

УНИВЕРЗИТЕТ У БЕОГРАДУ
ЕЛЕКТРОТЕХНИЧКИ ФАКУЛТЕТ



Анализа пост-квантних асиметричних криптографских алгоритама

Мастер рад

Ментор:

др Павле Вулетић, ванр. проф.

Кандидат:

Алекса Новаковић 3162/2021

Београд, август 2022.

САДРЖАЈ

САДРЖАЈ	2
1. УВОД.....	3
2. ПРОБЛЕМ РАЗМЕНЕ КЉУЧЕВА.....	4
3. ОСНОВЕ АСИМЕТРИЧНЕ КРИПТОГРАФИЈЕ.....	7
3.1. <i>ONE-WAY TRAP-DOOR</i> ФУНКЦИЈЕ	8
3.2. ДЕФИНИЦИЈА АСИМЕТРИЧНОГ АЛГОРИТМА.....	9
3.3. ПРИМЕНА АСИМЕТРИЧНИХ АЛГОРИТАМА.....	10
3.4. КРИПТОАНАЛИЗА АСИМЕТРИЧНИХ АЛГОРИТАМА.....	12
4. ПРЕГЛЕД СТАНДАРДНИХ АСИМЕТРИЧНИХ АЛГОРИТАМА.....	16
4.1. <i>DIFFIE-HELLMAN</i> И <i>ELGAMAL</i> АЛГОРИТМИ	16
4.2. <i>RSA</i> АЛГОРИТАМ.....	18
4.3. АЛГОРИТМИ ЗАСНОВАНИ НА <i>ECC</i>	20
5. КОНЦЕПТ КВАНТНИХ РАЧУНАРА И ШОРОВ АЛГОРИТАМ.....	25
6. ПОСТ-КВАНТНИ АСИМЕТРИЧНИ АЛГОРИТМИ	30
6.1. <i>CLASSIC McELIECE</i>	30
6.2. <i>CRYSTALS-KYBER</i>	34
6.3. <i>NTRU</i>	38
6.4. <i>SABER</i>	40
7. АПЛИКАЦИЈА ЗА ТЕСТИРАЊЕ ПЕРФОРМАНСИ АСИМЕТРИЧНИХ АЛГОРИТАМА.....	44
7.1. ОПИС ФУНКЦИОНАЛНОСТИ	44
7.2. ИМПЛЕМЕНТАЦИОНИ ДЕТАЉИ	45
7.3. ПОДРЖАНИ АЛГОРИТМИ.....	45
7.4. СТРУКТУРА АПЛИКАЦИЈЕ	46
7.5. ИНСТРУКЦИЈЕ ЗА ПРЕВОЂЕЊЕ И ПОКРЕТАЊЕ АПЛИКАЦИЈЕ.....	48
7.6. РЕЗУЛТАТИ РАДА АПЛИКАЦИЈЕ	49
8. ЗАКЉУЧАК.....	56
ЛИТЕРАТУРА.....	57
СПИСАК СКРАЋЕНИЦА	60
СПИСАК СЛИКА.....	62
СПИСАК ТАБЕЛА.....	63
А. УВОД У МАТЕМАТИКУ РЕШЕТКИ	64

1. Увод

Асиметрични алгоритми и њихове функционалности енкрипције и потписивања представљају изузетно битан концепт у оквиру криптографије. У оквиру концепта *CIA* тријаде (енг. *Confidentiality, Integrity, and Availability* – тајност, интегритет и доступност) информационе безбедности, примена асиметричних алгоритама је често неизбежна ради остваривања својстава тајности и интегритета информација. Поменути својствима се често додаје и својство непорецивости (енг. *non-repudiation*). Асиметрични алгоритми налазе примену у размени кључева који се користе у оквиру симетричних алгоритама за енкрипцију, аутентификацији учесника у комуникацији, ради провере интегритета и дигиталног потписивања докумената, у дигиталним сертификатима итд.

