три савремена алата за активно надгледање мрежа: *iPerf2* [133] за мерење пропусног опсега (енгл. *bandwidth*), *PF_RING Zero Copy* [134] за мерење протока пакета (енгл. *packet rate*) у јединици времена и *Sockperf* [135] за мерење кашњења (енгл. *latency*). Сваки од ових алата врши активно надгледање у смислу да генерише тестни мрежни саобраћај за који затим обавља мерење перформанси тестне мреже. *iPerf2* је предвиђен за процену максималног остваривог пропусног опсега *TCP* или *UDP* саобраћаја. Он успоставља *TCP* сесије између изворишта и одредишта тестног саобраћаја како би одредио највећи оствариви пропусни опсег. *iPerf2* може постићи пропусни опсег већи од 10 Gb/s са подразумеваним параметрима и једним *TCP* мрежним током [136]. *PF_RING* је нови тип мрежне утичнице који побољшава брзину снимања пакета (енгл. *packet capture*) избегавајући било какве интервенције потпуним заобилажењем језгра оперативног система и има подршку за пропусни опсег до 100 Gb/s за било коју величину пакета. *Sockperf* је услужни програм за евалуацију перформанси мрежа осмишљен примарно за тестирање кашњења са резолуцијом испод наносекунде. Он може да измери кашњење сваког појединачног пакета, чак и при оптерећењу од неколико милиона пакета у секунди.

6.1 Евалуација *Netronome LISPP* система

Перформансе *Netronome LISPP* система су темељно евалуиране коришћењем стварних мрежних уређаја, а не симулираног окружења. Два сервера, са *Intel® Xeon® E5-2660* процесором и 40 GB оперативне меморије, коришћена су као извориште и одредиште тестног саобраћаја. Ова два сервера су међусобно повезана преко *Netronome Agilio CX* програмабилне мрежне картице са два 10 Gb/s интерфејса, инсталиране на трећем серверу са *Intel® Xeon® E5-2680* процесором и 64 GB оперативне меморије, као што је приказано на слици 6.1. Везе (енгл. *links*) између самих сервера успостављене су преко 10 Gb/s интерфејса на свичу. Ови интерфејси су коришћени искључиво за потребе тестног окружења (енгл. *test bed*), како не би било мешања било каквог другог саобраћаја са тестним саобраћајем. Максимална трансмисиона јединица (енгл. *Maximum Transmission Unit, MTU*) на свим интерфејсима је 1500 бајтова у свим експериментима, осим уколико није другачије наглашено.

6.1.1 Експериментална евалуација

Приликом евалуације перформанси, тестни мрежни саобраћај је слат од изворишног ка одредишном серверу кроз *LISPP* систем на *Netronome SmartNIC* уређају. Табела 6.1 приказује *iPerf2 TCP* пропусни опсег за три различите *FF3-1* имплементације описане у секцији 5.2. Сва мерења су извршена за *IPv4* и *IPv6* користећи мрежне маске /24 и /64, што значи да су дужине отвореног и шифрованог текста износиле 24 и 80 бита, респективно. За референтну вредност, приказану у врсти *Pass-Through*, измерен је *TCP* пропусни опсег за



Слика 6.1: Поставка за тестирање *Netronome LISPP* система

извршни код који прослеђује саобраћај са једног на други интерфејс картице без икаквих измена.

Табела 6.1: *TCP* пропусни опсег за три различите *Netronome LISPP* имплементације

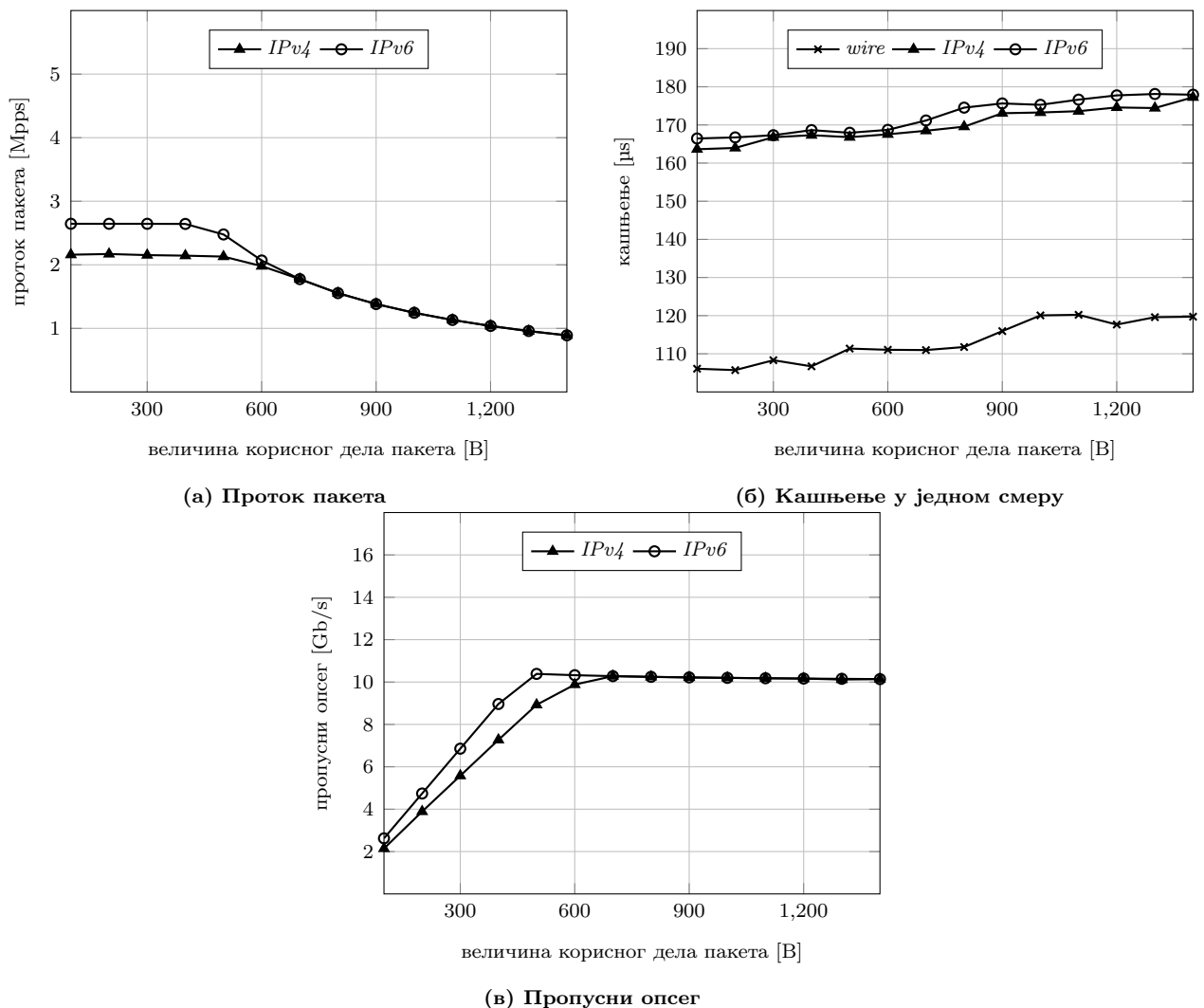
Имплементација Протокол	Само <i>P4</i>		<i>AES</i> (μC)		<i>FF3-1</i> (μC)	
	IPv4	IPv6	IPv4	IPv6	IPv4	IPv6
<i>Pass-Through</i> [Gb/s]	9,38	9,26	9,38	9,26	9,38	9,26
<i>FF3-1</i> [Gb/s]	0,156	0,149	7,27	6,97	9,38	9,26

Резултати мерења показују да последња имплементација, где је целокупан *FF3-1* алгоритам написан на *Micro-C* језику, има максимални пропусни опсег једнак брзини везе. Насупрот томе, прве две имплементације, само *P4* и она у којој је *AES* алгоритам написан на *Micro-C* језику, остварују знатно мање пропусне опсеге. Оваква разлика у перформансама сугерише да превођење *P4* језика у извршни код, за сложену обраду пакета, на конкретној хардверској конфигурацији није оптимално. Овим се намеће закључак да, иако је коришћењем искључиво *P4* језика могуће постићи преносивост кода на друге платформе, перформансе извршног кода нису загарантоване. Резултати такође показују разлику у пропусном опсегу за *IPv4* и *IPv6*. Ова разлика може се приписати разлици у величини заглавља, које је веће за *IPv6*, чиме се смањује користан опсег за податке у *TCP* мрежном току, иако не треба занемарити ни утицај обраде на крајњим тачкама. Поред *TCP* тестова пропусног опсега, једносмерни тестови саобраћаја коришћењем *PF_RING Zero Copy* токова пакета показали су да систем може постићи пропусни опсег на нивоу брзине везе, тј. 10 Gb/s, када је целокупна *FF3-1* имплементација на *Micro-C* језику.

Оперативни трошкови по пакету остају константни за све величине пакета, јер *LISPP* обавља шифровање/дешифровање искључиво појединих поља заглавља пакета. Поља заглавља пакета имају исту величину у сваком пакету који *LISPP* обради. Као резултат тога, оперативни трошак није под утицајем величине корисног дела (енгл. *payload*) пакета, чиме се обезбеђује уједначеност оперативних трошкова без обзира на промене у величини пакета. Према томе, ограничења *LISPP* система по питању перформанси боље се изражавају кроз максимални могући проток пакета, а не према стандардним мерама пропусног опсега. Овај аспект постаје посебно значајан у контексту *TCP* протокола, који често користи пакете величине једнаке *MTU* за пренос велике количине података, што је уобичајена појава код тестирања *TCP* пропусног опсега. Важно је нагласити и да је *LISPP* систем у потпуности самостално оперативан на *Netronome Agilio CX* картици. Не постоји потреба за било каквом интеракцијом са сервером, осим иницијалног превођења изворног кода и затим учитавања конфигурације, што доводи до занемарљивог коришћења процесора рачунара на који је *Netronome Agilio CX* картица повезана.

Сprovedено је и тестирање са циљем оцене утицаја величине тестних пакета на пропусни опсег, проток пакета и кашњење пакета применом *Netronome LISPP* имплементације са

FF3-1 на *Micro-C* језику. Сlike 6.2в и 6.2а приказују пропусни опсег и проток пакета измерено коришћењем *PF_RING Zero Copy* алата. Тестни мрежни саобраћај, сачињен од пакета искључиво већих од 600 бајтова, у потпуности засићује везу између тестних сервера на 10 Gb/s, због чега су забележене ниже вредности протока пакета (на пример, веза од 10 Gb/s је у потпуности засићена слањем било 1,25 милиона пакета величине 1000 бајтова, или приближно 0,9 милиона пакета величине 1400 бајтова у секунди). Тестни мрежни саобраћај, сачињен од пакета искључиво мањих од 600 бајтова, открива највећи проток пакета који се може постићи са *LISPP* системом на датој *Netronome Agilio CX* картици, достижући 2,16 Mpps за *IPv4* и 2,64 Mpps за *IPv6*. Слика 6.2б приказује кашњење у једном смеру које *LISPP* уноси за величине пакета од 100 до 1400 бајтова, при чему је за референтну вредност означену као *wire* измерено је кашњење у једном смеру за пакете који пролазе кроз *Netronome Agilio CX* картицу без примене *LISPP* или било какве друге обраде. Значајно је напоменути да *LISPP* имплементација са потпуним *FF3-1* на *Micro-C* језику конзистентно додаје око 60 μ s кашњења, без обзира на величину пакета, чиме се потврђује хипотеза наведена у претходном пасусу.



Слика 6.2: Проток пакета, кашњење пакета у једном смеру и пропусни опсег са *FF3-1* на *Micro-C* језику за пакете различитих величина код *Netronome LISPP* система

Подаци приказани на слици 6.2б такође указују да кашњење које *Netronome LISPP* уноси не зависи од величине шифрованог текста. Додатно кашњење остало је доследно за *IPv4*, са 24-битним шифрованим текстом, и за *IPv6*, са 80-битним шифрованим текстом. Овај налаз сугерише да је *LISPP* систем скалабилан са бројем уређаја односно величином

заштићене мреже, што директно дефинише величину хост дела адресе који се шифрује. На крају, кључни параметар који дефинише границе перформанси *LISPP* система је проток пакета. За потребе већег протока пакета, потребно је додатно тестирање са снажнијим мрежним акцелераторима и примена механизма за распоређивање оптерећења (енгл. *load balancing*).

6.1.2 Смањење броја *AES* рунди

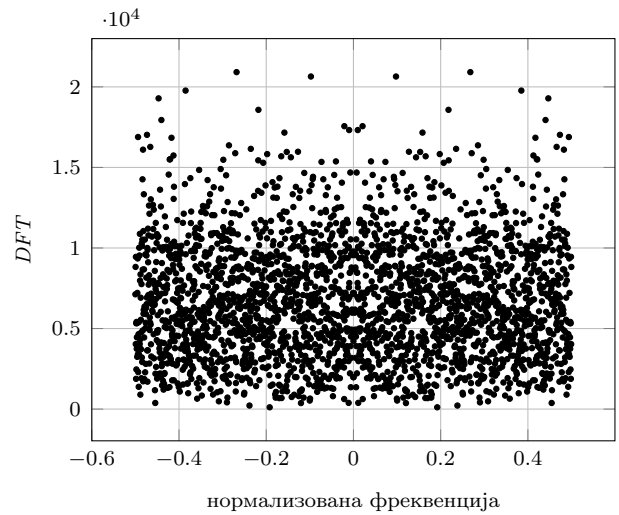
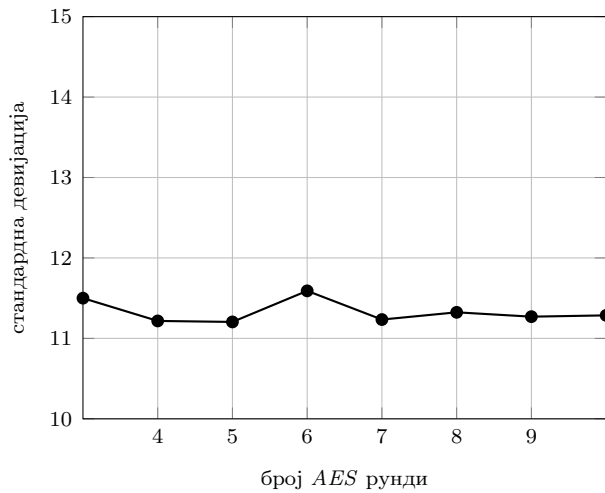
FF3-1 је алгоритам са осам рунди где се у свакој рунди позива *AES* шифровање, које пак има десет својих унутрашњих рунди. У оквиру *Netronome LISPP* имплементације, где је *FF3-1* у потпуности написан на *Micro-C* језику, *AES* заузима 54% процесорског времена. Смањење броја рунди у било ком од ових алгоритама побољшало би време обраде и укупни пропусни опсег алгоритма. Међутим, таква измена потенцијално би довела до смањења безбедности алгоритма. Недавни криптоаналитички радови показали су да комплексност напада на *FPE* [82] зависи од броја Фајстелових рунди. Смањење броја Фајстелових рунди смањило би комплексност напада на *FF3-1* алгоритам, што није прихватљива опција.

Током евалуације кандидата у завршној фази избора *AES* алгоритма, откривено је да излаз *Rijndael* алгоритма, који је касније изабран за *AES* алгоритам, делује насумично већ након прве три рунде. Наредне рунде производе насумичност сличну оној добијеној у трећој рунди [137]. Пошто се *AES* алгоритам у оквиру *FF3-1* користи искључиво као генератор псеудослучајних бројева, а не за шифровање података, повећање перформанси може се постићи смањењем броја рунди *AES* алгоритма без жртвовања сигурности *FPE* алгоритма.

Сprovedени су тестови насумичности над низовима *FF3-1* шифрованих хост адреса. Низови адреса добијени су шифровањем 9-битне хост адресе и 16-битног изворишног порта. При сваком покретању, одабрано је 20 насумичних хост адреса за које је итерирано свих 65 536 различитих вредности изворишног порта и анализиран је низ добијених шифрованих хост адреса. Исти тестови спроведени су за *FF3-1* имплементације са *AES* алгоритмом од 3 до 10 рунди. Коришћењем Дискретне Фуријеове трансформације (енгл. *Discrete Fourier transform*), извршена је и спектрална анализа тестова из уобичајених скупова тестова насумичности [138]. У сваком случају, потврђена је хипотеза да је излазни низ насумичан.

Слика 6.3а приказује стандардна одступања емпиријски уочене фреквенције појављивања свих могућих *IP* адреса у 9-битном опсегу адреса, за различит број *AES* рунди. Као што се види на слици, стандардна одступања имају приближно исту вредност без обзира на број рунди, што значи да се варијабилност емпиријски уочене фреквенције не мења са бројем рунди. Слика 6.3б приказује *DFT* амплитуде низа шифрованих адреса добијених коришћењем *FF3-1* са *AES* алгоритмом од тачно 3 рунде. Визуелна инспекција показује да је спектар „раван” за читав опсег вредности фреквенција, без вредности које значајно прелазе праг, потврђујући да се шифровани низ адреса понаша као насумичан низ. Ово сугерише да се код *FF3-1* могу постићи побољшања перформанси смањењем броја *AES* рунди без жртвовања безбедности алгоритма. Ипак, потребна је детаљнија анализа ове хипотезе у области криптоанализе.

У трећем низу тестова перформанси анализирано је потенцијално повећање протока пакета смањењем броја *AES* рунди. Упућено је десет милиона пакета, величине 100 бајтова, у секунди кроз *LISPP* систем и измерен је број пакета који су стигли до одредишта. Слика 6.4а приказује проток пакета. Имплементација са *FF3-1* на *Micro-C* језику може обрадити више од 2 Mrps, док се проток пакета може скоро удвостручити коришћењем *AES* алгоритма са свега три рунде за функцију рунде. Интересантно је приметити да *Netronome Agilio CX* картица постиже већи проток пакета за *IPv6* пакете, што је вероватно последица једноставније обраде *IPv6* заглавља на самој картици, услед непостојања контролне суме

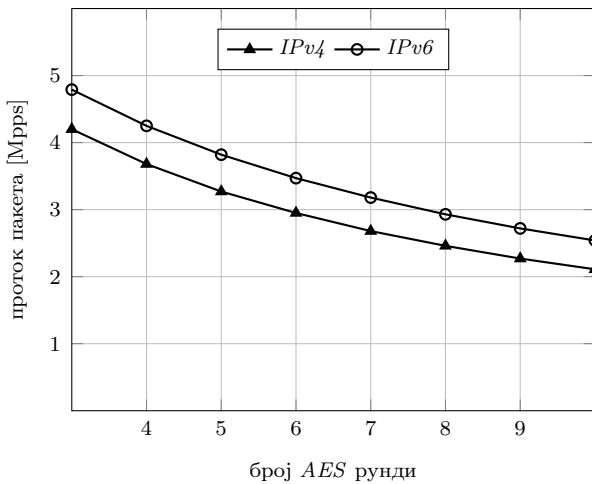


(а) Стандардна девијација униформних интервала

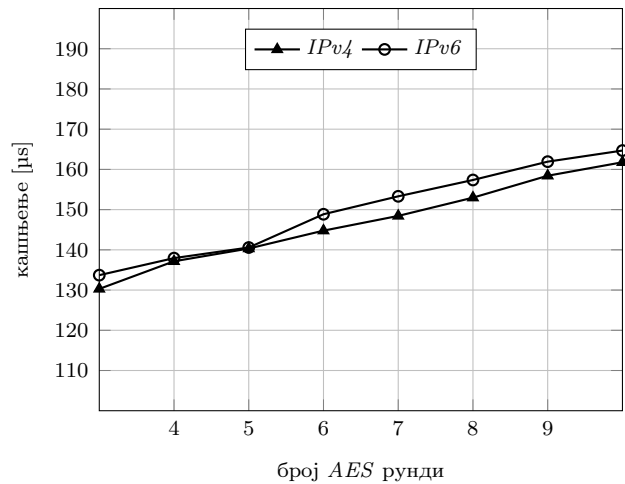
(б) DFT временске серије AES -3 шифровања

Слика 6.3: Стандардно одступање емпиријски уочене фреквенције појављивања свих могућих шифрованих IP адреса за различит број AES рунди и DFT амплитуда низа шифрованих адреса добијених коришћењем $FF3$ -1 са три AES рунде код $Netronome LISPP$ система

заглавља.



(а) Проток пакета



(б) Кашњење у једном смеру

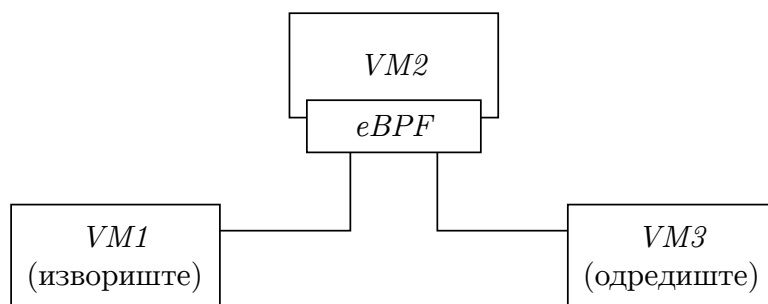
Слика 6.4: Проток пакета и кашњење у једном смеру са умањеним бројем AES рунди код $Netronome LISPP$ система

Коначно, измерено је додатно кашњење уведено услед шифровања поља заглавља пакета помоћу $Netronome LISPP$ система, када се користи мањи број AES рунди. Слика 6.4б приказује кашњење у једном смеру по пакету измерено помоћу $Socketperf$ алата. Додатно кашњење које $LISPP$ уноси креће се од $30 \mu s$ до $60 \mu s$ за $FF3$ -1 имплементације са од 3 до 10 AES рунди, респективно. Овакво додатно кашњење је занемарљиво у поређењу са кашњењима на међународним везама и одговара кашњењу проузрокованом пропагацијом сигнала између чворова удаљених свега 10 km до 20 km .

6.2 Евалуација $eBPF LISPP$ система

Перформансе $eBPF LISPP$ система су темељно евалуиране коришћењем виртуелних машина под тип 1 хипервизором [139], а не симулираног окружења. Две виртуелне машине,

са по два виртуелна процесора и 4 GB оперативне меморије, коришћене су као извориште и одредиште тестног саобраћаја. Ове две виртуелне машине су међусобно повезане преко треће виртуелне машине, са осам виртуелних процесора и 16 GB оперативне меморије, на којој је покренут *eBPF* програм, као што је приказано на слици 6.5. Све три виртуелне машине користе оперативни систем *Ubuntu 23.10* и подигнуте су на кластеру сервера који се састоји од два идентична *DELL PowerEdge R650xs* сервера. Сваки од ових сервера опремљен је са два *Intel® Xeon® Silver 4310* процесора и 128 GB *DDR4-3200* оперативне меморије. Везе између самих виртуелних машина успостављене су преко виртуелних мрежних интерфејса. Ови интерфејси су коришћени искључиво за потребе тестног окружења, како не би било мешања било каквог другог саобраћаја са тестним саобраћајем. Максимална трансмисиона јединица на свим интерфејсима је 1500 бајтова у свим експериментима, осим уколико није другачије наглашено.



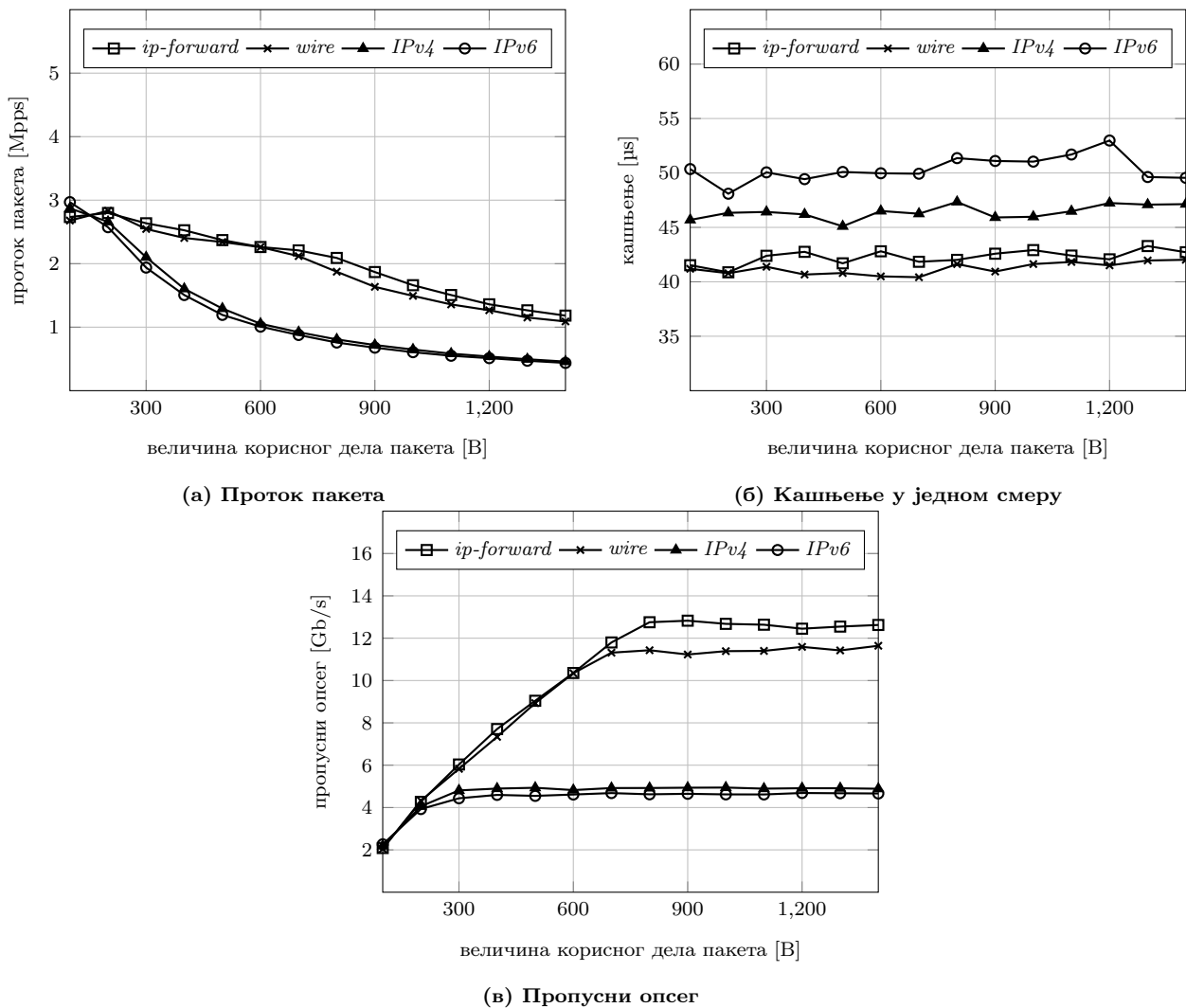
Слика 6.5: Поставка за тестирање *eBPF LISPP* система

6.2.1 Експериментална евалуација

Током процеса евалуације перформанси, слично као у случају *Netronome LISPP* система, тестни мрежни саобраћај је генерисан на изворишној *VM1* виртуелној машини и усмераван ка одредишној *VM3* виртуелној машини, пролазећи кроз *eBPF LISPP* систем унутар језгра *Linux* оперативног система *VM2* виртуелне машине. Експерименти су обухватили оба протокола мрежног слоја, *IPv4* и *IPv6*, при чему су коришћене мрежне маске /24 и /64 што имплицира да су криптографске операције примењиване на текстове дужине 24 и 80 бита, респективно. Како би се успоставила основа за поређење, за сваку од дискутованих величина извршена су два референтна мерења: *ip-forward*, који представља стандардно прослеђивање пакета од стране језгра оперативног система, и *wire*, који подразумева пролазак пакета кроз *eBPF* програм без икакве додатне обраде или модификације.

Утицај величине пакета на кључне параметре перформанси, као што су пропусни опсег, проток пакета и кашњење пакета, анализиран је кроз серију тестова за *eBPF LISPP* систем. Резултати мерења за пропусни опсег и проток пакета приказани су на сликама 6.6в и 6.6а. Мерења спроведена помоћу *PF_RING Zero Copy* алата показују да пакети тестног саобраћаја величине изнад 300 бајтова доводе до засићења везе између тестних сервера већ при пропусном опсегу од 4,5 Gb/s. У овом режиму, проток пакета је ограничен пропусном моћи везе, па се тако, на пример, максимално заузеће постиже са 1,25 милиона пакета величине 500 бајтова или приближно 0,5 милиона пакета величине 1200 бајтова у секунди. Са друге стране, тестирање са мањим пакетима тестног саобраћаја величине испод 300 бајтова открива стварне границе обраде *LISPP* система у датој поставци са *eBPF* технологијом на виртуелној машини, где је измерен максимални проток пакета од 2,97 Mpps за *IPv6* и 2,87 Mpps за *IPv4*. Проток пакета представља адекватнију метрику за процену перформанси у односу на стандардне мере пропусног опсега и у случају *eBPF LISPP* система јер се криптографске операције врше искључиво над појединим пољима

заглавља, чије су величине идентичне у сваком пакету. Што се тиче кашњења, слика 6.6б илуструје да *LISPP* систем уноси конзистентно кашњење од свега 5 μ s до 10 μ s по пакету у једном смеру, независно од његове величине, што је у складу са теоријским очекивањима изнетим у претходном пасусу.



Слика 6.6: Проток пакета, кашњење у једном смеру и пропусни опсег за пакете различитих величина код *eBPF LISPP* система

Подаци приказани на слици 6.6б такође указују да кашњење које *eBPF LISPP* уноси не зависи од величине шифрованог текста. Додатно кашњење остало је доследно за *IPv4*, са 24-битним шифрованим текстом, и за *IPv6*, са 80-битним шифрованим текстом. Овај налаз поново сугерише да је *LISPP* систем скалабилан са бројем уређаја односно величином заштићене мреже, што директно дефинише величину хост дела адресе који се шифрује. На крају, кључни параметар који дефинише границе перформанси *LISPP* система је проток пакета.

Интересантно је запазити на сликама 6.6а и 6.6в да *ip-forward* остварује незнатно већи проток пакета и пропусни опсег у поређењу са *wire*, иако *wire* подразумева пролазак пакета кроз *eBPF* програм без икакве додатне обраде или модификације. Овај феномен је последица коришћења виртуелних машина, при чему у оквиру коришћеног тип 1 хипервизора није било могуће конфигурирати *XDP native* режим рада, па је коришћен *XDP generic* режим рада за *eBPF* програме на сваком од *virtio* интерфејса [140, 141]. *XDP native* режим рада гарантује најбоље перформансе, због извршавања директно унутар драјвера мрежног уређаја, док *XDP generic* режим рада подразумева пролазак кроз део мрежног стека језгра

и скупу операцију алокације меморије за `sk_buff` структуру, што доводи до слабијих перформанси. Додатно, *wire* захтева припрему *eBPF* виртуелне машине за *eBPF* програм, што додаје изванредан временски пенал, за разлику од *ip-forward* где се користи уобичајени пут обраде пакета унутар језгра оперативног система. Из свега претходно наведеног закључује се да за потребе већег протока пакета и пропусног опсега, мора бити извршено додатно тестирање на стварним мрежним картицама или под хипервизором са подршком за *XDP native* режим рада за *virtio* интерфејсе.

6.3 Резиме резултата

Експериментална евалуација обе имплементације *LISPP* система показала је да је заштита приватности на мрежном слоју остварива уз прихватљиве перформансе. *Netronome* имплементација остварила је проток пакета од 2,16 Мррps за *IPv4* и 2,64 Мррps за *IPv6*, уз конзистентно кашњење од око 60 μ s по пакету, без смањивања броја *AES* рунди. Са друге стране, *eBPF* имплементација постигла је нешто већи проток пакета од 2,87 Мррps за *IPv4* и 2,97 Мррps за *IPv6*, уз ниже кашњење од оквирно 10 μ s по пакету. Иако *eBPF* имплементација остварује боље перформансе у погледу кашњења, треба напоменути да је тестирање извршено у *XDP generic* режиму рада на виртуелним машинама, док је *Netronome* тестиран на стварном хардверу са физичким интерфејсима. На основу претходног излагања се закључује да важи полазна хипотеза да програмабилни мрежни уређаји омогућавају реализацију заштите приватности на мрежном слоју коришћењем *FPE* уз минимално нарушавање перформанси. Кључна разлика између ове две имплементације огледа се у економским и оперативним аспектима примене. *Netronome Agilio CX* картица представља наменски хардвер чија је тржишна цена износила приближно неколико стотина евра по комаду, што повећава трошкове увођења *LISPP* система у мрежну инфраструктуру. Насупрот томе, *eBPF* технологија пружа изузетну флексибилност јер се може применити на било којој виртуелној машини унутар постојеће инфраструктуре, без потребе за набавком додатног хардвера. Ова флексибилност омогућава примену *LISPP* система на границама мреже, у облаку или било где другде где је потребна заштита приватности, уз минималне додатне трошкове и могућност динамичког скалирања према потребама.

7. Примена *LISPP* система

Један од протокола чијој је заштити посвећена највећа пажња јесте *DNS* сервис. *DNS* је кључна компонента инфраструктуре интернета задужена за превођење назива домена у њему придружену *IP* адресу. Ипак, његова сврха у пракси је далеко превазишла пуко превођење назива домена у *IP* адресе, тако да се *DNS* данас индиректно користи за многе критичне процесе у оквиру мрежа као што су балансирање саобраћаја и испорука садржаја. *DNS* сервис укључен је у готово све мрежне интеракције између корисника и бројних мрежних сервиса.

Највећи удео *DNS* саобраћаја шаље се у виду отвореног текста путем класичног *DNS* протокола, чија архитектура нема фундаменталне безбедносне функционалности нити механизме за заштиту приватности, што отвара бројне безбедносне проблеме и уноси ризик по заштиту приватности [142, 143, 144]. На пример, нападач позициониран између крајева комуникације може пресрести *DNS* одговоре како би убацио лажне *DNS* податке, чиме се жртва преусмерава на злонамерно одредиште. Слањем *DNS* упита и одговора у отвореном тексту није нарушена само безбедност, већ је у потпуности угрожена и приватност корисника [145, 146]. Увидом у садржај упита могуће је вршити анализу понашања и идентификацију корисника. Могуће је идентификовати појединачне кориснике на основу њихових упита, без обзира на повремену промену њихових *IP* адреса и коришћених уређаја, на основу следеће две чињенице [147]. Прво, иако корисници могу с времена на време динамички мењати *IP* адресе, обрасци њихових *DNS* упита биће у највећој мери доследни током коришћења једне *IP* адресе. Друго, *DNS* упити у одређеном периоду су јединствени за сваког корисника јер је њихово лично понашање на мрежи доследно и понавља се током релативно кратког периода као што су дани или недеље.

У наставку ће кроз секцију 7.1 прво бити дат преглед постојећих механизма за заштиту приватности *DNS* сервиса, заснованих на шифровању и анонимизацији. Затим ће у секцији 7.2 бити изложена примена *LISPP* система за анонимизацију у контексту *DNS* сервиса. С обзиром на важност и осетљивост *DNS* података, примена криптографских техника, као што је *LISPP*, може значајно побољшати безбедност и приватност корисника.

7.1 *DNS*

DNS [148] представља хијерархијски и дистрибуирани сервис одговоран за превођење назива домена у њему придружене информације. На пример, приликом сваког приступа веб страници, *DNS* у позадини преводи називе домена у *IP* адресе неопходне за лоцирање и приступ жељеном одредишту. *DNS* се састоји од три главне компоненте у које спадају простор назива домена и записи ресурса (енгл. *resource records*), сервери назива (енгл. *name servers*) и разрешивачи (енгл. *resolvers*).

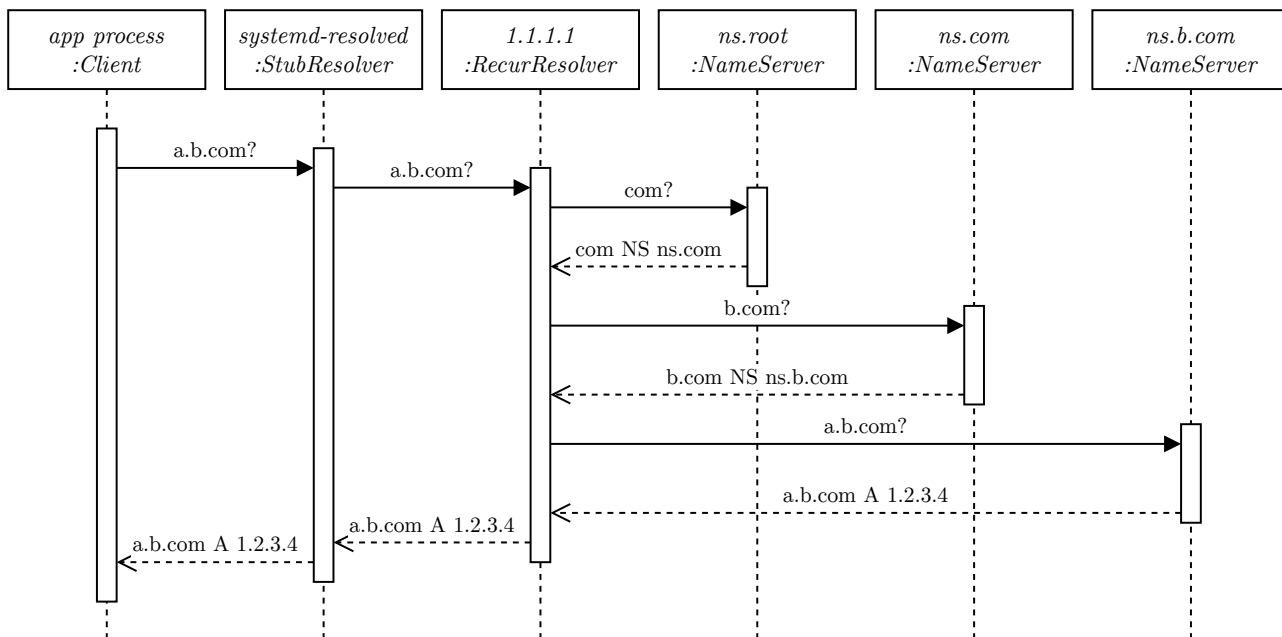
Простор назива домена јесте скуп свих назива домена представљених у виду стабла, док записи ресурса представљају податке повезане са називима домена. Хијерархија *DNS* сервиса огледа се у структури стабла простора назива домена, где је сваки назив домена јединствен и представља путању од корена стабла до одговарајућег чвора раздвојену

тачкама. Сваки чвор стабла логички именује један скуп записа ресурса, а операције упита представљају покушај дохватања специфичног типа записа из именованог скупа. Најчешћи типови записа чувани у *DNS* бази података су почетак ауторитета (*start of authority, SOA*), сервери назива (*name server, NS*), *IP* адресе (*A* и *AAAA*), алијаси назива домена (*canonical name, CNAME*), *SMTP* размењивачи поште (*mail exchanger, MX*) и показивачи за обрнуту *DNS* претрагу (*pointer, PTR*). Дакле, упит специфицира назив домена од интереса и наводи жељени тип записа придруженог датом називу домена. Са тачке гледишта корисника, *DNS* сервису се приступа путем претходно наведене процедуре најчешће реализоване као позив оперативног система ка локалном разрешивачу. Простор назива домена се састоји од тачно једног стабла и корисник може захтевати записе из било ког дела стабла.

Сервери назива у оквиру своје *DNS* базе података чувају структуру стабла простора назива домена и њима придружене записе ресурса. Појединачни сервер назива може кеширати записе за било који део стабла, али изворно поседује потпуне записе за само један део стабла и референце ка другим серверима назива који се могу користити за долазак до записа за остале делове стабла. Сервер назива зна за које тачно делове стабла поседује потпуне записе и за те делове стабла посматрани сервер назива представља ауторитет (енгл. *authority*). Ауторитативни записи су организовани у јединице које се називају зоне (енгл. *zones*), а свака зона се може аутоматски дистрибуирати серверима назива зарад постизања редундансе записа из посматране зоне. Са тачке гледишта сервера назива, *DNS* се састоји од одвојених скупова локалних записа односно зона. Задатак сервера назива, с обзиром да поседује локалне копије зона, јесте да периодично освежава своје зоне из главних копија у локалним датотекама или са других сервера назива. Паралелно са освежавањем зона, сервер назива мора и обрађивати упите пристигле од разрешивача.

Разрешивачи дохватају записе са сервера назива као одговор на упите клијената. Разрешивачи морају бити у стању да приступе најмање једном серверу назива и користе записе тог сервера назива да директно одговоре на упит клијента, или да наставе даље са слањем упита на основу референци ка другим серверима назива. Могуће је разликовати више врста разрешивача, на основу њихових функционалности, од којих су најзначајнији рекурзивни разрешивач (енгл. *recursive resolver*) и везивни разрешивач (енгл. *stub resolver*) [149]. Рекурзивни разрешивач функционише у рекурзивном моду услед чега у потпуности разрешава упит без упућивања на друге сервере назива, док везивни разрешивач ради исту ствар али не може сам да постигне целокупно разрешавање већ прослеђује упит рекурзивном разрешивачу. Разрешивач је обично системска процедура, директно доступна корисничким програмима, тако да није потребан наменски протокол између корисничког програма и разрешивача. Са тачке гледишта разрешивача, *DNS* се састоји од непознатог броја сервера назива са једним или више делова записа целокупног стабла. Разрешавање назива домена од стране *DNS* сервиса, илустровано на слици 7.1, започиње од клијента који индиректно преко свог везивног разрешивача добија жељене записе за назив домена од интереса, тако што везивни разрешивач прослеђује упит рекурзивном разрешивачу, а рекурзивни разрешивач проналази одговоре слањем упита једном или више ауторитативних сервера. С обзиром да се клијент и везивни разрешивач углавном извршавају на истом физичком уређају, у наставку ће се уместо термина везивни разрешивач користити термин клијент.

У интересу перформанси, приликом имплементације компоненте *DNS* сервиса могу бити међусобно спрегнуте. На пример, на истој физичкој машини, разрешивач и сервер назива могу делити *DNS* базу података сачињену од зона којима управља сервер назива и кеша којим управља разрешивач. Због количине садржаја и фреквенције његовог ажурирања, *DNS* база података мора бити дистрибуирана на велики број сервера назива уз интензивно локално кеширање ради унапређења перформанси. Из сличних разлога и механизам за доделу и брисање назива домена такође мора бити дистрибуиран односно делегиран. Надле-



Слика 7.1: Дијаграм секвенце разрешавања назива домена помоћу *DNS* сервиса

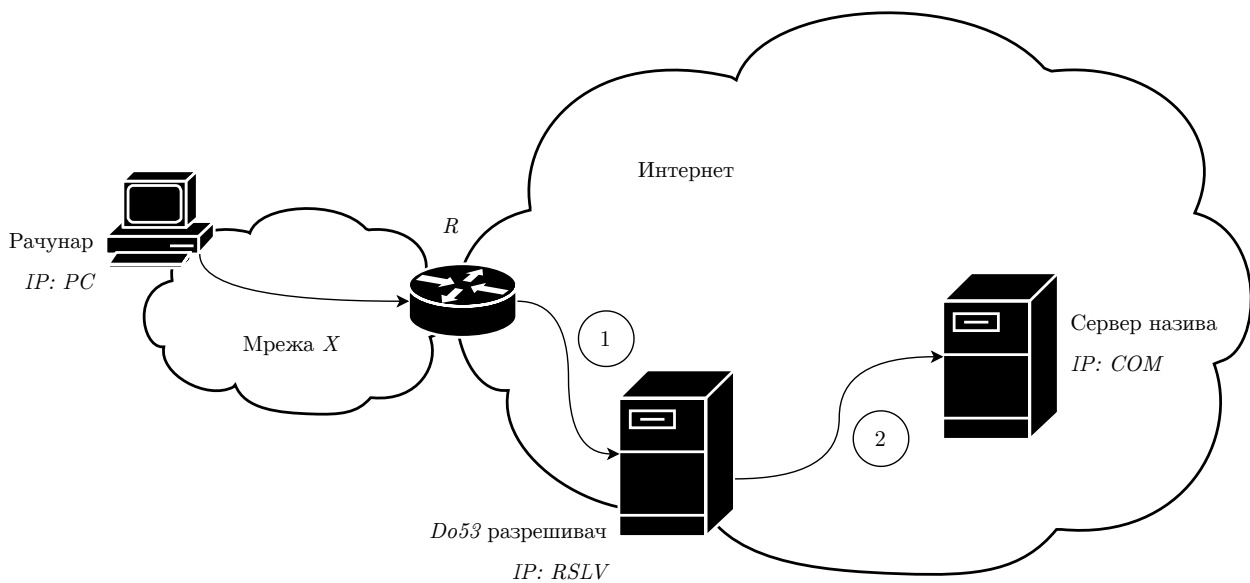
жност за додељивање назива домена *DNS* сервис делегира ка ауторитативним серверима за појединачне домене. Надлежност над поддоменима неког домена ауторитативни сервер посматраног домена може даље делегирати другим ауторитативним серверима. Оваквим дистрибуираним механизмом избегава се постојање једне велике централне *DNS* базе података чиме се обезбеђује услуга отпорнија на грешке и отказе.

DNS представља сервис без којег практично није могуће покренути ниједан други сервис, па се као такав извршава пре било ког другог сервиса. Самим тим није једноставно успоставити безбедносне механизме на тако фундаменталном сервису без угрожавања свих сервиса виших слојева. Управо зато приликом пројектовања *DNS* сервиса није вођено рачуна о приватности нити интегритету, услед чега *DNS* протокол изворно није имао никакав механизам гаранције аутентичности записа ресурса. Стога је предложена сигурносна *DNS* екстензија (енгл. *DNS Security Extension, DNSSEC*) [150] како би се корисници између осталог заштитили од напада као што је тровање *DNS* кеша (енгл. *DNS cache poisoning*). *DNSSEC* додаје могућност криптографског потписивања записа ресурса. Захваљујући томе разрешивачи могу проверити да ли примљени записи заиста потичу од власника зоне и да нису мењани током транспорта. Ауторитативни сервери, као власници зона, потписују своје записе својим приватним кључевима. Разрешивачи, користећи јавне кључеве власника зона, валидирају потписе записа добијених у склопу одговора и на тај начин осигуравају њихов интегритет и аутентичност. *DNSSEC* не осигурава тајност записа, што и даље отвара могућност увида у то шта корисник разрешава, али без његове примене нападач би могао да лажира одговоре, наводећи кориснике да посете злонамерна одредишта.

Због сложености имплементације и одржавања *DNS* сервиса, његова намена приликом пројектовања намерно није ограничена само на једну ствар. Сваки податак повезан са неким називом домена може бити сачуван у *DNS* бази података као запис ресурса одговарајућег типа. *DNS* је временом проширен како би чувао записе за друге типове података, било за аутоматску претрагу, као што су *DNSSEC* записи, или за неформалне упите као што су записи о одговорним особама (*responsible person, RP*). Као база података опште намене, *DNS* се такође користи у борби против нежељених мејлова чувањем црних листа. *DNS* база података се конвенционално чува у структурираној текстуалној датотеци званој зонска датотека, али су уобичајени и други системи база података.

7.1.1 Класични DNS (Do53)

Класични DNS сервис ослушкује упите путем UDP на резервисаном порту 53 због чега је познат као DNS преко порта 53 (енгл. *DNS over port 53, Do53*). Клијент шаље упит у отвореном тексту кроз тачно један UDP пакет, што је илустровано на слици 7.2, на који разрешивач шаље одговор такође у отвореном тексту и кроз тачно један UDP пакет. Максимална величина UDP пакета за DNS је традиционално 512 бајтова, али могу се користити и већи UDP пакети уз помоћ механизма DNS екстензија. Поред тога, коришћење UDP ограничава DNS сервис и недостатком шифровања и аутентификације на транспортном слоју. Пренос упита и одговора могућ је опционо и путем TCP [151]. Фрагментацијом дугих одговора, TCP омогућава дуже одговоре, поуздану испоруку и поновну употребу дуготрајних веза између клијената и разрешивача.



	Src IP	Dst IP	Src Port	Dst Port	DNS
1	PC	RSLV	1234	53	queries: [d.com]
	Src IP	Dst IP	Src Port	Dst Port	DNS
2	RSLV	COM	3456	53	queries: [d.com]

Слика 7.2: Принцип рада Do53 сервиса

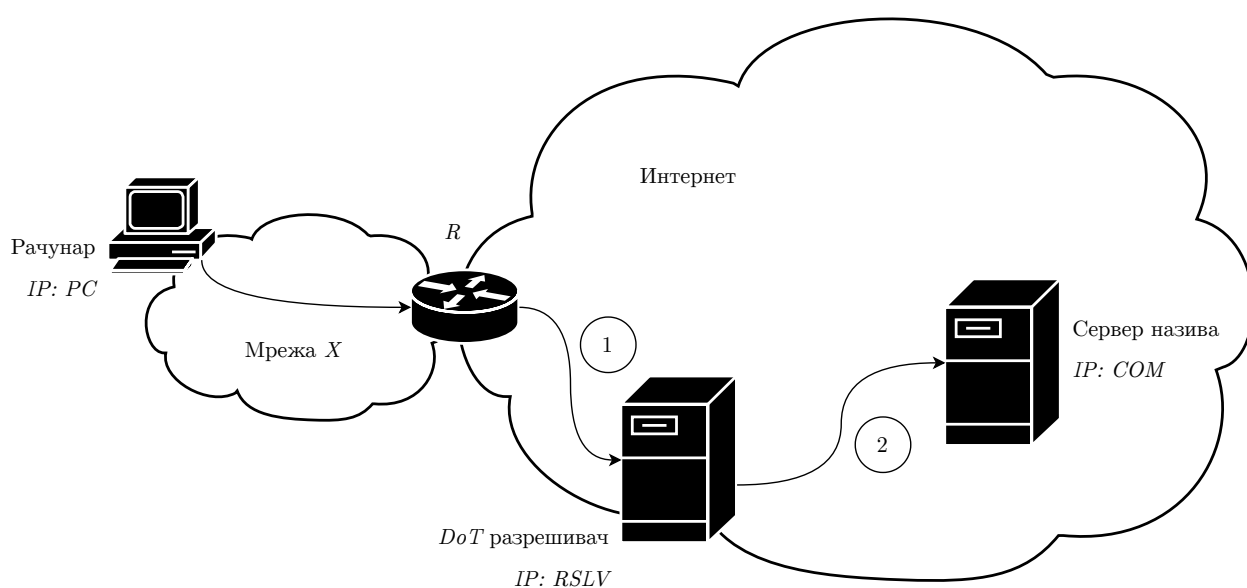
Имајући у виду да се размена Do53 порука, односно упита клијента и одговора рекурзивних разрешивача, врши у отвореном тексту, активност корисника је директно изложена посматрачима на комуникационом каналу. Додатно, ауторитет који надгледа комуникациони канал између корисника и разрешивача може једноставно препознати и цензурисати жељене Do53 поруке. Стога се, ради заштите приватности корисника, истражују и имплементирају различити алгоритми шифровања у контексту DNS сервиса како би се DNS поруке заштитиле од прислушкивања [152].