Асиметрични алгоритми, због својих специфичних особина се користе у великом броју апликација и протокола: *SSL/TLS* (протокола који се користе ради остваривања сигурне конекције између веб клијента и сервера), *S/MIME* (протокола који се користи за енкрипцију и дигитално потписивање електронске поште), *IPsec* (протокола за заштиту *IP* пакета), *PGP* (софтвера који служи за енкрипцију и дигитално потписивање електронске поште, фајлова, текстуалних порука и партиција на дисковима), *OTR* (протокола који се користи за сигурност порука које се размењују у оквиру *instant messaging* клијентских програма) итд.

Будући да сигурност великог броја система почива на сигурности асиметричних алгоритама, од великог је интереса њихова константна ревизија и анализа, како би била гарантована безбедност система и података са којима они раде. Развој квантних рачунара и алгоритама који их користе временом прети да поништи сигурност сваког асиметричног алгорита који је тренутно у широкој употреби за релативно кратко време. Управо због тога је амерички национални институт за стандарде и технологију (енг. *National Institute of Standards and Technology - NIST*) најавио децембра 2016. године процес стандардизације алгоритама пост-квантне криптографије.

У оквиру рада биће изложена анализа тренутно најкоришћенијих асиметричних алгоритама који се користе за енкрипцију и размену кључева (*RSA*, алгоритми размене кључева засновани на *Diffie-Hellman* алгоритму и алгоритми који користе *ECC*), њихове слабости против квантних рачунара и анализа пост-квантних алгоритама за размену кључева и енкрипцију јавним кључем који су финалисти треће фазе *NIST*-овог процеса стандардизације (*Classic McEliece*, *CRYSTALS-Kyber*, *NTRU* и *Saber*). Анализа перформанси и меморијских захтева класичних и пост-квантних асиметричних алгоритама биће приказана кроз имплементацију апликације која користи постојеће имплементације алгоритама помоћу библиотеке отвореног кода.

У оквиру другог поглавља дат је опис проблема размене кључева. У оквиру трећег поглавља дат је приказ основа асиметричне криптографије. У четвртном поглављу дат је преглед стандардних и тренутно коришћених асиметричних алгоритама. Пето поглавље даје увид у принципе функционисања квантних рачунара и Шоровог алгорита. У шестом поглављу дата је анализа пост-квантних алгоритама. Седмо поглавље описује апликацију за поређење перформанси између асиметричних алгоритама и њене резултате. У оквиру осмог поглавља дат је закључак, док је у прилогу дат увод у математику решетки.

2. Проблема размене кључева

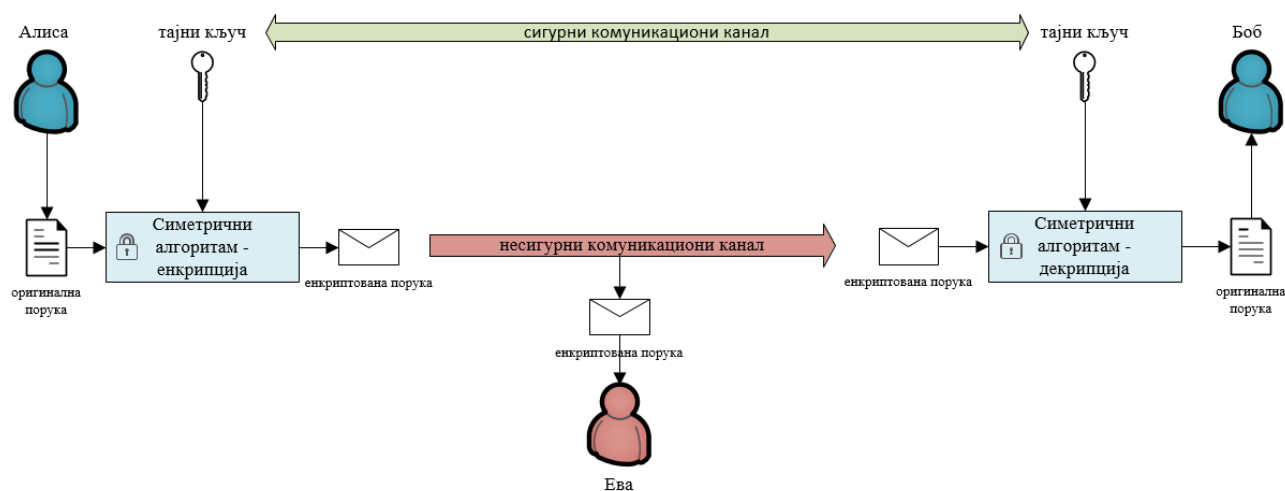
Једни од кључних проблема који су се јавили у оквиру развоја криптографије представљају проблем размене тајних кључева за симетричне алгоритме, којим се обезбеђује својство тајности порука које се преносе, као и проблем увођења функционалности дигиталних потписа, који би такође представљали врло битан концепт за ширу употребу криптографије, обезбеђивањем својства интегритета и непорецивости. Будући да се овај рад бави искључиво асиметричним алгоритмима који се користе за размену кључева и енкрипцију, биће узет у разматрање и анализиран искључиво проблем размене тајних кључева.