DNSCrypt [153] је један од првих напора да се реши проблем нарушене приватности код DNS сервиса. У *DNSCrypt* систему, клијент и рекурзивни разрешивач размењују своје јавне кључеве помоћу којих се затим шифрују упити и одговори. Шифровани упити и одговори преносе се на стандардан начин путем UDP или TCP у структурираном формату. Поред шифровања врши се и аутентификација клијената и разрешивача. Клијент шифрује и криптографски потписује упите, док разрешивач дешифрује и верификује потписе пре

него што одговори. *DNSECrypt* аутентификује комуникациони канал између клијента и разрешивача за разлику од *DNSSEC* који аутентификује *DNS* записе. Недостатак *DNSECrypt* система јесте изостанак формалне стандардизације и изостанак шире примене.

7.1.2 DNS преко TLS (DoT)

Како би се превазишао проблем недостатка приватности присутан код *Do53* сервиса, где се сав саобраћај шаље у отвореном тексту, предложено је неколико шема шифровања *DNS* порука ради спречавања њиховог прислушкивања. Заједничко за већину ових шема јесте да се, помоћу асиметричних криптографских алгоритама, успоставља сигурна шифрована веза између клијента и разрешивача, након чега се путем ње размењују *DNS* поруке. Једна од таквих шема јесте *DNS* преко *TLS* (енгл. *DNS over TLS, DoT*) [154] предложен 2016. године. *DoT* сервис, чији је принцип рада илустрован на слици 7.3, шифрује упите и одговоре између клијента и разрешивача користећи *TLS* протокол.



	Src IP	Dst IP	Src Port	Dst Port	DNS over TLS	
1	PC	RSLV	1234	853	TLS record	queries: [d.com]
	Src IP	Dst IP	Src Port	Dst Port	DNS	
2	RSLV	COM	3456	53	queries: [d.com]	

Слика 7.3: Принцип рада *DoT* сервиса

Клијент иницира *TLS* успостављање везе тзв. руковање (енгл. *handshake*) са разрешивачем на резервисаном порту 853 ради успостављања *TLS* везе. Упити и одговори код *DoT* сервиса размењују се директно између клијента и разрешивача, исто као и код *Do53* сервиса, при чему се размена врши преко шифроване *TLS* везе. Захваљујући шифровању упита и одговора, *DoT* сервис спречава прислушкивање и манипулацију од стране нападача позиционираног између клијента и разрешивача. Шифровање целокупне везе помоћу *TLS* протокола разликује *DoT* сервис од *DNSECrypt* система где се врши шифровање искључиво садржаја упита и одговора.

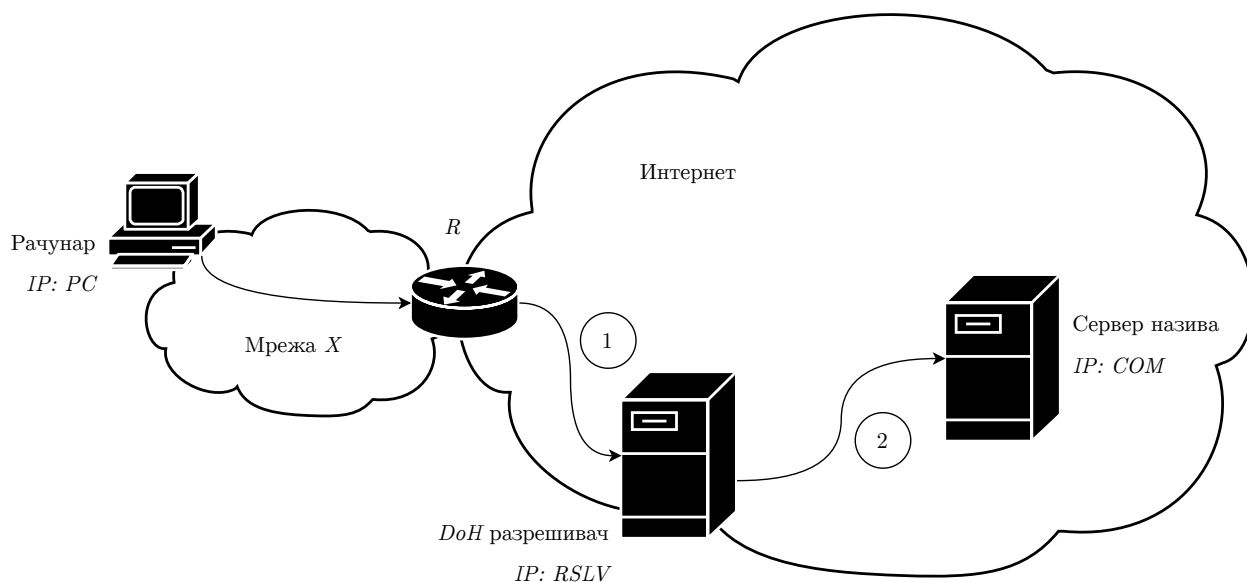
DoT сервис је применљив за било коју *DNS* трансакцију, иако је првобитно стандардизован за употребу између клијента и рекурзивног разрешивача. Накнадно је специфицирана употреба *DoT* сервиса између рекурзивних разрешивача и ауторитативних сервера, као и

имплементација трансфера зоне преко *TLS* протокола између два ауторитативна сервера. *DoT* клијенти не морају нужно слати упите директно ауторитативним серверима. Клијент се може ослонити на *DoT* разрешивач који у име клијента даље користи класичне упите, били они у отвореном или шифрованом тексту, да би коначно дошао до ауторитативних сервера. У складу са тим, *DoT* сервис не врши шифровање с краја на крај (енгл. *end-to-end*), већ врши шифровање од скока до скока (енгл. *hop-to-hop*).

DoT сервис се имплицитно заснива на инфраструктури јавних кључева (енгл. *public key infrastructure, PKI*), за успостављање сигурне шифроване *TLS* везе између клијента и разрешивача, што га чини рањивим на компромитовање сертификационих ауторитета (енгл. *certification authority, CA*). Такође, услед ослањања на наменски порт 853, *DoT* саобраћај се јасно истиче од осталог саобраћаја, тако да ауторитет који надгледа комуникациони канал између корисника и разрешивача може једноставно препознати и цензурисати *DoT* поруке. Коначно, као у случају било које друге комуникације, шифровање упита само по себи пружа заштиту приватности искључиво од посматрача, али не пружа никакве гаранције по питању анонимности када је реч о крајњој тачки шифровања односно разрешивачу којем су након дешифровања доступни сви подаци.

7.1.3 DNS преко HTTPS (DoH)

DNS преко *HTTPS* (енгл. *DNS over HTTPS, DoH*) [155], уведен 2018. године, размењује упите и одговоре путем сигурног протокола за пренос хипертекста (енгл. *hypertext transfer protocol secure, HTTPS*). Циљ *DoH* сервиса јесте заштита приватности и повећање безбедности корисника тиме што се спречава прислушкивање и манипулација *DNS* порука. Овај циљ постиже се шифровањем упита и одговора између клијента и разрешивача коришћењем *HTTPS* протокола, као што је илустровано на слици 7.4.



	Src IP	Dst IP	Src Port	Dst Port	HTTPS	
1	PC	RSLV	1234	443	TLS record	queries: [d.com]
	Src IP	Dst IP	Src Port	Dst Port	DNS	
2	RSLV	COM	3456	53	queries: [d.com]	

Слика 7.4: Принцип рада *DoH* сервиса

Клијент иницира комуникацију са разрешивачем на резервисаном порту 443 ради успостављања *HTTPS* везе. Упити и одговори код *DoH* сервиса размењују се директно између клијента и разрешивача, исто као и код *Do53* сервиса, при чему се размена врши преко шифроване *HTTPS* везе. *DoH* суштински тунелује упите и одговоре преко *HTTPS* протокола, што се своди на пренос путем *HTTP* преко *TLS* везе. Управо захваљујући слању упита и одговора преко шифроване *HTTPS* везе, *DoH* сервис спречава прислушкивање и манипулацију од стране нападача позиционираног између клијента и разрешивача.

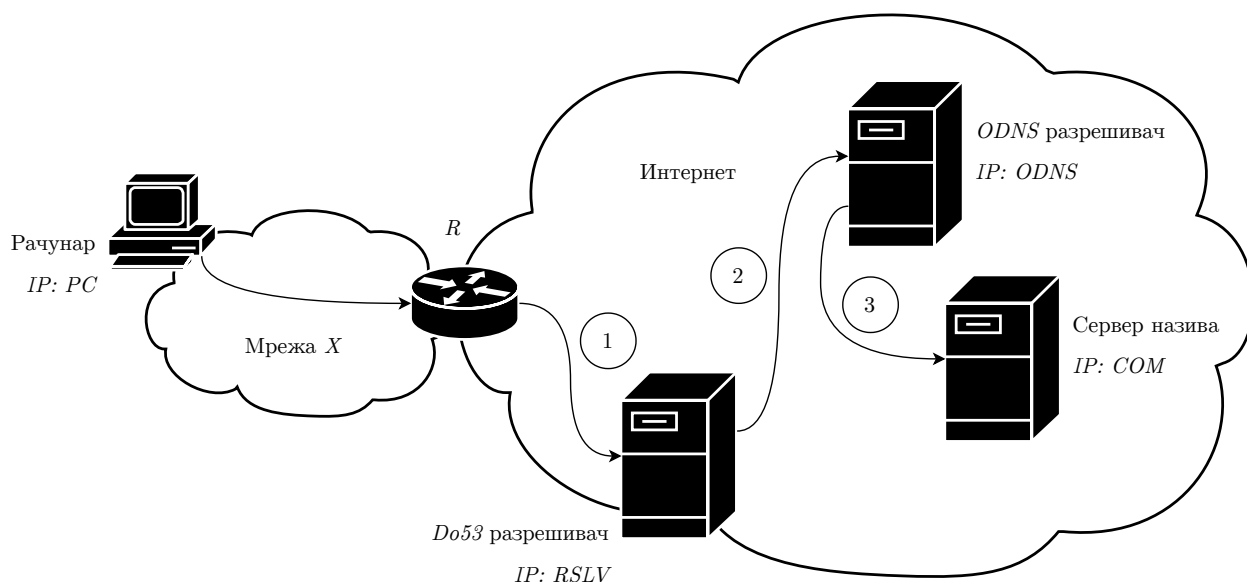
Алтернатива *DoH* сервису је *DoT* сервис који се разликује само у методама коришћеним за пренос упита и одговора. Са становишта приватности и безбедности, ниједан од ова два сервиса се не издваја као апсолутно супериоран. *DoH* у основи користи *TLS* везу као сигурни комуникациони канал, исто као и *DoT*, али се размена упита и одговора у случају *DoH* сервиса извршава путем *GET* или *POST* метода *HTTP* протокола преко тог сигурног комуникационог канала. Иако је могуће користити било коју актуелну верзију *HTTP* протокола, *HTTP/2* је препоручени минимум јер у том случају разрешивач може унапред послати (енгл. *server push*) оне записе за које предвиђа да ће клијенту бити корисни. Једна од предности *DoH* сервиса јесте то што отежава разликовање *DoH* саобраћаја од уобичајеног веб саобраћаја, услед ослањања на стандардни *HTTPS* протокол преко резервисаног порта 443, док се у случају *DoT* сервиса на основу броја порта експлицитно види да је реч о *DNS* саобраћају [156]. Такође, још једна предност *DoH* сервиса јесте уграђена подршка од стране већине веб прегледача.

DoH сервис, исто као и *DoT* сервис, уноси додатне трошкове што резултује смањењем перформанси у поређењу са *Do53* сервисом. Узрок додатних трошкова јесте употреба *TCP* протокола што захтева одржавање стања сесије на страни клијента и сервера, а само успостављање *TCP* везе може удвостручити кашњење за једну трансакцију упита и одговора. Додатно, *RTT* повезан са *TLS* руковањем уноси извесно кашњење. Како би се минимализовало кашњење повезано са успостављањем везе приликом комуникације са познатим уређајем, може се користити техника брзог успостављања *TCP* везе (енгл. *TCP Fast Open, TFO*) [157]. Поред смањења перформанси, постоји и проблем доступности сервиса јер и *DoH* и *DoT* захтевају подршку и на страни клијента и на страни разрешивача, као и пермисивну мрежну инфраструктуру на путањи између њих. Многе мреже су подешене тако да прихватају *DNS* саобраћај само преко порта 53, док неке чак прихватају *DNS* саобраћај искључиво преко *UDP* протокола.

7.1.4 Несвесни *DNS* (*ODNS*)

Ризик по заштиту приватности корисника присутан је чак и поред примене шифровања у склопу *DoT* или *DoH* сервиса. Рекурзивни разрешивачи због саме природе *DNS* сервиса увек виде садржај упита, чак иако су упити били шифровани током транспорта, тако да могу упарити упит у отвореном тексту са *IP* адресом корисника и на тај начин надгледати све његове активности. У циљу решавања описаног проблема односно ради скривања идентитета корисника, предложено је неколико шема анонимизације за скривање *IP* адресе корисника од рекурзивних разрешивача. Прва таква шема анонимизације за *DNS* сервисе јесте несвесни *DNS* (енгл. *Oblivious DNS, ODNS*) [158] предложен 2019. године. *ODNS* је пројектован да буде компатибилан са *Do53* сервисом, а његов принцип рада приказан је на слици 7.5.

ODNS уводи нови специјализован разрешивач, при чему се постојећи *Do53* рекурзивни разрешивач користи као релеј (енгл. *relay*) односно посредник за прослеђивање шифрованих упита и одговора између клијента и специјализованог *ODNS* разрешивача. Приликом прослеђивања шифрованих упита и одговора, посредник мења изворнишу *IP* адресу тако што на њено место ставља своју *IP* адресу. Дакле, између клијента и *ODNS* разрешивача се



	Src IP	Dst IP	Src Port	Dst Port	DNS
1	PC	RSLV	1234	53	queries: [x7wgham2p9kqx4.odns]
	Src IP	Dst IP	Src Port	Dst Port	DNS
2	RSLV	ODNS	3456	53	queries: [x7wgham2p9kqx4.odns]
	Src IP	Dst IP	Src Port	Dst Port	DNS
3	ODNS	COM	5678	53	queries: [d.com]

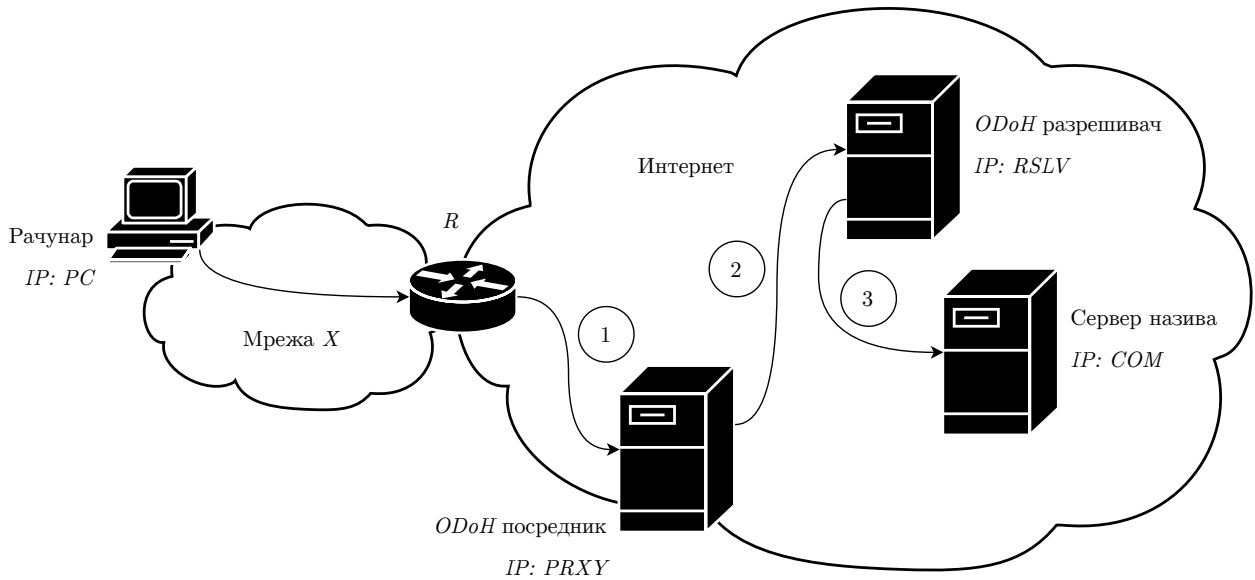
Слика 7.5: Принцип рада ODNS сервиса

логички успоставља шифровани комуникациони канал, али се шифровани упити и одговори размењују преко постојећег Do53 рекурзивног разрешивача, да би ODNS разрешивач коначно разрешавао упите у име рекурзивног разрешивача. На тај начин је IP адреса клијента сакривена од ODNS разрешивача захваљујући прослеђивању преко посредника, док је садржај упита и одговора сакривен од посредника захваљујући шифровању.

ODNS сервис уводи ауторитативни сервер за посебан .odns назив домена највишег нивоа (енгл. *top level domain*). Клијент формира упите тако што конкатенира оригинални назив домена шифрован генерисаним једнократним симетричним кључем, затим дати једнократни симетрични кључ шифрован јавним кључем .odns ауторитативног сервера и на крају .odns назив домена. Сваки тако формиран упит, преко посредника односно Do53 рекурзивног разрешивача, стиже до .odns ауторитативног сервера који заправо представља ODNS разрешивач. ODNS разрешивач прво својим приватним кључем дешифрује једнократни симетрични кључ, како би затим њиме дешифровао оригинални назив домена, чије се рекурзивно разрешавање даље извршава у отвореном тексту. Посредник, позициониран између клијента и ODNS разрешивача, управо на овај начин скрива IP адресу клијента од ODNS разрешивача који види упите у отвореном тексту. С друге стране, посредник не види упите нити одговоре у отвореном тексту захваљујући шифровању између клијента и ODNS разрешивача.

7.1.5 Несвесни DoH (ODoH)

ODNS је осмишљен као проширење за Do53 сервис, пре него што је DoH стандардизован и шире примењен у пракси. Услед тога ODNS за сваки појединачни упит прибегава шема шифровања заснованој на симетричном шифровању назива домена уз истовремено асиметрично шифровање једнократног кључа. Притом сваки упит мора бити прослеђен до ODNS разрешивача односно .odns ауторитативног сервера чије понашање одступа од стандарда. ODNS зато, иако компатибилан са Do53 сервисом уз елегантан приступ решавању проблема заштите анонимности, у извесној мери повећава RTT сваког упита. Управо је зато након распрострањења примене DoH сервиса, 2021. године дефинисан несвесни DoH (енгл. *Oblivious DoH*, ODoH) [159, 160], по узору на ODNS сервис уз извесна поједностављења како би се постигле боље перформансе. ODoH сервис суштински представља надоградњу DoH сервиса која осигурава да ниједан учесник процеса разрешавања упита није истовремено свестан и IP адресе клијента и садржаја упита, а његов принцип рада приказан је на слици 7.6.



	Src IP	Dst IP	Src Port	Dst Port		HTTPS
1	PC	PRXY	1234	443	TLS record 1	queries: [d.com]
2	PRXY	RSLV	3456	443	TLS record 2	queries: [d.com]
3	RSLV	COM	5678	53		DNS

Слика 7.6: Принцип рада ODoH сервиса

ODoH сервис задржава принцип раздвајања односно партиционисања приватности (енгл. *privacy partitioning*) [161], као код ODNS сервиса, ради очувања истог нивоа приватности клијента. Уместо Do53 рекурзивног разрешивача у функцији ODNS посредника, ODoH сервис уводи несвесног посредника (енгл. *oblivious proxy*) као наменски мрежни чвор за прослеђивање шифрованих упита и одговора између клијента и ODoH разрешивача, што директно узрокује изостанак компатибилности са Do53 сервисом. ODoH разрешивач се и логички и физички може поделити на Do53 рекурзивни разрешивач и несвесну мету (енгл.

oblivious target) чија је улога дешифровање упита, њихово прослеђивање *Do53* рекурзивном разрешивачу у отвореном тексту и шифровање одговора у повратном смеру.

Клијент сваки упит експлицитно шифрује јавним кључем несвесне мете, чиме се њихов садржај скрива од било ког посматрача укључујући и несвесног посредника. Сваки тако формиран упит, преко несвесног посредника ради скривања *IP* адресе клијента, стиже до несвесне мете. Притом се успостављају две независне *HTTPS* везе, прво између клијента и несвесног посредника, а затим између несвесног посредника и несвесне мете уз честу примену *HTTP* мултиплексирања кроз исту *TLS* везу. На овај начин, несвесни посредник иако зна *IP* адресу клијента и несвесне мете, не може да види садржај упита шифрован јавним кључем несвесне мете, док несвесна мета зна за несвесног посредника и види садржај упита, али не зна и *IP* адресу клијента. Овим се спречава повезивање *IP* адресе клијента са садржајем упита, осим у случају да несвесни посредник и несвесна мета међусобно злонамерно сарађују.

7.1.6 Анонимизовани *DNSCrypt* (*ADNSCrypt*)

Анонимизовани *DNSCrypt* (енгл. *Anonymized DNSCrypt, ADNSCrypt*) [162], дефинисан 2019. године, базира се на истом концепту као и *ODoH* сервис. Ово подразумева постојање наменског *ADNSCrypt* чвора са улогом посредника за слање упита и одговора ради скривања идентитета клијента од рекурзивног разрешивача. Клијент експлицитно бира посредника преко којег се шаљу упити и одговори шифровани помоћу *DNSCrypt* система. Посредник ефикасно скрива *IP* адресу клијента од рекурзивног разрешивача, чиме се остварује раздвајање *IP* адреса клијента од упита и одговора. Разлика између *ADNSCrypt* и *ODoH* сервиса лежи у томе што се у првом случају користи шифровање помоћу *DNSCrypt* система, док се у другом случају шифровање постиже коришћењем *HTTPS* протокола. Према томе, *ADNSCrypt* посредник путем *UDP* или *TCP* прослеђује упите и одговоре шифроване помоћу *DNSCrypt* система, док *ODoH* посредник мора да терминира *TLS* везу између клијента и разрешивача и да успостави нову *TLS* везу ради прослеђивања шифрованих упита и одговора.

7.1.7 Узајамни несвесни *DNS* (*uODNS*)

Претходно наведене шеме анонимизације, засноване на партиционисању приватности односно прослеђивању упита и одговора преко посредника ради скривања *IP* адресе клијента од разрешивача, имају слабу отпорност на колаборацију посредника и разрешивача. У случају *ODNS*, *ODoH* и *ADNSCrypt* сервиса, идентитет клијента може се у потпуности разоткрити сарадњом између посредника и разрешивача. Проблем изазван оваквом колаборацијом посебно забрињава јер посредницима и разрешивачима у највећој мери управља мали број истих великих ентитета. Уколико не верује датим великим ентитетима, а самим тим индиректно не верује ни посредницима којима они управљају, клијент потенцијално може инстанцирати властитог посредника искључиво за своје потребе. Међутим, *IP* адреса властитог посредника се излаже разрешивачу и због мале величине анонимног скупа може бити јединствено повезана са клијентом, што овај приступ са властитим посредником искључиво за своје потребе чини бескорисним.

Узајамни несвесни *DNS* (енгл. *Mutualized Oblivious DNS, uODNS*) [163], предложен 2023. године, пројектован је да заштити анонимност клијената у контексту *DNS* сервиса чак и када се суочава са неповерљивим и потенцијално колаборирајућим посредницима и разрешивачима. *uODNS* ово постиже под претпоставком да сваки клијент има бар једног поузданог посредника унутар своје мреже и да је притом спреман да тог посредника узајамно дели са другим клијентима у својој непосредној близини. Клијент прослеђује

упит до разрешивача преко датог поузданог посредника, али насумично бира још нула или више додатних посредника узајамно уступљених од других клијената. Заштита идентитета клијента постиже се управо узајамним уступањем посредника, чиме се постиже да из сваког посредника излазе упити и одговори великог броја клијената, што директно повећава анонимни скуп и скрива у маси идентитет клијената.