Симетрични алгоритам представља пар $\varepsilon = (E, D)$, који је дефинисан над простором (K, M, C) , где су E и D функције које се користе за енкрипцију и декрипцију, док K , M и C представљају скупе свих могућих кључева, порука и шифрованих порука, респективно. Функција за енкрипцију E као улазне податке добија поруку коју је потребно шифровати $m \in M$, кључ $k \in K$, док као излаз добијамо шифровану поруку $c \in C$, што је приказано у облику формуле: $c = E(k, m)$. Функција за декрипцију представља инверзну функцију којом из шифроване поруке, користећи исти тајни кључ, добијамо почетну поруку, тј. важи: $m = D(k, c)$. Инверзно својство декрипционе функције се такође може приказати као једнакост: $D(k, E(k, m)) = m$. За успешно извршавање операција енкрипције и декрипције неопходно је коришћење истог тајног кључа, који мора да се генерише насумично из скупа свих могућих кључева. Симетрични алгоритми се деле на *block* и *stream* алгоритме у зависности од тога да ли податке обрађују у блоковима фиксне величине или бајт по бајт/бит по бит. Додатно је потребно нагласити да се често у оквиру алгоритма користи и тзв. иницијализациони вектор (енг. *initialization vector*, IV), који означава почетно стање и који за разлику од кључа не мора бити тајан, али мора да се такође генерише насумично. IV додатно појачава безбедност алгоритма и ентропију шифроване поруке и омогућава да исте поруке енкриптоване користећи исти кључ, са различитим IV , имају различите шифроване облике. У оквиру процеса енкрипције користе се различите операције над поруком, тајним кључем и IV – пермутација битова поруке на различите начине, супституција битова поруке другим битовима, као и комбиновање битова поруке, битова тајног кључа и битова IV на различите начине. Ове операције се најчешће извршавају у више итерација или рунди у циљу што комплексније повезаности битова оригиналне поруке са битовима тајног кључа и IV , ради елиминисања статистичких особина првобитне поруке. [1] [5]

Величина кључева који се користе у оквиру симетричног алгоритма зависе директно од самог алгоритма, режима рада и од нивоа захтеване сигурности. Коришћењем тајних кључева недовољне дужине отвара се пут за *brute-force* нападе на алгоритам, зато што се смањује сложеност испробавања свих могућих кључева. Пример коришћења изузетно великих кључева може се видети у оквиру *one time pad* алгоритма, који је општеприхваћен као један од безусловно сигурних алгоритма (што је доказано на основу Теорије информација [3]), без обзира колико рачунарских ресурса поседује потенцијални нападач (уколико се кључеви генеришу насумично и не користе поново). Код овог алгоритма енкрипциона и декрипциона функција имају исти облик – раде ексклузивно сабирање (*XOR*) над својим параметрима у битској репрезентацији, где дужина поруке мора да буде мања или једнака дужини кључа, што може довести до веома великих кључева. Наравно, овакав алгоритам је непрактичан за коришћење у оквиру већине апликација, већ се у пракси углавном користе симетрични алгоритми код којих је процес напада и поништавања сигурности самог алгоритма рачунарски и временски неприхватљив за потенцијалног нападача, док су функције које се користе за

енкрипцију и декрипцију рачунарски ефикасне, а тајни кључеви релативно мале величине. Тренутно најкоришћенији симетрични алгоритми као што су *AES* и *ChaCha* користе кључеве величине неколико стотина бита (најчешће 128 или 256).