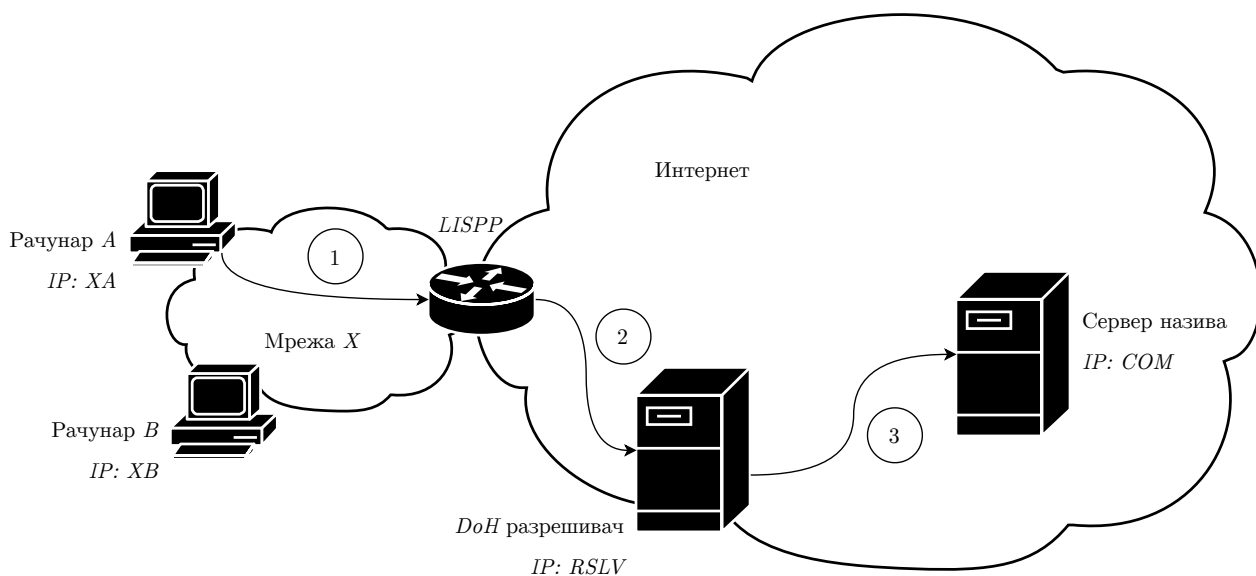
Посредници се сматрају делимично поштеним, што значи да раде исправно, али потенцијално могу надгледати садржај упита и одговора. Како би се спречило да посредници надгледају садржај упита и одговора, исто као у случају *ODNS*, *ODoH* и *ADNSCrypt* сервиса, клијент размењује упите и одговоре са разрешивачем шифроване с краја на крај. Под овом претпоставком, идентитет клијента остаје скривен од разрешивача у *uODNS* сервису чак и ако непознати подскуп посредника сарађује са разрешивачем. *uODNS* се суштински може посматрати као проширење постојећих шема анонимизације, заснованих на партиционисању приватности, при чему се проширење огледа у подршци за вишеструке посреднике и произвољни избор руте за сваки појединачни упит. У том смислу, *uODNS* усваја приступ сличан *Tor* систему, али са фокусом искључиво на *DNS* сервис.

7.2 *LISPP DoH (LDoH)*

LISPP DoH (LDoH) представља надоградњу *DoH* сервиса, без икаквих измена у принципу рада самог *DoH* сервиса, уз увођење замагљивања *IP* адреса помоћу *LISPP* система ради постизања анонимизације идентитета клијента. Анонимизација се остварује скривањем клијента у маси, односно у анонимном скупу, чиме се постиже ефекат еквивалентан по заштити приватности оном код *ODoH* сервиса. Разлика између ова два сервиса се огледа у начину скривања идентитета, јер *ODoH* сервис идентитет клијента скрива иза *IP* адресе несвесног посредника, док *LDoH* сервис идентитет клијента скрива иза *IP* адреса свих потенцијалних уређаја из локалне мреже штићене *LISPP* системом. Принцип рада *LDoH* сервиса приказан је на слици 7.7.

LDoH сервис, исто као и *ODoH* сервис, задржава принцип раздвајања односно партиционисања приватности ради очувања истог нивоа приватности клијента. *LDoH* сервис осигурава да ниједан учесник процеса разрешавања упита није истовремено свестан и *IP* адресе клијента и садржаја упита. Уместо *ODoH* несвесног посредника, *LDoH* сервис уводи *LISPP* систем као наменски мрежни чвор за прослеђивање упита и одговора шифрованих путем *HTTPS* протокола између клијента и *DoH* разрешивача уз замагљивање *IP* адресе клијента. На овај начин, *LISPP* систем иако зна *IP* адресу клијента, не може да види садржај шифрованог упита, док *DoH* разрешивач види садржај упита, али види само замагљену *IP* адресу клијента.

Поред елиминисања потребе за имплицитним поверењем у *ODoH* несвесног посредника због страха од колаборације са разрешивачем, главна мотивација за дефинисање *LDoH* сервиса, уз већ постојећи *ODoH* сервис, јесте постизање бољих перформанси. Услед примене *LISPP* система, *LDoH* сервис успоставља само једну *TLS* сесију између клијента и разрешивача, за разлику од *ODoH* сервиса где се успостављају две засебне *TLS* сесије јер несвесни посредник мора да терминира везу са клијентом пре успостављања нове везе са разрешивачем како би сакрио идентитет клијента. Како *LISPP* систем не терминира *TLS* сесију, *LDoH* не захтева додатно шифровање сваког упита јавним кључем разрешивача, што је неопходно у случају *ODoH* сервиса ради скривања садржаја упита од несвесног посредника. Такође, *LDoH* сервис не мора да чува стање за сваку сесију с обзиром да се скривање идентитета клијента заснива на пресликавању обезбеђеном криптографским алгоритмом. На основу претходно наведених аспеката, *LDoH* сервис представља ефикасније решење за заштиту приватности у контексту *DNS* сервиса у поређењу са *ODoH* сервисом



	Src IP	Dst IP	Src Port	Dst Port	HTTPS	
1	X	A	RSLV	1234	443	TLS record queries: [d.com]
	Src IP	Dst IP	Src Port	Dst Port	HTTPS	
2	X	C	RSLV	54321	443	TLS record queries: [d.com]
	Src IP	Dst IP	Src Port	Dst Port	DNS	
3	RSLV	COM	3456	53	queries: [d.com]	

Слика 7.7: Принцип рада *LDoH* сервиса

што је и потврђено експерименталном евалуацијом чији су резултати изложени у наставку.

7.2.1 Експериментална евалуација

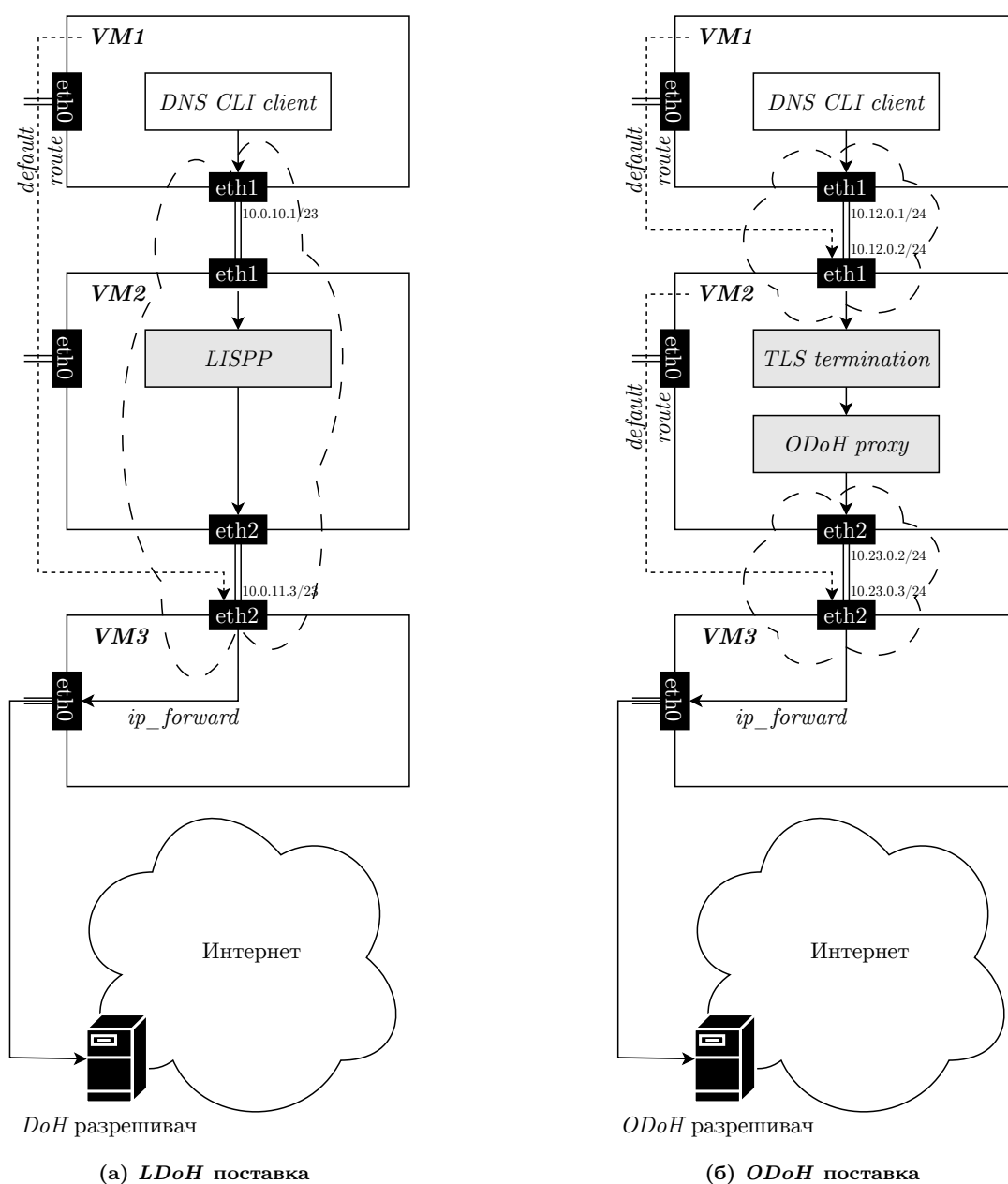
Експериментална евалуација је спроведена у истом тестном окружењу, као у случају *eBPF LISPP* система, користећи исте три виртуелне машине описане у подсекцији 6.2.1. Сходно томе, у оквиру експерименталне евалуације за *LDoH* сервис коришћен је управо *eBPF LISPP* систем. У наставку су детаљније описане специфичности поставке за тестирање *LDoH* и *ODoH* сервиса, приказане на слици 7.8, како би се јасно уочиле разлике у погледу тока саобраћаја и ангажованих компоненти.

Као *DNS* клијент у оба случаја коришћен је алат *q* [164]. Реч је о робусном *DNS* клијенту из командне линије, развијеном у програмском језику *Golang*, пројектованом за извршавање упита постижући притом високе перформансе. Алат изворно подржава широк спектар *DNS* сервиса, укључујући *Do53* сервис, као и варијанте попут *DoT*, *DoH* и *ODoH* сервиса. Због своје брзине, флексибилности и прецизности приликом мерења времена одзива, алат *q* представља адекватан избор за генерисање саобраћаја у овом истраживању.

У случају *LDoH* сервиса, чија је поставка илустрована на слици 7.8а, мрежни интерфејси *VM1* и *VM3* виртуелне машине припадају истој локалној мрежи, док су физички повезани преко *VM2* виртуелне машине на којој је покренут *eBPF LISPP* систем. На *VM1* виртуелној машини је конфигурирана подразумевана рута ка *VM3* виртуелној машини, што значи да ће сав саобраћај проћи кроз *eBPF LISPP* систем на *VM2* виртуелној машини. На овај начин се пресећу сви упити и врши замагљивање *IP* адреса клијента транспарентно за остале

учеснике у комуникацији. Саобраћај се након обраде прослеђује до *VM3* виртуелне машине, одакле се само даље усмерава ка интернету односно крајњем *Cloudflare one.one.one.one* разрешивачу.

С друге стране, у случају *ODoH* сервиса приказаног на слици 7.8б, на средњој *VM2* виртуелној машини покренут је сервер за терминацију *TLS* сесија и *ODoH* несвесни посредник. За реализацију посредничког слоја коришћени су алати *Caddy* сервер [165] за терминарање *TLS* сесије и *MODoH* [166] као имплементација *ODoH* сервера. На *VM1* виртуелној машини је конфигурисана подразумевана рута управо ка *VM2* виртуелној машини, како би саобраћај прво прошао кроз терминацију *TLS* сесије, а затим био прослеђен од стране *ODoH* несвесног посредника. На *VM2* виртуелној машини је конфигурисана подразумевана рута ка *VM3* виртуелној машини чији је задатак да саобраћај, сада већ прослеђен од стране несвесног посредника, само даље рутира ка интернету односно *odoh.cloudflare-dns.com* несвесној мети што је исти сервис као и *one.one.one.one* [167] чиме се гарантују истоветни услови.



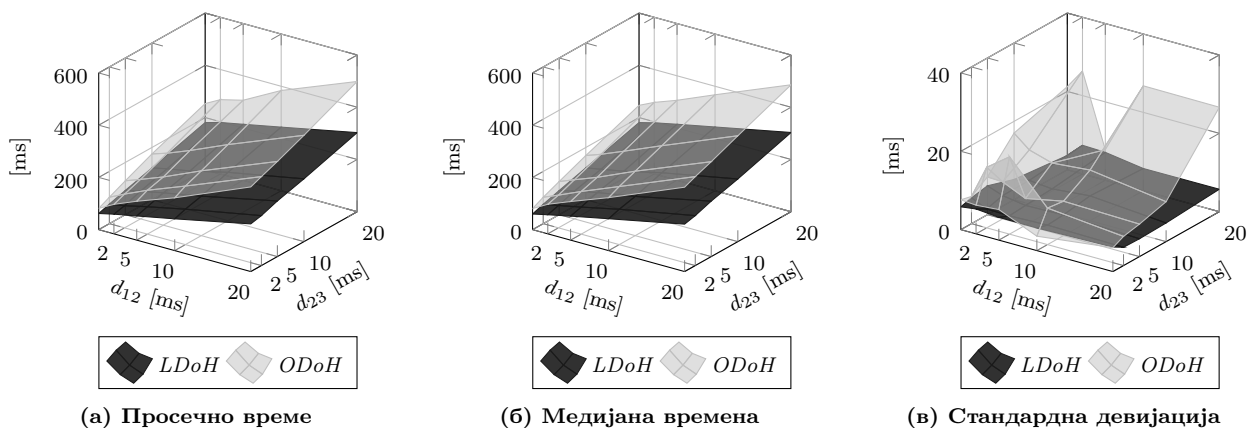
Слика 7.8: Поставка за тестирање *LDoH* и *ODoH* система

Експериментална евалуација се заснива на мерењу *RTT* за разрешење назива домена.

Као тестни узорак коришћено је првих десет најпопуларнијих назива домена са *Tranco* листе [168], која представља референтну листу веб страница ранжираних по популарности, отпорну на манипулације. Како би се добили статистички релевантни подаци, разрешење је вршено по 30 пута за сваки од одабраних назива домена. Пре самог почетка мерења *RTT* потребног за разрешење одређеног назива домена, прво је извршено његово иницијално разрешење. Овим кораком се осигурава да крајњи разрешивач рекурзивно дохвати и кешира неопходне записе ресурса, чиме се елиминишу варијације у времену одзива које би настале услед рекурзивних упита ка ауторитативним серверима. Сва наредна разрешења током мерења су усмерена ка истом разрешивачу, идентификованим његовом *IP* адресом, како би услови мерења били конзистентни.

Додатно, мерења су вршена у условима вештачки уведеног кашњења на мрежним интерфејсима *VM1* и *VM3* виртуелних машина. Вредности кашњења на сваком од интерфејса су независно вариране између 0, 2, 5, 10 и 20 милисекунди, чиме се симулира различита географска удаљеност између мрежних чворова, односно између клијента и посредника, као и између посредника и разрешивача. Вештачко увођење кашњења постигнуто је употребом уграђеног алата *Linux* оперативног система за контролу саобраћаја, конкретно коришћењем модула за емулацију мрежа (енгл. *Network Emulator, NetEm*).

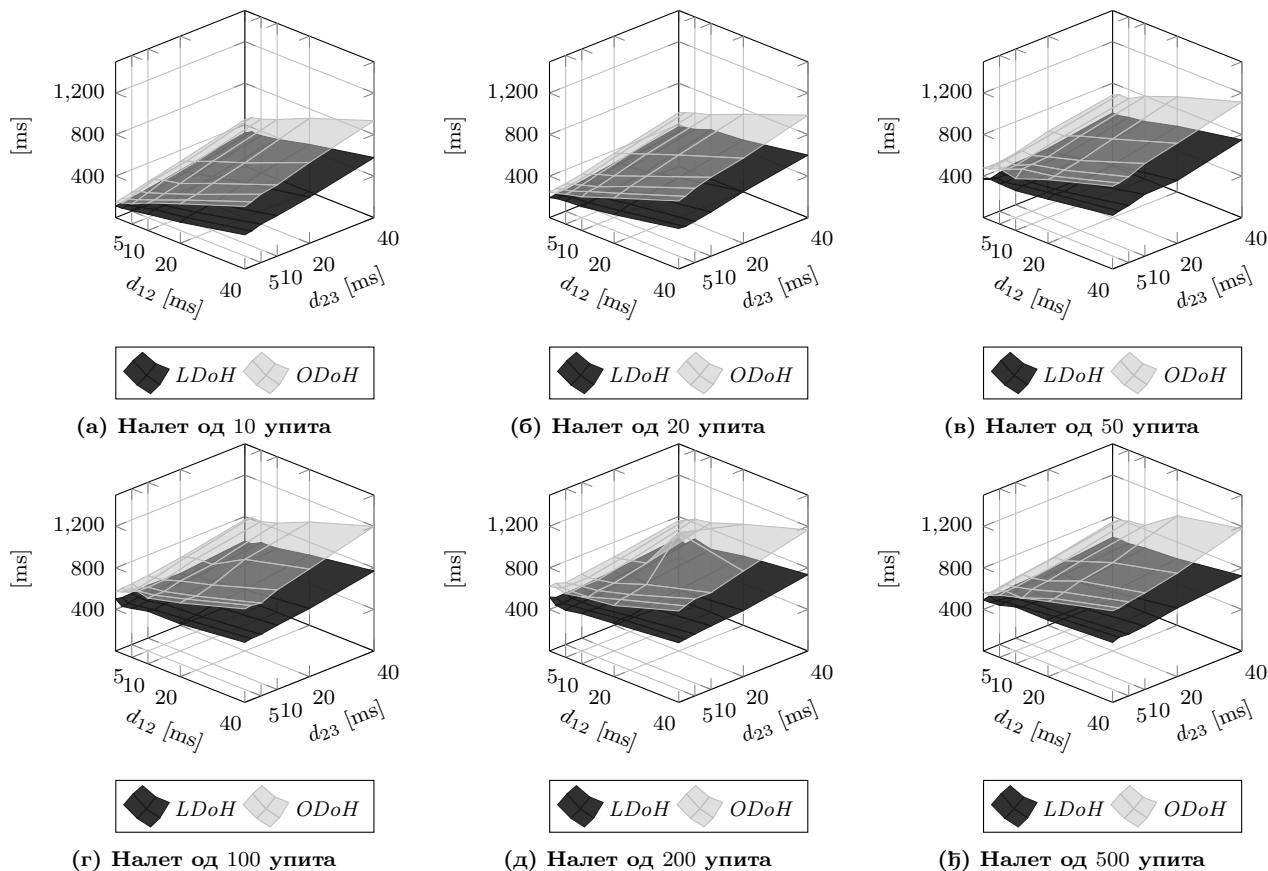
Резултати мерења *RTT* потребног за изоловано разрешење једног назива домена, у зависности од вредности вештачки уведеног кашњења, приказани су на слици 7.9. Слика обухвата три дијаграма за приказ просечног *RTT*, медијане *RTT*, као и стандардне девијације. На свим дијаграмима се јасно уочава да се површ која представља измерено време *LDoH* сервиса константно налази испод површи која одговара *ODoH* сервису, што указује на мање кашњење у свим мереним сценаријима. Анализом извршених мерења утврђено је да, у поређењу са *ODoH* сервисом, *LDoH* сервис остварује боље перформансе у опсегу од 25% у случају без вештачки уведеног кашњења, па све до 45% у случају максималних вредности вештачки уведеног кашњења. Оваква разлика у перформансама директна је последица елиминисања асиметричног шифровања и ефикаснијег руковања везама код *LDoH* приступа, који избегава успостављање двоструких *TLS* сесија.



Слика 7.9: Поређење *LDoH* и *ODoH* система у погледу *RTT* потребног за изоловано разрешење једног назива домена зависно од кашњења унетих између клијента и посредника (d_{12}) и између посредника и разрешивача (d_{23})

Резултати мерења *RTT* потребног за разрешење једног назива домена приликом налета упита, у зависности од вредности вештачки уведеног кашњења, приказани су на слици 7.10. Слика обухвата шест дијаграма за приказ просечног *RTT* у случају налета од 10, 20, 50, 100, 200 и 500 истовремених упита. На свим дијаграмима се јасно уочава да се површ која представља измерено време *LDoH* сервиса константно налази испод површи која одговара *ODoH* сервису, што указује на мање кашњење у свим мереним сценаријима. Анализом извршених мерења утврђено је да, у поређењу са *ODoH* сервисом, *LDoH* сервис остварује

боље перформансе у опсегу од 25 % у случају највећег оптерећења од 500 истовремених упита, па све до 40 % у случају оптерећења од 10 истовремених упита. Може се закључити да важи полазна хипотеза да је реализација заштите приватности на мрежном слоју коришћењем *FPE* ефикаснија од постојећих механизма за заштиту приватности.



Слика 7.10: Поређење *LDoH* и *ODoH* система у погледу просечног *RTT* потребног за разрешење једног назива домена приликом налета упита, а зависно од кашњења унетих између клијента и посредника (d_{12}) и између посредника и разрешивача (d_{23})

7.3 Промена криптографског материјала

Дискусија у секцији 4.6 пружа кратки осврт на стратегију за освежавање односно промену криптографског материјала. Идеалан тренутак за ротацију кључева је онај када у систему нема активних сесија, чиме се избегава потреба за сложеним механизмима одржавања стања или паралелним коришћењем старог и новог кључа, а уједно се елиминише ризик од прекида активних веза и губитка пакета. Док се општа примена *LISPP* система мора на конкретан начин суочити са изазовима дуготрајних *TCP* сесија, специфичан случај примене *LISPP* система искључиво за потребе заштите приватности *DNS* саобраћаја, као што је то случај код *LDoH* сервиса, карактеришу трансакције релативно кратког трајања. Управо кратко трајање трансакција отвара могућност да се приликом рада система појаве релативно чести и довољно дугачки временски интервали у којима неће бити ниједног активног разрешавања назива домена. С обзиром на природу *DNS* сервиса, анализа лога реалног *DNS* саобраћаја може пружити увид у постојање периода неактивности или смањеног интензитета саобраћаја, што би оправдало примену једноставнијих стратегија замене кључева без бојазни од значајног утицаја на квалитет сервиса. У циљу испитивања изводљивости оваквих стратегија, у наставку је извршена анализа *DNS* лога реалног мрежног саобраћаја.

DNS лог је забележен на *DNS* серверу Рачунарског центра Универзитета у Београду. Ово је један од *DNS* сервера који подржавају рад Академске мреже Србије и служи за то да обради упите корисника из школа у Србији. Бележење записа је трајало од 6:00¹ пре подне до 21:00 час на дан 28. маја 2024. године. У питању је уторак, радни дан у школама у току школске године, а циљ је био да се забележи статистика броја и разрешавања *DNS* имена домена током периода који покрива комплетан радни дан у школама, када је оптерећење *DNS* сервера највеће. У наведеном периоду је забележено укупно 24 659 821 записа. Пакете свих ових записа *LDoH* сервис може да обради за нешто мање од 10 секунди, што сугерише да је *LDoH* сервис спреман за далеко већа оптерећења или да би могао да ради и са скромнијим ресурсима. Независно од тога, свакако је урађена даља анализа лога у циљу утврђивања постојања периода неактивности или смањеног интензитета мрежног саобраћаја који би омогућили примену једноставнијих стратегија замене кључева.

Лог се састоји од записа који представљају два кључна догађаја приликом разрешавања *DNS* имена домена: упит разрешивача (енгл. *resolver query, RQ*) и одговор разрешивачу (енгл. *resolver response, RR*). У питању су упити послати од стране посматраног *DNS* сервера, према другим *DNS* серверима, са циљем добијања разрешења неког имена домена рекурзивним упитом, што представља најдужу операцију *DNS* сервера. Резолуција часовника коришћеног за бележење времена догађаја је једна милисекунда. Како школе користе сервис веб посредника, сви упити су стизали управо преко њега, чиме је избегнуто директно нарушавање приватности појединачних корисника.