Такође дефинишимо два ентитета – Алису и Боба, који желе да остваре сигурну размену порука преко несигурног медијума за пренос података, користећи симетрични алгоритам одговарајуће сигурности за енкрипцију порука, као и додатни ентитет – Еву, противника чији је циљ да прочита поруке које размењују Алиса и Боб.



Слика 2.1 – Блок дијаграм комуникације користећи симетрични алгоритам за енкрипцију порука

На слици 2.1 је приказан блок дијаграм комуникације између Алисе и Боба, где оба ентитета користе договорени симетрични алгоритам, са договореним параметрима и заједнички симетрични кључ за енкрипцију и декрипцију порука, који су разменили сигурним комуникационим каналом (који концептуално обично има много веће кашњење и много мањи доступан проток података од несигурног канала) претходно. На дијаграму је конкретно приказан процес слања поруке од Алисе ка Бобу, док би и у обрнутом смеру слања поруке дијаграм био еквивалентан, са заменом места актера у комуникацији. Уколико као почетну претпоставку усвојимо да је коришћени симетрични алгоритам сигуран, Ева неће моћи да дође до оригиналне поруке уколико је тајни кључ који се користи размењен на сигуран начин (такође се може сматрати да је симетричан алгоритам који користе Алиса и Боб доступан Еви).

Тајни кључ се може разменити на више начина, користећи релативно сигурне комуникационе канале – разменом кључева уживо, слањем кључа поштом, разменом кључа путем телефонског позива итд., у зависности од потребног степена сигурности. Овакви начини размене тајних кључева често имају велике трошкове у погледу времена и сигурности и нису увек примењиви. Уколико замислимо телекомуникациону мрежу од n ентитета, који желе да међусобно комуницирају, са одржавањем својства тајности, сваки ентитет би морао да чува $n - 1$ кључева ка осталим ентитетима, тј. уколико сматрамо да свака два ентитета за комуникацију користе заједнички тајни кључ, укупно би морало да се одради $\frac{n \cdot (n-1)}{2}$ претходно описаних размена кључева, што за велико n може бити неизводљиво и временски изузетно захтевно. Такође уколико би се временом, ради боље сигурности, радило генерисање нових тајних кључева и њихово слање несигурним каналом, енкриптовањем нових тајних

кључева са претходно коришћеним тајним кључевима, пожељни ниво сигурности не би био постигнут уколико је стари тајни кључ компромитован – нападач ће користећи њега доћи до новог тајног кључа и наставити да прислушкује комуникацију.

Решење поменутог проблема размене кључева је могуће постићи коришћењем само симетричних алгоритама, као што је дато у оквиру [4] – где је предложено коришћење ткзв. заштитног протокола (енг. *protective protocol*), где би у оквиру комуникационе мреже постојало m ентитета који би служили само за дистрибуцију тајних кључева за симетричне алгоритме. Сваки корисник у оквиру система би поседовао m јединствених тајних кључева за комуникацију са сваком од ентитета за дистрибуцију кључева, док би сваки ентитет за дистрибуцију кључева морао да има n јединствених тајних кључева за комуникацију са сваком од корисника. Уколико би корисници Алиса и Боб желели да успоставе комуникацију у оквиру мреже оваквог типа, прво би послали захтев за успостављање комуникације (који садржи детаље корисника који желе да остваре комуникацију). Затим би и Алиса и Боб добили одговор од сваког од m ентитета за дистрибуцију кључева у коме би се, енкриптован кључем који користе тај корисник и тај посматрани ентитет за дистрибуцију кључева, налазио генерисан тајни кључ од стране тог ентитета за дистрибуцију кључева (овај кључ је једнак и за Алису и за Боба). Затим би и Алиса и Боб одрадили *XOR* операцију над свим добијеним кључевима и на тај начин би добили једнаке тајне кључеве са којима би успешно започели сигурну комуникацију. Будући да би код протокола оваквог типа очигледна сигурносна претња била компромитација ентитета за дистрибуцију кључева, треба приметити да је за компромитацију процеса дистрибуције новог кључа нападачу неопходно да компромитује свим m ентитета за дистрибуцију кључева. Наравно, овакво решење не би било оптимално – будући да је неопходна почетна размена кључева сваког од корисника са сваком од ентитета за дистрибуцију кључева, као и прилично генерисаног саобраћаја и времена потребног за конструкцију кључа између клијената. Такође би постојала претња где би нападач потенцијално пробао да нападне ентитете за дистрибуцију кључева који нису под његовом контролом мрежним нападима и да их на тај начин учини недоступним, кад се детектује да је започет процес генерисања кључа или пре истог, што би му омогућило да дође до коначно генерисаног тајног кључа и да компромитује комуникацију између клијената.

У наставку рада аутори, Мартин Хелман и Витфилд Дифи, су такође назначили да би овако компликован протокол за размену симетричних кључева био беспотребан у случају развоја тада нове гране криптографије – асиметричне криптографије (или криптографије са јавним кључевима), чије су основе изложили у раду [2], који је објављен неколико месеци касније од рада [4] 1976. године, у коме су објавили основе првог асиметричног алгоритма – *Diffie-Hellman* алгоритма за размену кључева и на тај начин започели револуцију у оквиру развоја криптографије.

3. Основе асиметричне криптографије

Асиметрични алгоритми или алгоритми са јавним кључем представљају класу алгоритама који користе два математички повезана кључа у својим операцијама – приватни кључ, који се никад не открива другим странама, и јавни кључ, који је јавно доступан другим ентитетима и који други ентитети могу да користе на различите начине. Асиметрични алгоритми имају у потпуности другачију математичку основу која стоји иза њих у поређењу са симетричним алгоритмима. Најчешће се користе у оквиру хибридних шема у криптографским системима, заједно са симетричним алгоритмима. [5]

Постоји више различитих врста асиметричних алгоритама у зависности од њихове намене, тј. од њихових функционалности и проблема које решавају [5]:

- алгоритми који се користе за енкрипцију јавним кључем
- алгоритми који се користе за генерисање и проверу дигиталних потписа, где пошиљалац користи свој приватни кључ за потписивање поруке, над подацима поруке или над хеш функцијом која је извршена над подацима из поруке
- алгоритми за размену кључева, где обе стране користећи јавне кључеве од друге стране и својих приватних кључева долазе до заједничке дељене тајне (тајног кључа који се надаље користи за комуникацију путем симетричних алгоритама).

У оквиру табеле 3.1 налази се приказ често коришћених стандардних асиметричних алгоритама, заједно са њиховим функционалностима. Потребно је нагласити да сви алгоритми који подржавају енкрипцију користећи јавни кључ такође подржавају и размену кључева, док обрнуто не мора да буде случај. Неки алгоритми као што је *ECIES* представљају хибридне шеме за енкрипцију које у оквиру себе користе: хеш функције, алгоритам за договор кључа и симетричне алгоритме за енкрипцију података.

Алгоритам	Енкрипција са јавним кључем	Генерисање дигиталних потписа	Размена кључева
<i>RSA</i>	✓	✓	✓
<i>DSA</i>	×	✓	×
<i>Diffie-Hellman</i>	×	×	✓
<i>ElGamal encryption system</i>	✓	×	✓
<i>ElGamal signature scheme</i>	×	✓	×
<i>Elliptic Curve Diffie-Hellman</i>	×	×	✓
<i>ECDSA</i>	×	✓	×
<i>ECIES</i>	✓	×	✓

Табела 3.1 – Асиметрични алгоритми и њихове функционалности

3.1. *One-way trap-door* функције

One-way trap-door функције представљају математичку основу постојања асиметричних алгоритама.