У првој фази анализе лога је извршено чишћење података. Забележен је релативно велики број неразрешених упита, конкретно 1 017 660. Ово је последица различитих догађаја и ситуација током рада *DNS* сервера. У неким случајевима *DNS* сервер је за разрешење једног имена домена слао више од једног упита, а само је онај упит који је дао коначни одговор забележен као одговор у оквиру лога, док евентуални каснији одговори нису забележени што даје утисак да постоје упити на које није одговорено. Овакав пример је показан на листингу В.1 за назив домена `eu-auth2.samsungosp.com` када су послата два упита у размаку од 50 ms на два различита *DNS* сервера. Одговор добијен од једног *DNS* сервера представља разрешење овог имена домена, док евентуални одговор од другог *DNS* сервера није забележен у логу јер је приспећем првог одговора разрешење овог имена домена завршено. Други пример неразрешених упита је приказан на листингу В.2. Овде се види низ упита послатих од стране посматраног *DNS* сервера за разрешење назива домена `mc1b-gcp.nimbus.bitdefender.net`. Како одговори нису стизали, уочава се да је посматрани *DNS* сервер понављао слање упита на сваких 800 ms још 14 пута, да их је слао на различите адресе *DNS* сервера, све док коначно није стигао одговор.

Табела 7.1: Статистички параметри времена разрешења и броја активних сесија

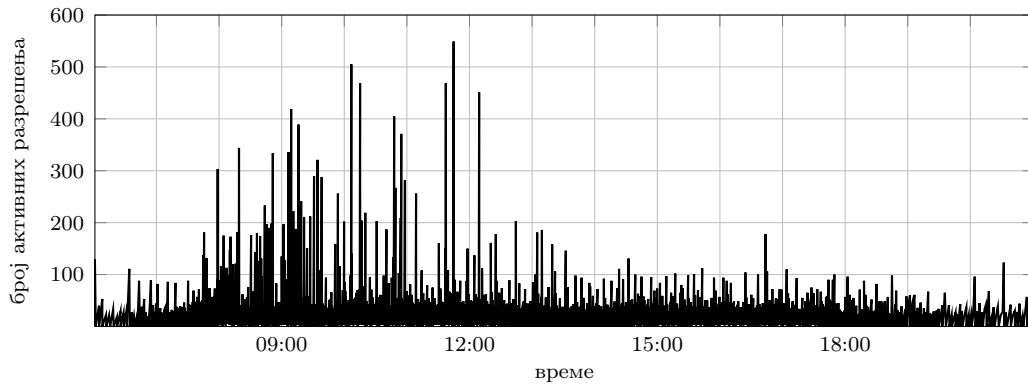
Мере	Вредност
Средње време разрешења	41,74 ms
Медијана времена разрешења	28,00 ms
Стандардна девијација времена разрешења	54,98 ms
Средњи број активних сесија	12,99
Медијана броја активних сесија	12
Максималан број активних сесија	549
Стандардна девијација броја активних сесија	12,28

У даљој анализи су коришћени само упити са пристиглим и забележеним одговором. Ово

¹Двогачка уместо тачке је коришћена у својству сепаратора, иако није правописно исправно, јер се тиме добија устаљени формат записа времена посебно када у оквиру времена фигуришу секунде и милисекунде.

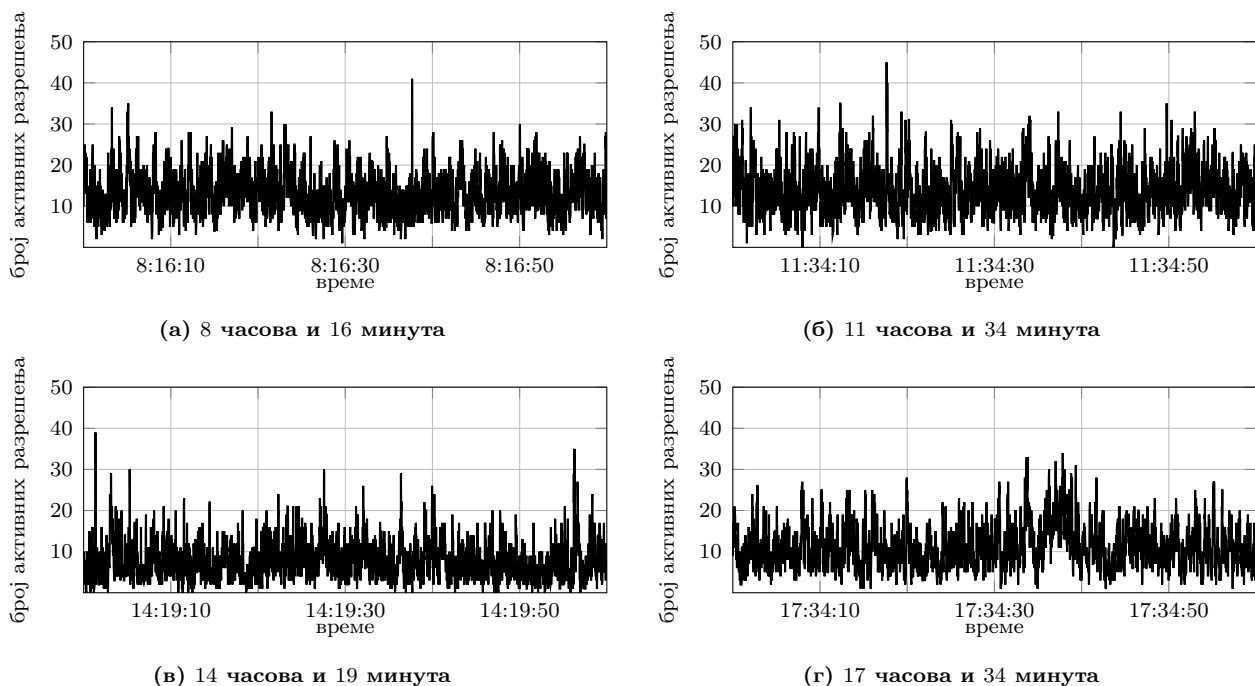
је у складу са чињеницом да су за анализу стратегије промене криптографског материјала од интереса времена разрешавања упита и број активних упита. Табела 7.1 показује основне статистичке параметре времена разрешења и броја активних сесија.

Слика 7.11 показује број истовремено активних разрешења односно оних у периоду између упита и одговора. Највећи број активних разрешења је у периоду закључно са 13:40 часова посматраног дана. Ово је очекивано јер ће највећи број упита бити генерисан у уобичајено радно време школа у преподневној смени.



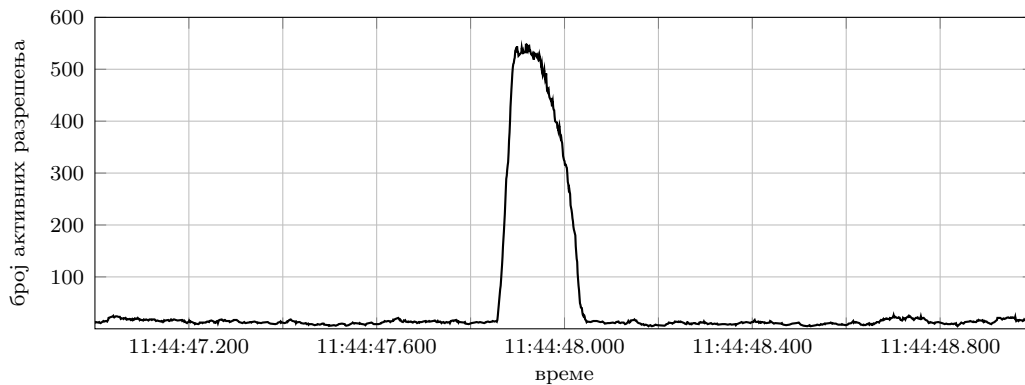
Слика 7.11: Број активних разрешења током читавог дана

Број активних разрешења односно сесија, који у неким периодима дана прелази и пет стотина, сугерише да би промена криптографског материјала у било ком тренутку угрожила најмање стотинак активних разрешења. Међутим, детаљнија анализа појединих временских интервала и вршних вредности показују да је број активних разрешења у већини интервала значајно мањи, а да су тренуци са вршним вредностима релативно ретки. Слика 7.12 показује како изгледа број активних разрешења у појединим тренуцима током радног времена и након њега. Као што се са слике уочава, број активних разрешења јесте нешто виши током радног времена, али је најчешће близак средњој вредности, изузев у тренуцима када је долазило до вршних вредности.



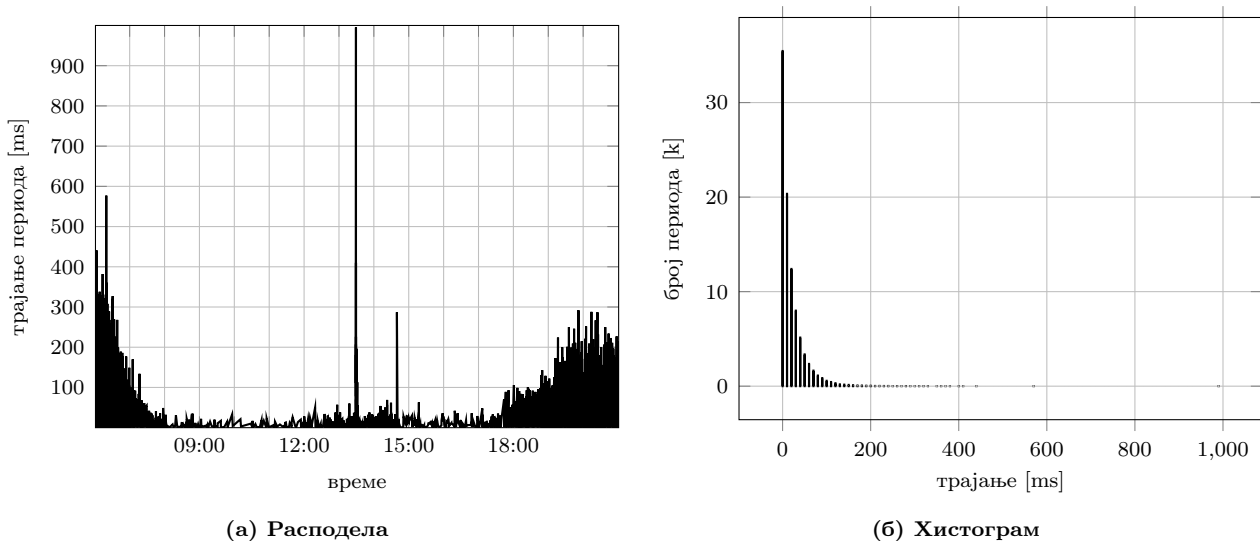
Слика 7.12: Број активних разрешења током различитих периода дана

Слика 7.13 показује период од две секунде између 11:44:47.000 и 11:44:49.000 часова када је дошло до максималне вредности од 549 активних сесија. Као што може да се види ова вршна вредност је настала нагло и трајала је мање од 200 ms. И пре и после овог догађаја, број истовремених сесија је био сличан као на претходним сликама. Детаљним прегледом је утврђено да је реч о комуникацији *ESET* антивирусног софтвера са централном локацијом која се врши легитимним *DNS* експилтрацијама и слањем великог броја упита у кратком временском периоду.



Слика 7.13: Број активних разрешења током вршне вредности

Урађена је и додатна анализа којом је установљено да је током времена бележења овог лога забележено укупно 92 958 интервала без иједне активне сесије. Ови интервали су били релативно кратки, са средњом вредношћу од 23,498 ms и медијаном од 14,0 ms. Њихова расподела и хистограм су приказани на слици 7.14. Такође, у периоду највећег оптерећења између 8:00 и 15:00 часова било је мање од 2000 тренутака када је број активних сесија падао на нулу, док је већина осталих интервала била ван периода радног времена и највећег коришћења.



Слика 7.14: Периоди без активних разрешења

Главни фактор приликом разматрања стратегије за промену криптографског материјала у контексту *eBPF LISPP* система у оквиру *LDoH* сервиса, поред анализе образаца *DNS* саобраћаја, јесте и само време потребно за техничку реализацију замене активног кључа. С обзиром да је пројектни циљ *LDoH* сервиса минимизација прекида у раду, неопходно је сагледати колико дуго траје критични интервал током којег се врши сама замена, јер управо тај интервал дефинише потенцијални ризик по активне сесије. Могуће је разликовати два

основна приступа реализацији замене кључева зависно од тога да ли се криптографски материјал смешта у сам *eBPF* програм или чува у *eBPF* мапама.

Први приступ подразумева да је криптографски материјал уграђен у сам изворни код *eBPF* програма. Услед тога приликом сваке замене кључева неопходно је нови *eBPF* програм поново превести, верификовати, учитати у језгро оперативног система и коначно закачити на одговарајући прикључак. Иако превођење, верификација и учитавање могу захтевати незанемарљиво време, ови кораци не представљају проблем јер се могу извршити унапред, као припремна фаза за промену криптографског материјала, без икаквог утицаја на обраду текућег саобраћаја. Стога је једино битно време потребно да се већ учитан *eBPF* програм са новим кључевима закачи на одговарајући прикључак. Ово време се типично мери у микросекундама до неколико милисекунди [169, 170], а зависи од типа прикључка и тренутног оптерећења оперативног система.

Други, флексибилнији приступ, подразумева смештање криптографског материјала у *eBPF* мапе, чиме се подаци одвајају од логике *eBPF* програма. У овом случају, *eBPF* програм чита кључеве из мапе, а процес замене кључева своди се на једноставно ажурирање садржаја мапе. Ова операција је атомична са становишта извршавањем *eBPF* програма и временски је изузетно ефикасна. Време потребно за ажурирање *eBPF* мапе зависи од величине податка и стања кеша, али се у сваком случају ради о кашњењу реда величине неколико микросекунди [171]. С обзиром на изузетно кратко трајање операције ажурирања мапе, овај приступ омогућава замену кључева готово тренутно, без потребе за поновним инстанцирањем самог програма.

На основу анализе *DNS* лога са реалним саобраћајем и процене времена потребног за техничку реализацију промене криптографског материјала, могуће је дефинисати оптималну стратегију за ротацију кључева у оквиру *LDoH* сервиса заснованог на *eBPF LISPP* систему. Узимајући у обзир процену да просечан корисник интернета генерише око 5000 упита дневно [172], анализирани *DNS* лог, који обухвата близу 25 милиона записа, грубо одговара саобраћају који би генерисала организација од приближно 2400 активних корисника. Овакав обим саобраћаја упоредив је са мрежама средњих предузећа, банака или факултета, што резултате чини релевантним за ширу примену. Прва и најфлексибилнија стратегија подразумевала би фреквентну замену кључева у произвољним временским тренуцима. Како *DNS* клијенти поседују робусне механизме за поновно слање упита у случају изостанка одговора по истеку предефинисаног времена, а број погођених активних сесија би чак и током дана био релативно мали, потенцијални кратки прекиди изазвани заменом кључева остали би неприметни за крајње кориснике. Друга, конзервативнија опција, подразумевала би ротацију кључева у унапред дефинисаним терминима ван периода интензивног саобраћаја, на пример у раним јутарњим часовима, чиме се вероватноћа прекида активних сесија значајно смањује. Коначно, трећа опција би подразумевала адаптивни приступ уз праћење броја активних сесија и вршење замене тек када тај број падне на нулу, али би овај приступ увео непотребну сложеност праћења стања која није у духу *LISPP* архитектуре. Стога се, због своје једноставности и занемарљивог утицаја на квалитет сервиса, прва стратегија периодичне замене намеће као адекватно решење.

8. Закључак

Масовно прикупљање и анализа података о личности поставили су заштиту приватности као једну од кључних истраживачких тема. Иако су корисници данас свеснији праћења путем веб колачића и апликација, често се занемарује чињеница да значајан део информација неприметно цури на мрежном слоју. Анализа заглавља мрежних пакета, а пре свега *IP* адреса, омогућава нападачима прецизно профилисање корисника, откривање њихове географске локације и праћење навика без њиховог знања. Управо из тог разлога, *IP* адресе су у оквиру важећих регулатива препознате као информације које омогућавају идентификацију корисника, што захтева адекватне техничке мере њихове заштите без нарушавања функционисања интернета.

Докторска дисертација је анализирала постојеће механизме заштите приватности на мрежном слоју, уз критички осврт на њихове мане и ограничења. Постојећа решења базирају се на шифровању осетљивих поља заглавља пакета користећи класичне алгоритме за блоковско шифровање, што се показало као успешан приступ прикривању *IP* адреса и генерално заштити приватности корисника на мрежном слоју. Овим је уједно потврђена полазна хипотеза да је успостављање корелације између пакета посматраног корисника могуће отежати замагљивањем делова заглавља пакета. Међутим, услед ослањања на блоковско шифровање постојећа решења захтевају или операције које чувају стање или превођење из *IPv4* у *IPv6* пакете, што доводи до значајних проблема приликом њихове имплементације и коришћења. Може се закључити да важи и полазна хипотеза да постојећи механизми заштите приватности на мрежном слоју имају проблеме у погледу перформанси услед интензивне примене алгоритама шифровања или чувања стања током рада. Узрок проблема лежи у природи симетричних алгоритама шифровања који нису пројектовани за рад са пољима заглавља пакета произвољне дужине и формата. Како поља заглавља пакета услед својих разноликих дужина нису поравната са величином блока код блоковских шифара или границом бајтова код проточних шифара, постојећа решења прибегавају проширивању отвореног текста што захтева додатни простор за складиштење датог проширења и индиректно узрокује увођење нових заглавља или протокола. Ова констатација је у сагласности са полазном хипотезом да употреба блоковских алгоритама шифровања у циљу замагљивања делова заглавља пакета није прикладна јер није могуће постићи довољно фину грануларност замагљивања, што доводи до нарушавања стандардног формата тих заглавља или до потребе за сложеним праћењем стања активних сесија.

Управо зато је истражена могућност примене *FPE* алгоритама за заштиту приватности на мрежном слоју. *FPE* алгоритми шифрују отворени текст произвољне дужине, без проширивања, омогућавајући тиме замену поља заглавља произвољног протокола шифрованим текстом исте величине. Захваљујући томе, *FPE* алгоритми представљају идеално решење за замагљивање *IP* адреса и портова у заглављима пакета, без нарушавања њиховог стандардног формата. У оквиру докторске дисертације показано је да се применом *FPE* алгоритма постиже равномерна расподела шифрованих изворишних *IP* адреса за различите вредности изворишних портова. Овим је и потврђена полазна хипотеза да се *FPE* алгоритми могу ефикасно користити за заштиту приватности на мрежном слоју и спречавање профилисања корисника.

Предложени *LISPP* систем, заснован на *FF3-1 FPE* алгоритму, може у потпуности транспарентно да замагли изворишне *IP* адресе и портове, и за *IPv4* и за *IPv6*, уз минимално додатно кашњење у једном смеру. *LISPP* систем је имплементиран за програмабилну мрежну картицу *Netronome Agilio CX*. Постоје три различите имплементације, у распону од употребе искључиво *P4* језика до комбинације *P4* и *Micro-C* језика, у циљу истраживања перформанси добијених за преносиви *P4* изворни код, као и његове зависности од специфичне хардверске конфигурације. Експериментална евалуација све три имплементације открила је субоптималне перформансе *P4* кода на циљној програмабилној мрежној картици. Ово указује да се процес превођења *P4* изворног кода за специфичне хардверске архитектуре може осетно побољшати, нарочито када је реч о процесорски захтевним апликацијама, као што су нови криптографски алгоритми без хардверског убрзања. Иако је функционалност *P4* кода иста на различитим циљним платформама, његове перформансе су далеко од оптималних, а притом не постоје аутоматизоване опције за оптимизацију. Ипак, *Netronome LISPP* систем у имплементацији заснованој у највећој мери на *Micro-C* језику остварује проток пакета од 2,16 Мррps за *IPv4* и 2,64 Мррps за *IPv6*, уз конзистентно кашњење од око 60 μ s по пакету, у сценарију без смањивања броја *AES* рунди. Према томе, перформансе на програмабилној мрежној картици у стварним мрежним окружењима су довољно добре за практичну употребу у продукционим мрежама, што је у сагласности са полазном хипотезом да програмабилни мрежни уређаји омогућавају реализацију заштите приватности на мрежном слоју коришћењем *FPE* алгоритама уз минимално нарушавање перформанси.

Поред имплементације за програмабилну мрежну картицу *Netronome Agilio CX*, *LISPP* систем је имплементиран и за *eBPF* технологију како би се елиминисала потреба за наменским мрежним уређајима. Без предуслова у виду додатног специјализованог хардвера, стављање *LISPP* система у практичну употребу у оквиру постојеће мрежне инфраструктуре постаје знатно једноставније. *eBPF* технологија представља једноставну општенаменску виртуелну машину унутар језгра *Linux* оперативног система, а извршавањем *eBPF* програма унутар језгра избегавају се режијски трошкови узроковани учесталим системским позивима. Флексибилност при дефинисању логике прослеђивања и обраде пакета је гарантована употребом општенаменских програмских језика. Међутим, *eBPF* извршни модел подразумева извесна ограничења, као што су ограничен скуп доступних операција и формална верификација програма, што утиче на приступ имплементацији сложених криптографских алгоритама. У комбинацији са *XDP* прикључком, захваљујући којем обрада пакета заобилази највећи део сложеног мрежног стека унутар језгра, *eBPF LISPP* систем остварује проток пакета од 2,87 Мррps за *IPv4* и 2,97 Мррps за *IPv6*, уз кашњење од оквирно 10 μ s по пакету, у сценарију извршавања под тип 1 хипервизором на серверу описане спецификације. Стога је кључни закључак да *FPE* алгоритми представљају практично употребљиву опцију за замагљивање заглавља пакета и заштиту приватности.

Употребљивост *LISPP* система у реалним сценаријима практично је демонстрирана кроз развој *LDoH* сервиса, који комбинује механизам замагљивања мрежних адреса са стандардном заштитом *DNS* саобраћаја. Основна замисао *LDoH* сервиса је да замени архитектуру *ODoH* сервиса, који се ослања на посредника ради раздвајања идентитета корисника од садржаја упита, ефикаснијим решењем за заштиту приватности директно на мрежном слоју. *ODoH* уводи велико додатно кашњење по пакету услед вишеструког шифровања и прослеђивања преко посредника који терминира *TLS* везу између клијента и разрешивача. Насупрот томе, *LDoH* избегава употребу асиметричног шифровања и не терминира *TLS* везу између клијента и разрешивача одржавајући време разрешења готово идентичним као код стандардног *DoH* сервиса. Резултати експерименталне евалуације показали су да *LDoH* систем надмашује *ODoH* у погледу перформанси и до 45 % како за појединачни упит тако и у случају истовременог налета од више стотина упита, чи-

ме је потврђена полазна хипотеза да је реализација заштите приватности на мрежном слоју коришћењем *FPE* алгоритама ефикаснија од постојећих механизма за заштиту приватности.

Даље истраживачке активности усмерене су у два кључна правца. Први истраживачки правац тиче се даље оптимизације перформанси *LISPP* система. У том циљу посебна пажња биће посвећена вишепроцесорским платформама програмабилних мрежних картица, као и бољој интеграцији са језгром *Linux* оперативног система у случају *eBPF* технологије како би се остварио приступ наменским инструкцијама за криптографске операције. Такође, биће имплементирано још *FPE* алгоритама, поред *FF3-1* алгорита који је у међувремену изгубио статус препорученог, као што су *FF1* алгоритам или неки од једноставнијих криптографских алгоритама у циљу постизања бољих перформанси. Други истраживачки правац тиче се развоја додатних специјализованих решења базираних на *LISPP* систему за друге мрежне протоколе поред *DNS* сервиса. Једна од идеја јесте интеграција са контролном равни етапног рутирања ради постизања мањег кашњења него што га имају традиционални системи етапног рутирања. За крај, потребно је размотрити повезивање са *IRTF* радним групама ради дискусије о практичној примени система заснованих на *FPE* алгоритмима у циљу заштите приватности.