Уколико дефинишемо X и Y као два ограничена скупа, функција $f: X \rightarrow Y$ представљаће *one-way* функцију уколико испуњава следеће захтеве [5][6]:

- за свако $x \in X$ постоји излазна вредност функције $y \in Y$, која се добија као $y = f(x)$, чији је процес рачунања ефикасан. Ово обично означава да алгоритам којим се рачуна $f(x)$ има полиномијалну временску сложеност реда $O(n^a)$, где n представља битску дужину улазне променљиве x , док a представља константу која означава степен полиномијалне сложености за посматрани алгоритам рачунања излазне вредности, тј. проблем рачунања излазне вредности у припада P класи комплексности у којој се налазе проблеми који могу да се реше у полиномијалном времену од стране детерминистичке Тјурингове машине.
- да постоји инверзна функција f^{-1} од функције f која пресликава излазни податак $y \in Y$ у одговарајући улазни податак $x \in X$, тј. важи $x = f^{-1}(y)$, где је алгоритам којим се рачуна инверзна вредност изузетно велике сложености (експоненцијалне реда $O(a^n)$, где је a константа и n битска ширина улазног податка) и припада класи NP проблема (а да истовремено није у оквиру P класе), у којој се налазе проблеми чије решење може да провери детерминистичка Тјуринога машина за полиномијално време, док до самог решења може да дође недетерминистичка Тјурингова машина за полиномијално време.

Код другог наведеног захтева за *one-way* функцију потребно је нагласити да је изузетно тешко испитати да ли је најефикаснији алгоритам којим се рачуна излазна вредност инверзне функције заиста одговарајуће сложености. Такође, традиционални појмови из анализирања комплексности алгоритама – сложеност у просечном и најгорем случају не играју превелику улогу, будући да је у оквиру криптографије врло битно да одговарајућа комплексност важи за рачунање инверзне вредности свих или готово свих улазних параметара. Поред примене у асиметричној криптографији *one-way* функције се користе и код дефинисања и конструкције хеш функција. [6]

One-way trap-door функције представљају категорију *one-way* функција, где се вредност функције $y = f_k(x)$ рачуна алгоритмом који је ефикасан у једном смеру, док се рачунање инверзне функције $x = f_k^{-1}(y)$ ради користећи алгоритам изузетно велике сложености (експоненцијалне), осим у случају кад је позната тајна информација k која представља приватни кључ асиметричног алгоритма, кад је алгоритам за рачунање инверзне вредности полиномијалне сложености. [5]

Уколико су скупови X и Y једнаки онда се *one-way trap-door* функције називају *one-way trap-door* пермутације.

Будући да *one-way trap-door* функције представљају централни део сваког асиметричног алгоритма, уколико се пронађе ефикаснији алгоритам којим се претходно поменут проблем инверзије решава полиномијалном сложеносту (тима додатно класификујући тај проблем у проблеме P класе сложености), цео алгоритам заснован на тој *one-way trap-door* функцији постаје рањив на нападе.

3.2. Дефиниција асиметричног алгоритма

Асиметрични алгоритам $\varepsilon = (G, E, D)$ представља скуп од три функције. Такође дефинишимо и скупове ограничене величине X и Y , који представљају скупове свих могућих порука и шифрованих порука, респективно. Тада за ε важи [5]:

- G представља функцију која генерише повезани пар кључева – јавни (pk – енг. *public key*) и приватни кључ (sk – енг. *secret key*) насумично из скупа свих могућих кључева на ефикасан начин, тј. важи $(pk, sk) \xleftarrow{R} G()$. Иако су јавни и приватни кључ математички повезани, за потенцијалног нападача је изузетно захтевно да преко јавног кључа дође до одговарајућег приватног кључа.
- E представља *one-way trap-door* функцију, која се користи за енкрипцију порука - за задату поруку $x \in X$ и одговарајући јавни кључ pk рачуна шифровану поруку $y \in Y$, тј. важи $y = E(pk, x)$.
- D представља функцију која је инверзна функцији E и користи се за декрипцију порука – за задату шифровану поруку $y \in Y$ и за одговарајући приватни кључ sk рачуна одговарајућу почетну поруку $x \in X$, тј. важи $x = D(sk, y)$. Претходно описана функционалност се може написати и као $x = D(sk, E(pk, x))$.
- додатно да се и приватни кључ може користити за енкрипцију, а јавни за декрипцију, тј. да важи $x = D(sk, E(pk, x)) = D(pk, E(sk, x))$. Ово својство, које обезбеђује аутентификацију (друга страна је сигурна да је поруку послао одговарајући пошиљалац) и интегритет, није обавезно за све асиметричне алгоритме.

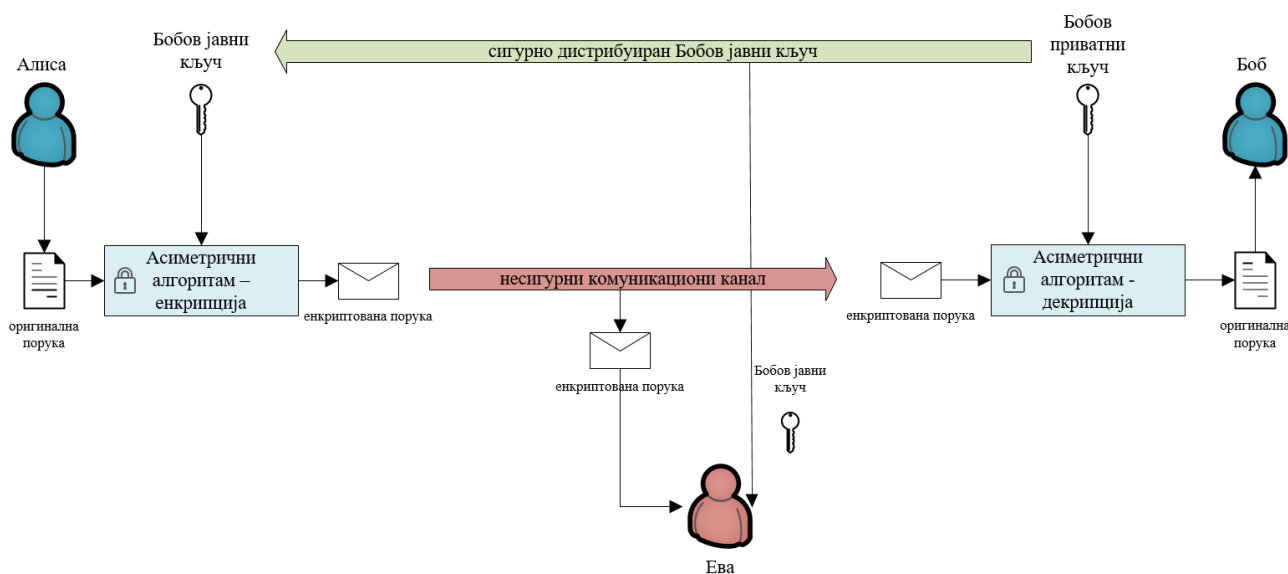
Додатно је потребно напоменути да наведена дефиниција асиметричних алгоритама важи „у ужем смислу“, будући да постоје хибридни асиметрични алгоритми и алгоритми за размену кључева који не подржавају директну енкрипцију јавним кључем, нпр. *Diffie-Hellman* алгоритам, код кога би *one-way* функција представљала процес генерисања јавних и приватних кључева, а дељењем својих јавних кључева обе стране долазе до заједничке тајне користећи своје приватне кључеве, на основу које могу да генеришу тајни кључ који ће се користити у оквиру симетричних алгоритама. *RSA*, који представља један од тренутно најкоришћенијих асиметричних алгоритама у потпуности подржава приказану дефиницију.

Основу развоја већине данашњих асиметричних алгоритама представља развој гране математике која је позната као теорија бројева, која се бави проучавањем особина (целих) бројева и функција. Разлог за ово је једноставна битска представа бројева, као и избегавање прорачунских грешака рачунара које се јављају у оквиру операција са реалним бројевима са покретним зарезом.

Асиметрична енкрипција није настала као замена симетричној, иако подржава исте операције, већ је њена намена решавање претходно поменутих проблема размене кључева и дигиталног потписивања које је било немогуће решити на задовољавајући начин користећи само симетричну криптографију. Додатни разлог за ово представљају разлике у перформансама у поређењу са симетричним алгоритмима (асиметрични алгоритми су обично спорији за неколико редова величине и тиме неефикасни за енкрипцију велике количине података [7]).