Литература

- [1] T. Bujlow, V. Carela-Español, J. Solé-Pareta, and P. Barlet-Ros, “A Survey on Web Tracking: Mechanisms, Implications, and Defenses,” *Proceedings of the IEEE*, vol. 105, no. 8, pp. 1476–1510, Mar. 2017. doi: <https://doi.org/10.1109/JPROC.2016.2637878>.
- [2] D. J. Leith, “Web Browser Privacy: What Do Browsers Say When They Phone Home?” *IEEE Access*, vol. 9, p. 41615–41627, 2021. doi: <https://doi.org/10.1109/access.2021.3065243>.
- [3] Z. Weinberg, E. Y. Chen, P. R. Jayaraman, and C. Jackson, “I Still Know What You Visited Last Summer: Leaking Browsing History via User Interaction and Side Channel Attacks,” in *2011 IEEE Symposium on Security and Privacy*. IEEE, May 2011, p. 147–161. doi: <https://doi.org/10.1109/sp.2011.23>.
- [4] S. Farrell and H. Tschofenig, “Pervasive Monitoring Is an Attack,” Internet Engineering Task Force, Request for Comments 7258, May 2014, Accessed on 23 Apr 2025. [Online]. Available: <https://www.rfc-editor.org/info/rfc7258>
- [5] J. Muir and P. V. Oorschot, “Internet geolocation: Evasion and counterevasion,” *ACM Computing Surveys*, vol. 42, no. 1, Dec. 2009. doi: <https://doi.org/10.1145/1592451.1592455>.
- [6] N. P. Hoang, A. A. Niaki, P. Gill, and M. Polychronakis, “Domain name encryption is not enough: privacy leakage via IP-based website fingerprinting,” in *Proceedings on Privacy Enhancing Technologies (PETS)*, vol. 2021. Privacy Enhancing Technologies Symposium Advisory Board, Jul. 2021, p. 420–440. doi: <https://doi.org/10.2478/popets-2021-0078>.
- [7] D. W. Kim and J. Zhang, “You Are How You Query: Deriving Behavioral Fingerprints from DNS Traffic,” in *Security and Privacy in Communication Networks - SecureComm 2015*. Springer International Publishing, 2015, p. 348–366. doi: https://doi.org/10.1007/978-3-319-28865-9_19.
- [8] S. J. Saidi, O. Gasser, and G. Smaragdakis, “One bad apple can spoil your IPv6 privacy,” *ACM SIGCOMM Computer Communication Review*, vol. 52, no. 2, p. 10–19, Apr. 2022. doi: <https://doi.org/10.1145/3544912.3544915>.
- [9] R. Koch, “What Is Considered Personal Data under the EU GDPR?” GDPR.eu, Tech. Rep., 2019, Accessed on 23 Apr 2025. [Online]. Available: <https://gdpr.eu/eu-gdpr-personal-data/>
- [10] M. Mićović, U. Radenković, and P. Vuletić, “Network Layer Privacy Protection Using Format-Preserving Encryption,” *Electronics*, vol. 12, no. 23, Nov. 2023. doi: <https://doi.org/10.3390/electronics12234800>.

- [11] E. Kfoury, S. Choueiri, A. Mazloun, A. AlSabeH, J. Gomez, and J. Crichigno, “A Comprehensive Survey on SmartNICs: Architectures, Development Models, Applications, and Research Directions,” *IEEE Access*, vol. 12, pp. 107 297–107 336, 2024. doi: <https://doi.org/10.1109/ACCESS.2024.3437203>.
- [12] S. Elizalde, A. AlSabeH, A. Mazloun, S. Choueiri, E. Kfoury, J. Gomez, and J. Crichigno, “A survey on security applications with SmartNICs: Taxonomy, implementations, challenges, and future trends,” *Journal of Network and Computer Applications*, vol. 242, Oct. 2025. doi: <https://doi.org/10.1016/j.jnca.2025.104257>.
- [13] A. Cahn, S. Alfeld, P. Barford, and S. Muthukrishnan, “An Empirical Study of Web Cookies,” in *Proceedings of the 25th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2016, p. 891–901. doi: <https://doi.org/10.1145/2872427.2882991>.
- [14] J. R. Mayer and J. C. Mitchell, “Third-Party Web Tracking: Policy and Technology,” in *2012 IEEE Symposium on Security and Privacy*. IEEE, May 2012, pp. 413–427. doi: <https://doi.org/10.1109/SP.2012.47>.
- [15] T.-C. Li, H. Hang, M. Faloutsos, and P. Efstathopoulos, “TrackAdvisor: Taking Back Browsing Privacy from Third-Party Trackers,” in *Passive and Active Measurement*. Springer International Publishing, 2015, p. 277–289. doi: https://doi.org/10.1007/978-3-319-15509-8_21.
- [16] X. Hu and N. Sastry, “Characterising Third Party Cookie Usage in the EU after GDPR,” in *Proceedings of the 10th ACM Conference on Web Science*. Association for Computing Machinery (ACM), 2019, p. 137–141. doi: <https://doi.org/10.1145/3292522.3326039>.
- [17] M. Kretschmer, J. Pennekamp, and K. Wehrle, “Cookie Banners and Privacy Policies: Measuring the Impact of the GDPR on the Web,” *ACM Transactions on the Web*, vol. 15, no. 4, Jul. 2021. doi: <https://doi.org/10.1145/3466722>.
- [18] E. Papadogiannakis, P. Papadopoulos, N. Kourtellis, and E. P. Markatos, “User tracking in the post-cookie era: How websites bypass gdpr consent to track users,” in *Proceedings of the Web Conference 2021*. Association for Computing Machinery (ACM), 2021, p. 2130–2141. doi: <https://doi.org/10.1145/3442381.3450056>.
- [19] D. Martínez, A. Molero, E. Calle, D. C. Ametller, and A. Jové, “Large-scale web tracking and cookie compliance: Evaluating one million websites under GDPR with AI categorization,” *Journal of Network and Computer Applications*, vol. 242, Oct. 2025. doi: <https://doi.org/10.1016/j.jnca.2025.104222>.
- [20] A. Gomez-Boix, P. Laperdrix, and B. Baudry, “Hiding in the Crowd,” in *Proceedings of the 2018 World Wide Web Conference on World Wide Web*. Association for Computing Machinery (ACM), 2018, p. 309–318. doi: <https://doi.org/10.1145/3178876.3186097>.
- [21] “Cover your tracks,” Electronic Frontier Foundation, 2025, Accessed on 23 Apr 2025. [Online]. Available: <https://coveryourtracks.eff.org/>
- [22] “Identify Every Visitor,” FingerprintJS, 2025, Accessed on 23 Apr 2025. [Online]. Available: <https://fingerprint.com/>
- [23] “BrowserLeaks,” 2025, Accessed on 23 Apr 2025. [Online]. Available: <https://browserleaks.com/>

- [24] P. Laperdrix, N. Bielova, B. Baudry, and G. Avoine, “Browser Fingerprinting: A Survey,” *ACM Transactions on the Web*, vol. 14, no. 2, Apr. 2020. doi: <https://doi.org/10.1145/3386040>.
- [25] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz, “The Web Never Forgets: Persistent Tracking Mechanisms in the Wild,” in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery (ACM), 2014, p. 674–689. doi: <https://doi.org/10.1145/2660267.2660347>.
- [26] T. Van Goethem, W. Joosen, and N. Nikiforakis, “The clock is still ticking: Timing attacks in the modern web,” in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery (ACM), Oct. 2015, p. 1382–1393. doi: <https://doi.org/10.1145/2810103.2813632>.
- [27] L. Olejnik, C. Castelluccia, and A. Janc, “Why Johnny Can’t Browse in Peace: On the Uniqueness of Web Browsing History Patterns,” in *International Symposium on Privacy Enhancing Technologies*, Jul. 2012. [Online]. Available: <https://petsymposium.org/2012/papers/hotpets12-4-johnny.pdf>
- [28] S. Bird, I. Segall, and M. Lopatka, “Replication: Why We Still Can’t Browse in Peace: On the Uniqueness and Reidentifiability of Web Browsing Histories,” in *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*. USENIX Association, Aug. 2020, pp. 489–503. [Online]. Available: <https://www.usenix.org/conference/soups2020/presentation/bird>
- [29] R. Dingleline, N. Mathewson, and P. Syverson, “Tor: The second-generation onion router,” in *USENIX security symposium*, vol. 4, Aug. 2004, pp. 303–320. [Online]. Available: https://www.usenix.org/legacy/publications/library/proceedings/sec04/tech/full_papers/dingleline/dingleline.pdf
- [30] “Tor,” Tor Project, 2025, Accessed on 23 Apr 2025. [Online]. Available: <https://www.torproject.org/>
- [31] F. Tschorsch and B. Scheuermann, “Mind the gap: towards a backpressure-based transport protocol for the Tor network,” in *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*. USENIX Association, 2016, p. 597–610.
- [32] R. Dingleline and S. J. Murdoch, “Network Performance Tests over the 100G BELLA Link between GÉANT and RNP,” Tor Project, Blog, Mar. 2009, Accessed on 23 Apr 2025. [Online]. Available: <https://svn-archive.torproject.org/svn/projects/roadmaps/2009-03-11-performance.pdf>
- [33] A. Panchenko, L. Pimenidis, and J. Renner, “Performance Analysis of Anonymous Communication Channels Provided by Tor,” in *2008 Third International Conference on Availability, Reliability and Security*. IEEE, Mar. 2008, p. 221–228. doi: <https://doi.org/10.1109/ares.2008.63>.
- [34] S. Murdoch and G. Danezis, “Low-cost traffic analysis of Tor,” in *2005 IEEE Symposium on Security and Privacy*. IEEE, May 2005, pp. 183–195. doi: <https://doi.org/10.1109/SP.2005.12>.
- [35] L. Overlier and P. Syverson, “Locating hidden servers,” in *2006 IEEE Symposium on Security and Privacy*. IEEE, May 2006. doi: <https://doi.org/10.1109/SP.2006.24>.

- [36] P. Syverson, G. Tsudik, M. Reed, and C. Landwehr, “Towards an analysis of onion routing security,” in *International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability*. Springer Berlin Heidelberg, 2001, p. 96–114. doi: <https://doi.org/10.5555/371931.371981>.
- [37] B. N. Levine, M. K. Reiter, C. Wang, and M. Wright, “Timing Attacks in Low-Latency Mix Systems,” in *Financial Cryptography*. Springer Berlin Heidelberg, 2004, p. 251–265. doi: https://doi.org/10.1007/978-3-540-27809-2_25.
- [38] B. Greschbach, T. Pulls, L. M. Roberts, P. Winter, and N. Feamster, “The Effect of DNS on Tor’s Anonymity,” in *Proceedings 2017 Network and Distributed System Security Symposium*. Internet Society, Feb. 2017. doi: <https://doi.org/10.14722/ndss.2017.23311>.
- [39] D. L. Chaum, “Untraceable electronic mail, return addresses, and digital pseudonyms,” *Communications of the ACM*, vol. 24, no. 2, p. 84–90, Feb. 1981. doi: <https://doi.org/10.1145/358549.358563>.
- [40] K. Sampigethaya and R. Poovendran, “A Survey on Mix Networks and Their Secure Applications,” *Proceedings of the IEEE*, vol. 94, no. 12, pp. 2142–2181, 2006. doi: <https://doi.org/10.1109/JPROC.2006.889687>.
- [41] C. Diaz, *Mix Networks*. Springer Berlin Heidelberg, 2019, pp. 1–5. doi: https://doi.org/10.1007/978-3-642-27739-9_1754-1.
- [42] “HTTPS Encryption on the Web,” Google, 2025, Accessed on 23 Apr 2025. [Online]. Available: <https://transparencyreport.google.com/https/overview>
- [43] I. Karunanayake, N. Ahmed, R. Malaney, R. Islam, and S. K. Jha, “De-Anonymisation Attacks on Tor: A Survey,” *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, p. 2324–2350, 2021. doi: <https://doi.org/10.1109/comst.2021.3093615>.
- [44] A. AlSabeih, J. Khoury, E. Kfoury, J. Crichigno, and E. Bou-Harb, “A survey on security applications of P4 programmable switches and a STRIDE-based vulnerability assessment,” *Computer Networks*, vol. 207, Apr. 2022. doi: <https://doi.org/10.1016/j.comnet.2022.108800>.
- [45] P. Subharthi, P. Jianli, and J. Raj, “Architectures for the future networks and the next generation Internet: A survey,” *Computer Communications*, vol. 34, no. 1, pp. 2–42, 2011. doi: <https://doi.org/10.1016/j.comcom.2010.08.001>.
- [46] C. Chen, D. E. Asoni, D. Barrera, G. Danezis, and A. Perrig, “HORNET: High-speed Onion Routing at the Network Layer,” in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery (ACM), Oct. 2015, p. 1441–1454. doi: <https://doi.org/10.1145/2810103.2813628>.
- [47] C. Filsfils, K. Talaulikar, D. Voyer, A. Bogdanov, and P. Mattes, “Segment Routing Architecture,” Internet Engineering Task Force, Request for Comments 9256, Jul. 2022, Accessed on 23 Apr 2025. [Online]. Available: <https://www.rfc-editor.org/info/rfc9256>
- [48] X. Yang, D. Clark, and A. W. Berger, “NIRA: A New Inter-Domain Routing Architecture,” *IEEE/ACM Transactions on Networking*, vol. 15, no. 4, pp. 775–788, Jul. 2007. doi: <https://doi.org/10.1109/TNET.2007.893888>.

- [49] X. Zhang, H.-C. Hsiao, G. Hasker, H. Chan, A. Perrig, and D. Andersen, “SCION: Scalability, Control, and Isolation on Next-Generation Networks,” in *2011 IEEE Symposium on Security and Privacy*. IEEE, May 2011, pp. 212–227. doi: <https://doi.org/10.1109/SP.2011.45>.
- [50] B. Godfrey, I. Ganichev, S. Shenker, and I. Stoica, “Pathlet routing,” *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, p. 111–122, Aug. 2009. doi: <https://doi.org/10.1145/1594977.1592583>.
- [51] G. Danezis and I. Goldberg, “Sphinx: A Compact and Provably Secure Mix Format,” in *2009 30th IEEE Symposium on Security and Privacy*. IEEE, Aug. 2009, pp. 269–282. doi: <https://doi.org/10.1109/SP.2009.15>.
- [52] A. Pfitzmann and M. Köhntopp, *Anonymity, Unobservability, and Pseudonymity — A Proposal for Terminology*. Springer Berlin Heidelberg, 2001, p. 1–9. doi: https://doi.org/10.1007/3-540-44702-4_1.
- [53] C. Chen, D. E. Asoni, A. Perrig, D. Barrera, G. Danezis, and C. Troncoso, “TARANET: Traffic-Analysis Resistant Anonymity at the Network Layer,” in *2018 IEEE European Symposium on Security and Privacy (EuroSec’P)*. IEEE, Apr. 2018, p. 137–152. doi: <https://doi.org/10.1109/eurosp.2018.00018>.
- [54] T. Grube, M. Thummerer, J. Daubert, and M. Mühlhäuser, “Cover Traffic: A Trade of Anonymity and Efficiency,” in *Security and Trust Management*. Springer International Publishing, 2017, p. 213–223. doi: https://doi.org/10.1007/978-3-319-68063-7_15.
- [55] H.-C. Hsiao, T. H.-J. Kim, A. Perrig, A. Yamada, S. C. Nelson, M. Gruteser, and W. Meng, “LAP: Lightweight Anonymity and Privacy,” in *2012 IEEE Symposium on Security and Privacy*. IEEE, May 2012, p. 506–520. doi: <https://doi.org/10.1109/sp.2012.37>.
- [56] J. de Ruiter and C. Schutijser, “Next-generation internet at terabit speed: SCION in P4,” in *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies*. Association for Computing Machinery (ACM), 2021, p. 119–125. doi: <https://doi.org/10.1145/3485983.3494839>.
- [57] P. Mahadevan, D. Krioukov, M. Fomenkov, X. Dimitropoulos, k. c. claffy, and A. Vahdat, “The internet AS-level topology: three data sources and one definitive metric,” *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, p. 17–26, Jan. 2005. doi: <https://doi.org/10.1145/1111322.1111328>.
- [58] J. Sankey and M. Wright, “Dovetail: Stronger Anonymity in Next-Generation Internet Routing,” in *Privacy Enhancing Technologies (PETS)*, vol. 8555. Springer International Publishing, 2014, p. 283–303. doi: https://doi.org/10.1007/978-3-319-08506-7_15.
- [59] A. Pfitzmann and M. Hansen, “A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management,” Aug. 2010. [Online]. Available: https://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.34.pdf
- [60] V. Giotsas and S. Zhou, “Valley-free violation in Internet routing - Analysis based on BGP Community data,” in *2012 IEEE International Conference on Communications (ICC)*. IEEE, Jun. 2012, p. 1193–1197. doi: <https://doi.org/10.1109/icc.2012.6363987>.

- [61] C. Chen and A. Perrig, “PHI: Path-Hidden Lightweight Anonymity Protocol at Network Layer,” in *Proceedings on Privacy Enhancing Technologies (PETS)*, vol. 2017. Privacy Enhancing Technologies Symposium Advisory Board, Dec. 2016, p. 100–117. doi: <https://doi.org/10.1515/popets-2017-0007>.
- [62] H. M. Moghaddam and A. Mosenia, “Anonymizing Masses: Practical Light-weight Anonymity at the Network Level,” 2019. doi: <https://doi.org/10.48550/arXiv.1911.09642>., Accessed on 23 Apr 2025.
- [63] B. Raghavan, T. Kohno, A. C. Snoeren, and D. Wetherall, “Enlisting ISPs to Improve Online Privacy: IP Address Mixing by Default,” in *Privacy Enhancing Technologies (PETS)*, vol. 5672. Springer Berlin Heidelberg, 2009, p. 143–163. doi: https://doi.org/10.1007/978-3-642-03168-7_9.
- [64] T. Datta, N. Feamster, J. Rexford, and L. Wang, “SPINE: Surveillance Protection in the Network Elements,” in *9th USENIX Workshop on Free and Open Communications on the Internet (FOCI 19)*. USENIX Association, Aug. 2019. [Online]. Available: <https://www.usenix.org/conference/foci19/presentation/datta>
- [65] S. Patil and N. Borisov, “What can you learn from an IP?” in *Proceedings of the Applied Networking Research Workshop*. Association for Computing Machinery (ACM), Jul. 2019, p. 45–51. doi: <https://doi.org/10.1145/3340301.3341133>.
- [66] L. Wang, H. Kim, P. Mittal, and J. Rexford, “Programmable In-Network Obfuscation of Traffic,” 2020. doi: <https://doi.org/10.48550/arXiv.2006.00097>., Accessed on 23 Apr 2025.
- [67] P. Richter, F. Wohlfart, N. Vallina-Rodriguez, M. Allman, R. Bush, A. Feldmann, C. Kreibich, N. Weaver, and V. Paxson, “A Multi-perspective Analysis of Carrier-Grade NAT Deployment,” in *Proceedings of the 2016 Internet Measurement Conference*. Association for Computing Machinery (ACM), 2016, p. 215–229. doi: <https://doi.org/10.1145/2987443.2987474>.
- [68] C. Kuhn, M. Beck, and T. Strufe, “Breaking and (Partially) Fixing Provably Secure Onion Routing,” in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, May 2020, p. 168–185. doi: <https://doi.org/10.1109/sp40000.2020.00039>.
- [69] D. Das, S. Meiser, E. Mohammadi, and A. Kate, “Anonymity Trilemma: Strong Anonymity, Low Bandwidth Overhead, Low Latency - Choose Two,” in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, May 2018, pp. 108–126. doi: <https://doi.org/10.1109/SP.2018.00011>.
- [70] G. Huston, “AS6447 IPv6 BGP Table Data,” APNIC Labs, BGP Metrics, 2024, Accessed on 23 Apr 2025. [Online]. Available: <https://bgp.potaroo.net/v6/as6447/>
- [71] R. Schroepfel, “Hasty Pudding Cipher Specification,” Jun. 1998, Accessed on 23 Apr 2025. [Online]. Available: <https://web.archive.org/web/20111007174344/http://richard.schroepfel.name:8015/hpc/hpc-spec>
- [72] M. Dworkin, “Recommendation for Block Cipher Modes of Operation: Methods for Format-Preserving Encryption,” National Institute of Standards and Technology, NIST Special Publication 800-38G, 2016, Accessed on 16 May 2024. [Online]. Available: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38G.pdf>

- [73] J.-K. Lee, B. Koo, D. Roh, W.-H. Kim, and D. Kwon, “Format-Preserving Encryption Algorithms Using Families of Tweakable Blockciphers,” in *Information Security and Cryptology - ICISC 2014*, vol. 8949. Springer International Publishing, 2015, p. 132–159. doi: https://doi.org/10.1007/978-3-319-15943-0_9.
- [74] M. Liskov, R. L. Rivest, and D. Wagner, “Tweakable Block Ciphers,” in *Advances in Cryptology - CRYPTO 2002*. Springer Berlin Heidelberg, 2002, pp. 31–46. doi: https://doi.org/10.1007/3-540-45708-9_3.
- [75] D. Goldenberg, S. Hohenberger, M. Liskov, E. C. Schwartz, and H. Seyalioglu, “On Tweaking Luby-Rackoff Blockciphers,” in *Advances in Cryptology - ASIACRYPT 2007*. Springer Berlin Heidelberg, 2007, pp. 342–356. doi: https://doi.org/10.1007/978-3-540-76900-2_21.
- [76] W. Jang and S.-Y. Lee, “A format-preserving encryption FF1, FF3-1 using lightweight block ciphers LEA and, SPECK,” in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*. Association for Computing Machinery (ACM), Mar. 2020, p. 369–375. doi: <https://doi.org/10.1145/3341105.3373953>.
- [77] M. Bellare, P. Rogaway, and T. Spies, “The FFX Mode of Operation for Format-Preserving Encryption,” Feb. 2010. [Online]. Available: <https://csrc.nist.gov/csrc/media/projects/block-cipher-techniques/documents/bcm/proposed-modes/ffx/ffx-spec.pdf>
- [78] M. Bellare, P. Rogaway, and T. Spies, “Addendum to The FFX Mode of Operation for Format-Preserving Encryption,” Sep. 2010. [Online]. Available: <https://csrc.nist.gov/csrc/media/projects/block-cipher-techniques/documents/bcm/proposed-modes/ffx/ffx-spec2.pdf>
- [79] O. Berthold, A. Pfitzmann, and R. Standtke, *The Disadvantages of Free MIX Routes and How to Overcome Them*. Springer Berlin Heidelberg, 2001, p. 30–45. doi: https://doi.org/10.1007/3-540-44702-4_3.
- [80] E. Brier, T. Peyrin, and J. Stern, “BPS: a Format-Preserving Encryption Proposal,” 2010. [Online]. Available: <https://csrc.nist.gov/csrc/media/projects/block-cipher-techniques/documents/bcm/proposed-modes/bps/bps-spec.pdf>
- [81] F. B. Durak and S. Vaudenay, “Breaking the FF3 Format-Preserving Encryption Standard over Small Domains,” in *Advances in Cryptology – EUROCRYPT 2017*, vol. 10402. Springer International Publishing, 2017, p. 679–707. doi: https://doi.org/10.1007/978-3-319-63715-0_23.
- [82] T. Beyne, “Linear Cryptanalysis of FF3-1 and FEA,” in *Advances in Cryptology – EUROCRYPT 2021*, vol. 12825. Springer International Publishing, 2021, p. 41–69. doi: https://doi.org/10.1007/978-3-030-84242-0_3.
- [83] O. Dunkelman, A. Kumar, E. Lambooi, and S. K. Sanadhya, “Cryptanalysis of Feistel-Based Format-Preserving Encryption,” 2020, Accessed on 23 Apr 2025. [Online]. Available: <https://eprint.iacr.org/2020/1311>
- [84] O. Amon, O. Dunkelman, N. Keller, E. Ronen, and A. Shamir, “Three Third Generation Attacks on the Format Preserving Encryption Scheme FF3,” in *Advances in Cryptology – EUROCRYPT 2021*, vol. 12697. Springer International Publishing, 2021, p. 127–154. doi: https://doi.org/10.1007/978-3-030-77886-6_5.

- [85] D. Chang, M. Ghosh, K. C. Gupta, A. Jati, A. Kumar, D. Moon, I. G. Ray, and S. K. Sanadhya, “SPF: A New Family of Efficient Format-Preserving Encryption Algorithms,” in *Information Security and Cryptology*, vol. 10143. Springer International Publishing, 2017, p. 64–83. doi: https://doi.org/10.1007/978-3-319-54705-3_5.
- [86] F. B. Durak, H. Horst, M. Horst, and S. Vaudenay, “FAST: Secure and High Performance Format-Preserving Encryption and Tokenization,” in *Advances in Cryptology - ASIACRYPT 2021*, vol. 13092. Springer International Publishing, 2021, p. 465–489. doi: https://doi.org/10.1007/978-3-030-92078-4_16.
- [87] K. Ye, “An Efficient Format-Preserving Encryption Scheme Based on RECTANGLE,” in *2025 8th International Conference on Computer Information Science and Application Technology (CISAT)*, Jul. 2025, pp. 1294–1301. doi: <https://doi.org/10.1109/CISAT66811.2025.11181692>.
- [88] M. Larsen and F. Gont, “Recommendations for Transport-Protocol Port Randomization,” Internet Engineering Task Force, Request for Comments 6056, Jan. 2011, Accessed on 23 Apr 2025. [Online]. Available: <https://www.rfc-editor.org/info/rfc6056>
- [89] M. Cotton, L. Eggert, D. J. Touch, M. Westerlund, and S. Cheshire, “Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry,” Internet Engineering Task Force, Request for Comments 6335, Aug. 2011, Accessed on 23 Apr 2025. [Online]. Available: <https://www.rfc-editor.org/info/rfc6335>
- [90] F. Gont, S. Krishnan, T. Narten, and R. Draves, “Temporary Address Extensions for Stateless Address Autoconfiguration in IPv6,” Internet Engineering Task Force, Request for Comments 8981, Feb. 2021, Accessed on 23 Apr 2025. [Online]. Available: <https://www.rfc-editor.org/info/rfc8981>
- [91] D. Herrmann, K.-P. Fuchs, J. Lindemann, and H. Federrath, “EncDNS: A Lightweight Privacy-Preserving Name Resolution Service,” in *Computer Security – ESORICS 2014*, vol. 8712. Springer International Publishing, 2014, p. 37–55. doi: https://doi.org/10.1007/978-3-319-11203-9_3.
- [92] “Data Retention,” Jun. 2021, Accessed on 23 Apr 2025. [Online]. Available: <https://www.statewatch.org/media/2592/eu-council-data-retention-com-non-paper-wk-7294-2021.pdf>
- [93] T. A. Nguyen, M. Kim, J. Lee, D. Min, J.-W. Lee, and D. Kim, “Performability evaluation of switch-over Moving Target Defence mechanisms in a Software Defined Networking using stochastic reward nets,” *Journal of Network and Computer Applications*, vol. 199, Mar. 2022. doi: <https://doi.org/10.1016/j.jnca.2021.103267>.
- [94] N. Feamster, J. Rexford, and E. Zegura, “The road to SDN: an intellectual history of programmable networks,” *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, p. 87–98, Apr. 2014. doi: <https://doi.org/10.1145/2602204.2602219>.
- [95] M. Casado, N. McKeown, and S. Shenker, “From ethane to SDN and beyond,” *ACM SIGCOMM Computer Communication Review*, vol. 49, no. 5, p. 92–95, Nov. 2019. doi: <https://doi.org/10.1145/3371934.3371963>.
- [96] F. Ruffy, “Static Analysis Tools For Network-Device Stacks,” Ph.D. dissertation, Department of Computer Science, New York University, 2025.

- [97] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: enabling innovation in campus networks,” *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, p. 69–74, Mar. 2008. doi: <https://doi.org/10.1145/1355734.1355746>.
- [98] *OpenFlow Switch Specification*, Open Networking Foundation, 2015, Accessed on 15 Feb 2025. [Online]. Available: <https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf>
- [99] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, “P4: Programming Protocol-Independent Packet Processors,” *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, p. 87–95, Jul. 2014. doi: <https://doi.org/10.1145/2656877.2656890>.
- [100] M. Budiu and C. Dodd, “The P416 Programming Language,” *ACM SIGOPS Operating Systems Review*, vol. 51, no. 1, p. 5–14, Sep. 2017. doi: <https://doi.org/10.1145/3139645.3139648>.
- [101] A. Liatifis, P. Sarigiannidis, V. Argyriou, and T. Lagkas, “Advancing SDN from OpenFlow to P4: A Survey,” *ACM Computing Surveys*, vol. 55, no. 9, Jan. 2023. doi: <https://doi.org/10.1145/3556973>.
- [102] *P4-14 Language Specification*, P4 Language Consortium, 2018, Accessed on 23 Apr 2025. [Online]. Available: <https://p4.org/wp-content/uploads/sites/53/p4-spec/p4-14/v1.1.0/tex/p4.pdf>
- [103] *P4-16 Language Specification*, P4 Language Consortium, 2024, Accessed on 23 Apr 2025. [Online]. Available: <https://p4.org/wp-content/uploads/sites/53/2024/10/P4-16-spec-v1.2.5.pdf>
- [104] *P4-16 Portable Switch Architecture (PSA)*, P4 Language Consortium, 2022, Accessed on 23 Apr 2025. [Online]. Available: <https://p4.org/wp-content/uploads/sites/53/p4-spec/docs/PSA-v1.2.pdf>
- [105] A. Mazloun, E. Kfoury, J. Gomez, and J. Crichigno, “A Survey on Rerouting Techniques with P4 Programmable Data Plane Switches,” *Computer Networks*, vol. 230, Jul. 2023. doi: <https://doi.org/10.1016/j.comnet.2023.109795>.
- [106] “Architecture for Simple Switch - v1model.p4,” P4 Language Consortium, Apr. 2019, Accessed on 23 Apr 2025. [Online]. Available: <https://github.com/p4lang/p4c/blob/main/p4include/v1model.p4>
- [107] F. Hauser, M. Häberle, D. Merling, S. Lindner, V. Gurevich, F. Zeiger, R. Frank, and M. Menth, “A survey on data plane programming with P4: Fundamentals, advances, and applied research,” *Journal of Network and Computer Applications*, vol. 212, Mar. 2023. doi: <https://doi.org/10.1016/j.jnca.2022.103561>.
- [108] S. McCanne and V. Jacobson, “The BSD Packet Filter: A New Architecture for User-level Packet Capture,” in *Proceedings of the USENIX Winter 1993 Conference*, vol. 46, Jan. 1993, pp. 259–270. [Online]. Available: <https://www.usenix.org/legacy/publications/library/proceedings/sd93/mccanne.pdf>

- [109] M. A. M. Vieira, M. S. Castanho, R. D. G. Pacífico, E. R. S. Santos, E. P. M. C. Júnior, and L. F. M. Vieira, “Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications,” *ACM Computing Surveys*, vol. 53, no. 1, Feb. 2020. doi: <https://doi.org/10.1145/3371038>.
- [110] L. Rice, *Learning eBPF*. O’Reilly Media, 2023.
- [111] J. Corbet, “BPF: the universal in-kernel virtual machine,” LWN.net, May 2014, Accessed on 23 Apr 2025. [Online]. Available: <https://lwn.net/Articles/599755/>
- [112] J. Corbet, “Extending extended BPF,” LWN.net, Jul. 2014, Accessed on 23 Apr 2025. [Online]. Available: <https://lwn.net/Articles/603983/>
- [113] J. Corbet, “A JIT for packet filters,” LWN.net, Apr. 2011, Accessed on 23 Apr 2025. [Online]. Available: <https://lwn.net/Articles/437981/>
- [114] D. Thaler, “BPF Instruction Set Architecture (ISA),” Internet Engineering Task Force, Request for Comments 9669, Oct. 2024, Accessed on 16 Dec 2025. [Online]. Available: <https://www.rfc-editor.org/info/rfc9669>
- [115] M. H. N. Mohamed, X. Wang, and B. Ravindran, “Understanding the Security of Linux eBPF Subsystem,” in *Proceedings of the 14th ACM SIGOPS Asia-Pacific Workshop on Systems*. Association for Computing Machinery (ACM), 2023, p. 87–92. doi: <https://doi.org/10.1145/3609510.3609822>.
- [116] D. Lu, B. Tang, M. Paper, and M. Kogias, “Towards Functional Verification of eBPF Programs,” in *Proceedings of the ACM SIGCOMM 2024 Workshop on EBPF and Kernel Extensions*. Association for Computing Machinery (ACM), 2024, p. 37–43. doi: <https://doi.org/10.1145/3672197.3673435>.
- [117] H. Sun and Z. Su, “Validating the eBPF verifier via state embedding,” in *Proceedings of the 18th USENIX Conference on Operating Systems Design and Implementation*. USENIX Association, 2024. doi: <https://doi.org/10.5555/3691938.3691971>.
- [118] T. Hoiland-Jorgensen, J. D. Brouer, D. Borkmann, J. Fastabend, T. Herbert, D. Ahern, and D. Miller, “The eXpress data path: fast programmable packet processing in the operating system kernel,” in *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*. Association for Computing Machinery (ACM), 2018, p. 54–66. doi: <https://doi.org/10.1145/3281411.3281443>.
- [119] *Drivers supporting native XDP*, Cilium, 2025, Accessed on 23 Apr 2025. [Online]. Available: <https://docs.cilium.io/en/stable/reference-guides/bpf/progtypes/#xdp-drivers>
- [120] *NFP-4000 Theory of Operation*, Netronome, 2023, Accessed on 17 Oct 2023. [Online]. Available: https://web.archive.org/web/20220730092929/https://www.netronome.com/static/app/img/products/silicon-solutions/WP_NFP4000_TOO.pdf
- [121] *The Joy of Micro-C*, Netronome, 2023, Accessed on 17 Oct 2023. [Online]. Available: https://web.archive.org/web/20221006024432/https://cdn.open-nfp.org/media/documents/the-joy-of-micro-c_fcjSfra.pdf
- [122] *Programming Netronome Agilio® SmartNICs*, Netronome, 2023, Accessed on 17 Oct 2023. [Online]. Available: https://web.archive.org/web/20230121100612/https://www.netronome.com/media/documents/WP_NFP_Programming_Model.pdf

- [123] “Behavioral Model (BMv2),” P4 Language Consortium, Jan. 2015, Accessed on 23 Apr 2025. [Online]. Available: <https://github.com/p4lang/behavioral-model>
- [124] M. Mićović, “LISPP: A Lightweight Stateless Network Layer Privacy Protection System,” Oct. 2023, Accessed on 23 Apr 2025. [Online]. Available: <https://github.com/marko-micovic/lispp>
- [125] S. Kaur, K. Kumar, and N. Aggarwal, “A review on P4-Programmable data planes: Architecture, research efforts, and future directions,” *Computer Communications*, vol. 170, p. 109–129, Mar. 2021. doi: <https://doi.org/10.1016/j.comcom.2021.01.027>.
- [126] X. Chen, “Implementing AES Encryption on Programmable Switches via Scrambled Lookup Tables,” in *Proceedings of the Workshop on Secure Programmable Network Infrastructure*. Association for Computing Machinery (ACM), Aug. 2020, p. 8–14. doi: <https://doi.org/10.1145/3405669.3405819>.
- [127] H. Harkous, M. Jarschel, M. He, R. Priest, and W. Kellerer, “Towards Understanding the Performance of P4 Programmable Hardware,” in *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE, Sep. 2019. doi: <https://doi.org/10.1109/ancs.2019.8901881>.
- [128] P. B. Viegas, A. G. de Castro, A. F. Lorenzon, F. D. Rossi, and M. C. Luizelli, “The Actual Cost of Programmable SmartNICs: Diving into the Existing Limits,” in *Lecture Notes in Networks and Systems*, vol. 225. Springer International Publishing, 2021, p. 181–194. doi: https://doi.org/10.1007/978-3-030-75100-5_17.
- [129] *Programming NFP with P4 and C*, Netronome, 2023, Accessed on 17 Oct 2023. [Online]. Available: https://web.archive.org/web/20221006035454/https://www.netronome.com/media/documents/WP_Programming_with_P4_and_C.pdf
- [130] G. Bertoni, L. Breveglieri, P. Fragneto, M. Macchetti, and S. Marchesin, “Efficient Software Implementation of AES on 32-Bit Platforms,” in *Cryptographic Hardware and Embedded Systems - CHES 2002*, vol. 2523. Springer Berlin Heidelberg, Jan. 2003, p. 159–171. doi: https://doi.org/10.1007/3-540-36400-5_13.
- [131] F. Shahinfar, S. Miano, A. Panda, and G. Antichi, “Demystifying Performance of eBPF Network Applications,” *Proceedings of the ACM on Networking*, vol. 3, no. CoNEXT3, Sep. 2025. doi: <https://doi.org/10.1145/3749216>.
- [132] N. Hedam, “ebpf - from a programmer’s perspective,” 2025, Accessed on 17 Oct 2025. [Online]. Available: <https://hed.am/papers/2021-EBPF.pdf>
- [133] “iperf3: A TCP, UDP, and SCTP Network Bandwidth Measurement Tool,” Energy Sciences Network (ESnet), Oct. 2023, Accessed on 23 Apr 2025. [Online]. Available: <https://github.com/esnet/iperf>
- [134] “PF_RING - High-Speed Packet Capture, Filtering and Analysis,” ntop, Apr. 2023, Accessed on 23 Apr 2025. [Online]. Available: https://www.ntop.org/products/packet-capture/pf_ring/
- [135] “sockperf: Network Benchmarking Utility,” Mellanox Technologies, Sep. 2022, Accessed on 23 Apr 2025. [Online]. Available: <https://github.com/Mellanox/sockperf>

- [136] R. Lopes, D. Rand, T. Chown, I. Golub, and P. Vuletić, “Network Performance Tests over the 100G BELLA Link between GÉANT and RNP,” GÉANT Association, GN4-3 Project, White Paper GN4-3-22-T5D0T7, Feb. 2023, Accessed on 23 Apr 2025. [Online]. Available: https://resources.geant.org/wp-content/uploads/2023/02/GN4-3_White-Paper_Network-Performance-Tests-Over-100G-BELLA-Link.pdf
- [137] J. Soto and L. Bassham, “Randomness Testing of the Advanced Encryption Standard Finalist Candidates,” Mar. 2000, Accessed on 23 Apr 2025. [Online]. Available: <https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir6483.pdf>
- [138] C. Kenny, “Random Number Generators: An Evaluation and Comparison of Random.Org and Some Commonly Used Generators,” The Distributed Systems Group, Trinity College Dublin, Tech. Rep., Apr. 2005, Accessed on 23 Apr 2025. [Online]. Available: <https://www.random.org/analysis/Analysis2005.pdf>
- [139] M. Eder, “Hypervisor- vs. Container-based Virtualization,” *Network Architectures and Services*, 2016. doi: https://doi.org/10.2313/NET-2016-07-1_01.
- [140] G. Motika and S. Weiss, “Virtio network paravirtualization driver: Implementation and performance of a de-facto standard,” *Computer Standards and Interfaces*, vol. 34, no. 1, pp. 36–47, 2012. doi: <https://doi.org/10.1016/j.csi.2011.05.002>.
- [141] R. Russell, “virtio: towards a de-facto standard for virtual I/O devices,” *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, p. 95–103, Jul. 2008. doi: <https://doi.org/10.1145/1400097.1400108>.
- [142] K. Hynek, “The Impact of Encrypted DNS on Network Security,” Ph.D. dissertation, Faculty of Information Technology, Czech Technical University in Prague, 2023.
- [143] S. Garcia, K. Hynek, D. Vekshin, T. Čejka, and A. Wasicek, “Large Scale Measurement on the Adoption of Encrypted DNS,” 2021. doi: <https://doi.org/10.48550/arXiv.2107.04436>., Accessed on 23 Apr 2025.
- [144] C. Lu, B. Liu, Z. Li, S. Hao, H. Duan, M. Zhang, C. Leng, Y. Liu, Z. Zhang, and J. Wu, “An End-to-End, Large-Scale Measurement of DNS-over-Encryption: How Far Have We Come?” in *Proceedings of the Internet Measurement Conference*. Association for Computing Machinery (ACM), Oct. 2019, p. 22–35. doi: <https://doi.org/10.1145/3355369.3355580>.
- [145] A. Khormali, J. Park, H. Alasmary, A. Anwar, M. Saad, and D. Mohaisen, “Domain name system security and privacy: A contemporary survey,” *Computer Networks*, vol. 185, Feb. 2021. doi: <https://doi.org/10.1016/j.comnet.2020.107699>.
- [146] T. Wicinski, “DNS Privacy Considerations,” Internet Engineering Task Force, Request for Comments 9076, Jul. 2021, Accessed on 23 Apr 2025. [Online]. Available: <https://www.rfc-editor.org/info/rfc9076>
- [147] M. Sun, G. Xu, J. Zhang, and D. W. Kim, “Tracking You through DNS Traffic: Linking User Sessions by Clustering with Dirichlet Mixture Model,” in *Proceedings of the 20th ACM International Conference on Modelling, Analysis and Simulation of Wireless and Mobile Systems*. Association for Computing Machinery (ACM), 2017, p. 303–310. doi: <https://doi.org/10.1145/3127540.3127567>.
- [148] P. Mockapetris, “Domain names - concepts and facilities,” Internet Engineering Task Force, Request for Comments 1034, Nov. 1987, Accessed on 23 Apr 2025. [Online]. Available: <https://www.rfc-editor.org/info/rfc1034>

- [149] P. E. Hoffman and K. Fujiwara, “DNS Terminology,” Internet Engineering Task Force, Request for Comments 9499, Mar. 2024, Accessed on 23 Apr 2025. [Online]. Available: <https://www.rfc-editor.org/info/rfc9499>
- [150] S. Rose, M. Larson, D. Massey, R. Austein, and R. Arends, “DNS Security Introduction and Requirements,” Internet Engineering Task Force, Request for Comments 4033, Mar. 2005, Accessed on 23 Apr 2025. [Online]. Available: <https://www.rfc-editor.org/info/rfc4033>
- [151] R. T. Braden, “Requirements for Internet Hosts - Application and Support,” Internet Engineering Task Force, Request for Comments 1123, Oct. 1989, Accessed on 23 Apr 2025. [Online]. Available: <https://www.rfc-editor.org/info/rfc1123>
- [152] Z. Yan and J.-H. Lee, “The road to DNS privacy,” *Future Generation Computer Systems*, vol. 112, p. 604–611, Nov. 2020. doi: <https://doi.org/10.1016/j.future.2020.06.012>.
- [153] “DNSCrypt,” DNSCrypt project, 2011, Accessed on 23 Apr 2025. [Online]. Available: <https://dnscrypt.info/>
- [154] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. E. Hoffman, “Specification for DNS over Transport Layer Security (TLS),” Internet Engineering Task Force, Request for Comments 7858, May 2016, Accessed on 23 Apr 2025. [Online]. Available: <https://www.rfc-editor.org/info/rfc7858>
- [155] P. E. Hoffman and P. McManus, “DNS Queries over HTTPS (DoH),” Internet Engineering Task Force, Request for Comments 8484, Oct. 2018, Accessed on 23 Apr 2025. [Online]. Available: <https://www.rfc-editor.org/info/rfc8484>
- [156] A. Aggarwal and M. Kumar, “An ensemble framework for detection of DNS-Over-HTTPS (DOH) traffic,” *Multimedia Tools and Applications*, vol. 83, no. 11, p. 32945–32972, Sep. 2023. doi: <https://doi.org/10.1007/s11042-023-16956-9>.
- [157] S. Radhakrishnan, Y. Cheng, J. Chu, A. Jain, and B. Raghavan, “TCP Fast Open,” in *Proceedings of the Seventh Conference on Emerging Networking Experiments and Technologies*. Association for Computing Machinery (ACM), 2011. doi: <https://doi.org/10.1145/2079296.2079317>.
- [158] P. Schmitt, A. Edmundson, A. Mankin, and N. Feamster, “Oblivious DNS: Practical Privacy for DNS Queries,” in *Proceedings on Privacy Enhancing Technologies (PETS)*, vol. 2019, no. 2. Privacy Enhancing Technologies Symposium Advisory Board, Apr. 2019, p. 228–244. [Online]. Available: <https://doi.org/10.2478/popets-2019-0028>
- [159] S. Singanamalla, S. Chunhapanaya, J. Hoyland, M. Vavruša, T. Verma, P. Wu, M. Fayed, K. Heimerl, N. Sullivan, and C. Wood, “Oblivious DNS over HTTPS (ODoH): A Practical Privacy Enhancement to DNS,” in *Proceedings on Privacy Enhancing Technologies (PETS)*, vol. 2021, no. 4. Privacy Enhancing Technologies Symposium Advisory Board, Jul. 2021, p. 575–592. [Online]. Available: <https://doi.org/10.2478/popets-2021-0085>
- [160] E. Kinnear, P. McManus, T. Pauly, T. Verma, and C. A. Wood, “Oblivious DNS over HTTPS,” Internet Engineering Task Force, Request for Comments 9230, Jun. 2022, Accessed on 23 Apr 2025. [Online]. Available: <https://www.rfc-editor.org/info/rfc9230>
- [161] M. Kuehlewind, T. Pauly, and C. A. Wood, “Partitioning as an Architecture for Privacy,” Internet Engineering Task Force, Request for Comments 9614, Jul. 2024, Accessed on 23 Apr 2025. [Online]. Available: <https://www.rfc-editor.org/info/rfc9614>

- [162] “Anonymized DNSCrypt,” DNSCrypt project, Oct. 2019, Accessed on 23 Apr 2025. [Online]. Available: <https://github.com/DNSCrypt/dnscrypt-proxy/wiki/Anonymized-DNS>
- [163] J. Kurihara, T. Tanaka, and T. Kubo, “uODNS: A distributed approach to DNS anonymization with collusion resistance,” *Computer Networks*, vol. 237, p. 110078, Dec. 2023. doi: <https://doi.org/10.1016/j.comnet.2023.110078>.
- [164] N. Sales, “q: A tiny and feature-rich command line DNS client with support for UDP, TCP, DoT, DoH, DoQ, and ODoH,” Mar. 2021, Accessed on 23 Apr 2025. [Online]. Available: <https://github.com/natesales/q>
- [165] “caddy: Every site on HTTPS,” Caddy, Apr. 2015, Accessed on 23 Apr 2025. [Online]. Available: <https://github.com/caddyserver/caddy>
- [166] J. Kurihara, “modoh-server: Relay and Target Server Implementation for Oblivious DNS over HTTPS (ODoH), ODoH-based Mutualized Oblivious DNS (uODoH), and Standard DoH,” Jan. 2024, Accessed on 23 Apr 2025. [Online]. Available: <https://github.com/junkurihara/modoh-server>
- [167] “Oblivious DNS over HTTPS,” Cloudflare, 2021, Accessed on 23 Apr 2025. [Online]. Available: <https://developers.cloudflare.com/1.1.1.1/encryption/oblivious-dns-over-https/>
- [168] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczynski, and W. Joosen, “Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation,” in *Proceedings 2019 Network and Distributed System Security Symposium*. Internet Society, Feb. 2019. doi: <https://doi.org/10.14722/ndss.2019.23386>.
- [169] S. Miano, M. Bertrone, F. Risso, M. Tumolo, and M. V. Bernal, “Creating Complex Network Services with eBPF: Experience and Lessons Learned,” in *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*. IEEE, Sep. 2018, pp. 1–8. doi: <https://doi.org/10.1109/HPSR.2018.8850758>.
- [170] B. Gbadamosi, L. Leonardi, T. Pulls, T. Høiland-Jørgensen, S. Ferlin-Reiter, S. Sorce, and A. Brunström, “The eBPF Runtime in the Linux Kernel,” 2024. doi: <https://doi.org/10.48550/arXiv.2410.00026>., Accessed on 23 Apr 2025.
- [171] C. Liu, B. Tak, and L. Wang, “Understanding Performance of eBPF Maps,” in *Proceedings of the ACM SIGCOMM 2024 Workshop on EBPF and Kernel Extensions*. Association for Computing Machinery (ACM), 2024, p. 9–15. doi: <https://doi.org/10.1145/3672197.3673430>.
- [172] “DNSFilter’s Annual Security Report Reveals Worrisome Spike in Malicious DNS Requests,” DNSFilter, Jan. 2025, Accessed on 28 Dec 2025. [Online]. Available: <https://www.dnsfilter.com/newsroom/2025-annual-security-report-reveals-worrisome-spike-in-malicious-dns-requests>

А. ЛИСТИНЗИ ЗА *Netronome LISPP*

```
1 if (standard_metadata.instance_type == PKT_INSTANCE_TYPE_NORMAL) {
2     // novi paket stigao po prvi put
3     meta.ff3_1.state = ...; // inicijalizacija ff3-1 stanja (a ++ b)
4     meta.ff3_1.round = 1;
5     meta.help.phase = PHASE_FF3_1_PROLOG;
6 }
7
8 if (meta.help.phase == PHASE_FF3_1_PROLOG) {
9     // prvi deo (pre poziva aes sifrovanja) ff3-1 runde
10    meta.ff3_1.state = ...; // azuriranje ff3-1 stanja
11    meta.aes.state = ...; // priprema ulaza za aes
12    meta.aes.index = ...; // prvi aes index
13    meta.aes.round = 1;
14    meta.help.phase = PHASE_AES;
15    resubmit(meta);
16 } else if (meta.help.phase == PHASE_AES) {
17     // jedan deo (oznaceno sa meta.aes.index) jedne aes runde
18    meta.aes.state = ...; // azuriranje aes stanja
19    meta.aes.index = ...; // naredni aes index
20    meta.aes.round = ...; // meta.aes.round ili meta.aes.round + 1
21    meta.help.phase = ...; // PHASE_AES ili PHASE_FF3_1_EPILOGUE
22    resubmit(meta);
23 } else if (meta.help.phase == PHASE_FF3_1_EPILOGUE) {
24     // drugi deo (posle poziva aes sifrovanja) ff3-1 runde
25    bit<W_128> aes_output = meta.aes.state;
26    meta.ff3_1.state = ...; // azuriranje ff3-1 stanja
27    if (meta.ff3_1.round < FF3_1_ROUNDS) {
28        meta.ff3_1.round = meta.ff3_1.round + 1;
29        meta.help.phase = PHASE_FF3_1_PROLOG;
30        resubmit(meta);
31    } else {
32        bit<W_N> ff3_1_output = meta.ff3_1.state;
33    }
34 }
```

Листинг А.1: Вишеструко враћање истог пакета кроз ток обраде

```

1 // makro za generisanje definicije mask_key akcije
2 #define GENERATE_ACTION_MASK_KEY() \
3     action mask_key(bit<W_128> subkey) \
4     { \
5         meta.aes.state = meta.aes.state ^ subkey; \
6     }
7
8 // makro za generisanje definicije mask_key_round tabela
9 #define GENERATE_TABLE_MASK_KEY(ROUND, SUBKEY) \
10     table mask_key_round##ROUND##_table \
11     { \
12         actions = { mask_key; } \
13         default_action = mask_key(SUBKEY); \
14     }
15
16 // makro za primenu mask_key_round tabele
17 #define APPLY_MASK_KEY(ROUND) mask_key_round##ROUND##_table.apply();

```

Листинг А.2: Примена *AES* кључа рунде помоћу наменских *P4* табела

```

1 typedef struct
2 {
3     uint32_t w0;
4     uint32_t w1;
5     uint32_t w2;
6     uint32_t w3;
7 } uint128_t;

```

Листинг А.3: Експлицитна дефиниција типа uint128_t у виду структуре

```

1 __declspec(local_mem shared)
2 static uint128_t const aes_key_expanded[11] = {...};
3 __declspec(local_mem shared)
4 static uint32_t const lut_0[256] = {...};
5 __declspec(cls shared)
6 static uint32_t const lut_1[256] = {...};
7 __declspec(cls shared)
8 static uint32_t const lut_2[256] = {...};
9 __declspec(cls shared)
10 static uint32_t const lut_3[256] = {...};
11 __declspec(cls shared)
12 static uint32_t const lut_sbox[256] = {...};
13
14 uint128_t
15 aes_encrypt(__declspec(gp_reg) uint128_t state)
16 {
17     __declspec(gp_reg) uint128_t tmp = state;
18     __declspec(gp_reg) uint32_t round = 0;
19
20     state = apply_mask_key(tmp, aes_key_expanded[round++]);
21
22     for (; round < AES_ROUNDS; ++round) {
23         tmp = apply_lut_std(state);
24         state = apply_mask_key(tmp, aes_key_expanded[round]);
25     }
26
27     tmp = apply_lut_end(state);
28     state = apply_mask_key(tmp, aes_key_expanded[round]);
29
30     return state;
31 }

```

Листинг А.4: Шифровање *AES* алгоритмом помоћу предшифрованих табела

```

1 #include <pif_plugin.h>
2
3 int
4 pif_plugin_aes_encrypt(EXTRACTED_HEADERS_T* headers,
5                       MATCH_DATA_T* data)
6 {
7     uint32_t r0 = pif_plugin_meta_get__aes__t0(headers);
8     uint32_t r1 = pif_plugin_meta_get__aes__t1(headers);
9     uint32_t r2 = pif_plugin_meta_get__aes__t2(headers);
10    uint32_t r3 = pif_plugin_meta_get__aes__t3(headers);
11
12    uint128_t aes_state = { r0, r1, r2, r3 };
13
14    aes_state = aes_encrypt(aes_state);
15
16    r0 = aes_state.w0;
17    r1 = aes_state.w1;
18    r2 = aes_state.w2;
19    r3 = aes_state.w3;
20
21    pif_plugin_meta_set__aes__r0(headers, r0);
22    pif_plugin_meta_set__aes__r1(headers, r1);
23    pif_plugin_meta_set__aes__r2(headers, r2);
24    pif_plugin_meta_set__aes__r3(headers, r3);
25
26    return PIF_PLUGIN_RETURN_FORWARD;
27 }

```

Листинг А.5: Дефиниција екстерне *P4* функције на *Micro-C* језику

```

1 __declspec(cls shared)
2 static uint32_t const reverser_lut[256] = {
3     0x00, 0x80, 0x40, 0xc0, 0x20, 0xa0, 0x60, 0xe0,
4     0x10, 0x90, 0x50, 0xd0, 0x30, 0xb0, 0x70, 0xf0,
5     // skraceno radi konciznosti ...
6     0x0f, 0x8f, 0x4f, 0xcf, 0x2f, 0xaf, 0x6f, 0xef,
7     0x1f, 0x9f, 0x5f, 0xdf, 0x3f, 0xbf, 0x7f, 0xff};
8
9 static uint32_t
10 reverse_bits_32(__declspec(gp_reg) uint32_t param)
11 {
12     __declspec(gp_reg) uint32_t reverse_param = 0;
13     reverse_param = reverser_lut[param & 0xFF] << 0x18 |
14                 reverser_lut[(param >> 0x08) & 0xFF] << 0x10 |
15                 reverser_lut[(param >> 0x10) & 0xFF] << 0x08 |
16                 reverser_lut[(param >> 0x18) & 0xFF];
17     return reverse_param;
18 }

```

Листинг А.6: Дефиниција *bit-reversal* функције за 32-битне речи

Б. Листинзи за *eBPF LISPP*

```
1 // kursor zaglavlja radi pracenja tekuce pozicije parsiranja
2 struct hdr_cursor
3 {
4     void* pos;
5 };
6
7 SEC("xdp/lispp-if-encrypt")
8 int
9 lispp_if_encrypt(struct xdp_md* ctx)
10 {
11     void* data_end = (void*)(long)ctx->data_end;
12     void* data = (void*)(long)ctx->data;
13
14     struct hdr_cursor next_hdr = { .pos = data };
15     int status;
16
17     struct ethhdr* eth_hdr;
18     status = parse_ethhdr(&next_hdr, data_end, &eth_hdr);
19     if (status >= 0 && eth_hdr->h_proto == bpf_htons(ETH_P_IP)) {
20         struct iphdr* ip_hdr;
21         status = parse_iphdr(&next_hdr, data_end, &ip_hdr);
22         if (status >= 0 && ip_hdr->protocol == IPPROTO_TCP) {
23             struct tcphdr* tcp_hdr;
24             status = parse_tcphdr(&next_hdr, data_end, &tcp_hdr);
25             if (status >= 0) {
26                 // <lispp-obrada-paketa>
27             }
28         }
29     }
30
31     return XDP_PASS;
32 }
```

Листинг Б.1: Програм за шифровање од стране *eBPF LISPP* система

```

1  uint32_t const src_addr = bpf_ntohl(ip_hdr->saddr);
2  uint16_t const src_port = bpf_ntohs(tcp_hdr->source);
3
4  uint128_t const ff3_1_input = get_ff3_1_input(
5      (uint128_t){ 0x00000000, 0x00000000, 0x00000000, src_addr },
6      src_port,
7      (W_N));
8
9  uint128_t const ff3_1_output = ff3_1_encrypt(ff3_1_input);
10
11 uint32_t const affected_mask = (1 << ((W_N)-16)) - 1;
12
13 uint32_t const new_src_addr =
14     (src_addr & ~affected_mask) |
15     ((ff3_1_output.w2 << 16 | ff3_1_output.w3 >> 16) & affected_mask);
16 uint16_t const new_src_port = ff3_1_output.w3 & 0xFFFF;
17
18 ip_hdr->saddr = bpf_htonl(new_src_addr);
19 tcp_hdr->source = bpf_htons(new_src_port);
20
21 ip_hdr->check = bpf_htons(csum_ipv4(ip_hdr, data_end));
22 tcp_hdr->check =
23     bpf_htons(csum_tcp_incremental_ipv4(src_addr,
24                                         src_port,
25                                         bpf_ntohs(tcp_hdr->check),
26                                         new_src_addr,
27                                         new_src_port));
28
29 return bpf_redirect_map(&decrypt_if_idx, 0, 0);

```

Листинг Б.2: Замагљивање поља *IPv4* пакета од стране *eBPF LISPP* система

```

1  struct
2  {
3      __uint(type, BPF_MAP_TYPE_DEVMAP);
4      __uint(max_entries, 1);
5      __type(key, uint32_t);
6      __type(value, uint32_t);
7  } decrypt_if_idx SEC(".maps");

```

Листинг Б.3: *eBPF* мапа за преусмеравање пакета обрађених од стране *eBPF LISPP* система

```

1  struct vlan_hdr
2  {
3      __be16 h_vlan_TCI;
4      __be16 h_vlan_encapsulated_proto;
5  };
6
7  struct vlan_ids
8  {
9      uint16_t id[VLAN_MAX_DEPTH];
10 };

```

Листинг Б.4: Помоћне структуре за парсирање *Vlan* заглавља у *XDP eBPF* програму

```

1 #define VLAN_MAX_DEPTH 2
2 #define VLAN_VID_MASK 0x0FFF
3
4 int
5 proto_is_vlan(uint16_t h_proto)
6 {
7     return !!(h_proto == bpf_htons(ETH_P_8021Q) ||
8             h_proto == bpf_htons(ETH_P_8021AD));
9 }
10
11 int
12 parse_ethhdr_vlan(struct hdr_cursor* next_hdr,
13                  void* data_end,
14                  struct ethhdr** eth_hdr_out,
15                  struct vlan_ids* vlans)
16 {
17     struct ethhdr* eth_hdr = next_hdr->pos;
18     if ((void*)(eth_hdr + 1) > data_end) {
19         return -1;
20     }
21
22     next_hdr->pos = eth_hdr + 1;
23     *eth_hdr_out = eth_hdr;
24
25     struct vlan_hdr* vlan_hdr = next_hdr->pos;
26     uint16_t h_proto = eth_hdr->h_proto;
27
28     for (int i = 0; i < VLAN_MAX_DEPTH; i++) {
29         if (!proto_is_vlan(h_proto)) {
30             break;
31         }
32         if ((void*)(vlan_hdr + 1) > data_end) {
33             break;
34         }
35
36         h_proto = vlan_hdr->h_vlan_encapsulated_proto;
37
38         if (vlans) {
39             vlans->id[i] =
40                 (bpf_ntohs(vlan_hdr->h_vlan_TCI) & VLAN_VID_MASK);
41         }
42
43         vlan_hdr++;
44     }
45
46     next_hdr->pos = vlan_hdr;
47     return h_proto; // mrezní format (big-endian)
48 }

```

Листинг Б.5: Парсирање *Vlan* заглавља у *XDP eBPF* програму

```

1 int
2 parse_ethhdr(struct hdr_cursor* next_hdr,
3             void* data_end,
4             struct ethhdr** eth_hdr_out)
5 {
6     return parse_ethhdr_vlan(next_hdr, data_end, eth_hdr_out, NULL);
7 }

```

Листинг Б.6: Парсирање заглавља *Ethernet* оквира у *XDP eBPF* програму

```

1 int
2 parse_iphdr(struct hdr_cursor* next_hdr,
3            void* data_end,
4            struct iphdr** ip_hdr_out)
5 {
6     struct iphdr* ip_hdr = next_hdr->pos;
7     if ((void*)(ip_hdr + 1) > data_end) {
8         return -1;
9     }
10    int ip_hdr_len = ip_hdr->ihl * 4;
11    if (ip_hdr_len < sizeof(*ip_hdr)) {
12        return -1;
13    }
14    if (next_hdr->pos + ip_hdr_len > data_end) {
15        return -1;
16    }
17    next_hdr->pos += ip_hdr_len;
18    *ip_hdr_out = ip_hdr;
19    return ip_hdr->protocol;
20 }

```

Листинг Б.7: Парсирање заглавља *IPv4* пакета у *XDP eBPF* програму

```

1 int
2 parse_tcphdr(struct hdr_cursor* next_hdr,
3             void* data_end,
4             struct tcphdr** tcp_hdr_out)
5 {
6     struct tcphdr* tcp_hdr = next_hdr->pos;
7     if ((void*)(tcp_hdr + 1) > data_end) {
8         return -1;
9     }
10    int tcp_hdr_len = tcp_hdr->doff * 4;
11    if (tcp_hdr_len < sizeof(*tcp_hdr)) {
12        return -1;
13    }
14    if (next_hdr->pos + tcp_hdr_len > data_end) {
15        return -1;
16    }
17    next_hdr->pos += tcp_hdr_len;
18    *tcp_hdr_out = tcp_hdr;
19    return tcp_hdr_len;
20 }

```

Листинг Б.8: Парсирање заглавља *TCP* сегмента у *XDP eBPF* програму

```

1  __always_inline uint16_t
2  csum_ipv4(struct iphdr* const ip_hdr, void* const data_end)
3  {
4      uint16_t* const ip_hdr_as_shorts = (uint16_t* const)ip_hdr;
5      uint8_t const ip_hdr_len_in_shorts = ip_hdr->ihl * 2;
6
7      uint32_t csum = 0;
8      for (uint8_t i = 0; i < ip_hdr_len_in_shorts; i++) {
9          if (i != 5 && ((void*)(ip_hdr_as_shorts + i + 1) <= data_end)) {
10             csum += bpf_ntohs(ip_hdr_as_shorts[i]);
11         }
12     }
13
14     uint16_t csum_carry = (csum & 0xFFFF) + ((csum >> 16) & 0x000F);
15     uint16_t csum_compl = (~csum_carry) & 0xFFFF;
16
17     return csum_compl;
18 }

```

Листинг Б.9: Израчунавање *IPv4* контролне суме

```

1  __always_inline uint16_t
2  csum_tcp_incremental_ipv4(uint32_t old_addr,
3                          uint16_t old_port,
4                          uint16_t old_csum,
5                          uint32_t new_addr,
6                          uint16_t new_port)
7  {
8      uint32_t new_csum;
9
10     new_csum = ~old_csum - ((old_addr >> 16) & 0xFFFF) -
11                 (old_addr & 0xFFFF) - old_port;
12     new_csum = (new_csum & 0xFFFF) + (new_csum >> 16);
13
14     new_csum = new_csum + ((new_addr >> 16) & 0xFFFF) +
15                 (new_addr & 0xFFFF) + new_port;
16     new_csum = (new_csum & 0xFFFF) + (new_csum >> 16);
17
18     return (uint16_t)((~new_csum) & 0xFFFF);
19 }

```

Листинг Б.10: Инкрементално израчунавање *TCP* контролне суме

В. Делови *DNS* лога

```
1 28-May-2024 06:01:03.410 RQ
2     147.91.1.6:50783 -> 203.241.135.111:53 UDP 64b
3     eu-auth2.samsungosp.com/IN/A
4 28-May-2024 06:01:03.460 RQ
5     147.91.1.6:53696 -> 211.189.122.246:53 UDP 64b
6     eu-auth2.samsungosp.com/IN/A
7 28-May-2024 06:01:03.770 RR
8     147.91.1.6:53696 <- 211.189.122.246:53 UDP 220b
9     eu-auth2.samsungosp.com/IN/A
```

Листинг В.1: Део *DNS* лога са истовременим слањем упита ка већем броју сервера

```

1 28-May-2024 13:29:31.497 RQ
2    2001:4170:0:1::6:42842 -> 2001:502:f3ff::f6:53 UDP 72b
3    mclb-gcp.nimbus.bitdefender.net/IN/AAAA
4 28-May-2024 13:29:32.297 RQ
5    147.91.1.6:46956 -> 156.154.66.210:53 UDP 72b
6    mclb-gcp.nimbus.bitdefender.net/IN/AAAA
7 28-May-2024 13:29:33.097 RQ
8    147.91.1.6:46343 -> 156.154.65.210:53 UDP 72b
9    mclb-gcp.nimbus.bitdefender.net/IN/AAAA
10 28-May-2024 13:29:33.897 RQ
11   147.91.1.6:42912 -> 156.154.67.210:53 UDP 72b
12   mclb-gcp.nimbus.bitdefender.net/IN/AAAA
13 28-May-2024 13:29:34.698 RQ
14   147.91.1.6:38122 -> 156.154.64.210:53 UDP 72b
15   mclb-gcp.nimbus.bitdefender.net/IN/AAAA
16 28-May-2024 13:29:35.498 RQ
17   2001:4170:0:1::6:44131 -> 2610:a1:1015::f6:53 UDP 72b
18   mclb-gcp.nimbus.bitdefender.net/IN/AAAA
19 28-May-2024 13:29:36.298 RQ
20   2001:4170:0:1::6:58523 -> 2610:a1:1014::f6:53 UDP 72b
21   mclb-gcp.nimbus.bitdefender.net/IN/AAAA
22 28-May-2024 13:29:37.099 RQ
23   2001:4170:0:1::6:38922 -> 2001:502:4612::f6:53 UDP 49b
24   mclb-gcp.nimbus.bitdefender.net/IN/AAAA
25 28-May-2024 13:29:37.899 RQ
26   147.91.1.6:58941 -> 156.154.67.210:53 UDP 49b
27   mclb-gcp.nimbus.bitdefender.net/IN/AAAA
28 28-May-2024 13:29:38.699 RQ
29   147.91.1.6:47949 -> 156.154.66.210:53 UDP 49b
30   mclb-gcp.nimbus.bitdefender.net/IN/AAAA
31 28-May-2024 13:29:39.499 RQ
32   147.91.1.6:40846 -> 156.154.65.210:53 UDP 49b
33   mclb-gcp.nimbus.bitdefender.net/IN/AAAA
34 28-May-2024 13:29:40.299 RQ
35   2001:4170:0:1::6:54094 -> 2001:502:f3ff::f6:53 UDP 49b
36   mclb-gcp.nimbus.bitdefender.net/IN/AAAA
37 28-May-2024 13:29:41.099 RQ
38   2001:4170:0:1::6:53727 -> 2001:502:4612::f6:53 UDP 49b
39   mclb-gcp.nimbus.bitdefender.net/IN/AAAA
40 28-May-2024 13:30:05.844 RQ
41   147.91.1.6:34403 -> 156.154.67.210:53 UDP 72b
42   mclb-gcp.nimbus.bitdefender.net/IN/AAAA
43 28-May-2024 13:30:06.644 RQ
44   147.91.1.6:51384 -> 156.154.65.210:53 UDP 72b
45   mclb-gcp.nimbus.bitdefender.net/IN/AAAA
46 28-May-2024 13:30:07.444 RQ
47   2001:4170:0:1::6:53694 -> 2001:502:f3ff::f6:53 UDP 72b
48   mclb-gcp.nimbus.bitdefender.net/IN/AAAA
49 28-May-2024 13:30:07.619 RR
50   2001:4170:0:1::6:53694 <- 2001:502:f3ff::f6:53 UDP 221b
51   mclb-gcp.nimbus.bitdefender.net/IN/AAAA

```

Листинг В.2: Део DNS лога са вишеструким поновним покушајем слања упита

Биографија аутора

Марко Мићовић рођен је 3. јула 1993. године у Београду. Основну школу Стеван Синђелић завршио је као носилац Вукове дипломе. Трећу београдску гимназију, на природно-математичком смеру, завршио је са одличним успехом.

Основне академске студије на Електротехничком факултету у Београду, на одсеку Електротехника и рачунарство, уписао је 2012. године. Основне академске студије на модулу Рачунарска техника и информатика завршио је 2016. године са просечном оценом 9,04. Дипломски рад на тему „Систем за идентификацију студената употребом бесконтактних картица” под менторством др Милоша Цветановића, ванредног професора, одбранио је са оценом 10. Током основних студија био је на стручној пракси у фирми Микроелектроника.

Мастер академске студије на Електротехничком факултету у Београду уписао је 2016. године. Мастер академске студије на модулу Рачунарска техника и информатика завршио је 2018. године са просечном оценом 10,00. Мастер рад на тему „Окружење за прикупљање информација о извршавању програма” под менторством др Саше Стојановића, ванредног професора, одбранио је са оценом 10.

Докторске академске студије на Електротехничком факултету у Београду уписао је 2018. године на модулу Рачунарска техника и информатика. Положио је све испите са просечном оценом 10,00. У истраживачком раду оријентисао се ка областима уграђених уређаја и рачунарских мрежа. Похађао је летњу школу *Advanced Computer Architecture and Compilation for High-performance Embedded Systems (ACACES)* 2019. године. Током мастер и докторских академских студија има објављена 2 научна рада у часопису са *SCI* листе, 9 научних радова на страним конференцијама и 11 научних радова на домаћим конференцијама као аутор или коаутор.

Почев од децембра 2016. године запослен је Електротехничком факултету у Београду. У периоду од 2016. до 2018. године био је изабран у звање сарадник у настави. Почев од 2018. године запослен је као асистент на истом факултету и тренутно је ангажован на предметима: Рачунарске мреже 1, Рачунарске мреже 2, Микропроцесорски системи, Оперативни системи 1, Системски софтвер и Програмирање мобилних уређаја. Био је на стручном усавршавању у фирми *Elsys Eastern Europe*.

За време периода у току којег је запослен на Електротехничком факултету учествује на више комерцијалних пројеката. Учествовао је као предавач у програму преквалификација у области информационих технологија који је организовао Развојни програм Уједињених нација током 2019, 2021, 2022. године. Сарађивао је у склопу комерцијалних пројеката са неколико фирми као и колегама са других катедри Електротехничког факултета. Најистакнутији такав пројекат јесте „Развој и реализација софтвера за прорачун звучне изолације” у сарадњи са фирмом *URSA*. Учесник је или је био учесник на неколико научних пројеката: (1) „Развој дигиталних технологија и умрежених сервиса у системима са уграђеним електронским компонентама” који финансира Министарство просвете, науке и технолошког развоја, (2) „Иновација групе предмета из области рачунарских мрежа, интернета и заштите података”, (3) „УНАР - Унапређење наставе из Архитектуре рачунара” које је финансирало Министарство просвете, науке и технолошког развоја, (4) „*Advancing novel textual similarity-based solutions in software development (AVANTES)*” који је финансирао Фонд за науку Републике Србије, (5) „*Belgrade Data Innovation Hub (BELDIH) - HORIZON 2020*” и (6) „*Software for Text Offenses Prevention in Serbian: AI-driven Hate Speech Detection (STOP)*” који је финансирао Фонд за науку Републике Србије.

Изјава о ауторству

Име и презиме аутора: Марко Мићовић

Број индекса: 2018/5013

Изјављујем

да је докторска дисертација под насловом

Заштита приватности на мрежном слоју

применом шифровања са очувањем формата

- резултат сопственог истраживачког рада;
- да дисертација ни у целини ни у деловима није била предложена за стицање друге дипломе према студијским програмима других високошколских установа;
- да су резултати коректно наведени и
- да нисам кршио ауторска права и користио интелектуалну својину других лица.

У Београду,

25.02.2026.

Потпис аутора

Марко Мићовић

Изјава о истоветности штампане и електронске верзије докторског рада

Име и презиме аутора: Марко Мићовић
Број индекса: 2018/5013
Студијски програм: Рачунарска техника и информатика
Наслов рада: Заштита приватности на мрежном слоју
применом шифровања са очувањем формата
Ментор: проф. др Павле Вулетић

Изјављујем да је штампана верзија мог докторског рада истоветна електронској верзији коју сам предао ради похрањивања у **Дигиталном репозиторијуму Универзитета у Београду**.

Дозвољавам да се објаве моји лични подаци везани за добијање академског назива доктора наука, као што су име и презиме, година и место рођења и датум одбране рада.

Ови лични подаци могу се објавити на мрежним станицама дигиталне библиотеке, у електронском каталогу и у публикацијама Универзитета у Београду.

У Београду,

25.02.2026.

Потпис аутора

Марко Мићовић

Изјава о коришћењу

Овлашћујем Универзитетску библиотеку „Светозар Марковић” да у Дигитални репозиторијум Универзитета у Београду унесе моју докторску дисертацију под насловом:

Заштита приватности на мрежном слоју

применом шифровања са очувањем формата

која је моје ауторско дело.

Дисертацију са свим прилозима предао сам у електронском формату погодном за трајно архивирање.

Моју докторску дисертацију похрањену у Дигиталном репозиторијуму Универзитета у Београду и доступну у отвореном приступу могу да користе сви који поштују одредбе садржане у одабраном типу лиценце Креативне заједнице (*Creative Commons*) за коју сам се одлучио.

- 1 Ауторство (CC BY)
- 2 Ауторство – некомерцијално (CC BY-NC)
- 3 Ауторство – некомерцијално – без прерада (CC BY-NC-ND)
- 4 Ауторство – некомерцијално – делити под истим условима (CC BY-NC-SA)
- 5 Ауторство – без прерада (CC BY-ND)
- 6 Ауторство – делити под истим условима (CC BY-SA)

Молимо да заокружите само једну од шест понуђених лиценци.
Кратак опис лиценци је саставни део ове изјаве.

У Београду,

25.02.2026.

Потпис аутора

Марко Митровић

1. **Ауторство.** Дозвољаваате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце, чак и у комерцијалне сврхе. Ово је најслободнија од свих лиценци.
2. **Ауторство – некомерцијално.** Дозвољаваате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца не дозвољава комерцијалну употребу дела.
3. **Ауторство – некомерцијално – без прерада.** Дозвољаваате умножавање, дистрибуцију и јавно саопштавање дела, без промена, преобликовања или употребе дела у свом делу, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца не дозвољава комерцијалну употребу дела. У односу на све остале лиценце, овом лиценцом се ограничава највећи обим права коришћења дела.
4. **Ауторство – некомерцијално – делити под истим условима.** Дозвољаваате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце и ако се прерада дистрибуира под истом или сличном лиценцом. Ова лиценца не дозвољава комерцијалну употребу дела и прерада.
5. **Ауторство – без прерада.** Дозвољаваате умножавање, дистрибуцију и јавно саопштавање дела, без промена, преобликовања или употребе дела у свом делу, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца дозвољава комерцијалну употребу дела.
6. **Ауторство – делити под истим условима.** Дозвољаваате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце и ако се прерада дистрибуира под истом или сличном лиценцом. Ова лиценца дозвољава комерцијалну употребу дела и прерада. Слична је софтверским лиценцама, односно лиценцама отвореног кода.