Као што је претходно поменуто код симетричних алгоритама, и код асиметричних кључеви морају бити одговарајуће дужине, да би се спречили *brute-force* напади. Међутим за разлику од симетричних алгоритама, овде постоји компромис између бирања сигурне дужине кључева и ефикасности операција генерисања кључева, енкрипције и декрипције, будући да сложеност рачунања ових операција није линеарна у зависности од битске дужине кључева, већ полиномијална. Потребно је пронаћи одговарајућу дужину кључева, којом *brute-force* напади постају непрактични, а операције задржавају одговарајућу ефикасност.

3.3. Примена асиметричних алгоритама

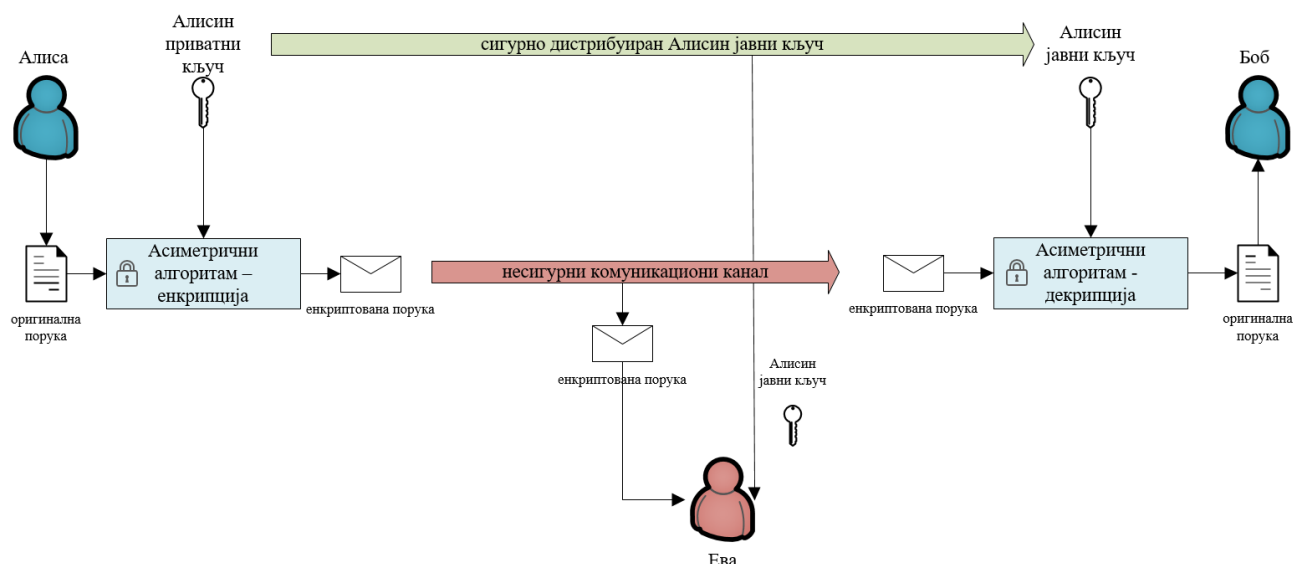


Слика 3.1 - Блок дијаграм комуникације користећи асиметрични алгоритам за енкрипцију порука - тајност

На слици 3.1 приказан је блок дијаграм комуникације између два ентитета – Алисе и Боба, са обезбеђеним својством тајности послатих порука, слично као што смо имали код симетричне енкрипције (слика 2.1) само што се у овом случају користи асиметрични алгоритам за енкрипцију. Пре почетка саме комуникације неопходно је да се Алиса и Боб договоре око асиметричног алгоритма, параметара асиметричног алгоритма, да генеришу парове својих приватних и јавних кључева, као и да размене на сигуран начин своје јавне кључеве (да би спречили *man-in-the-middle* нападе). Алиса и Боб ни у ком случају неће да откривају своје приватне кључеве другим странама.

Алиса ће, користећи Бобов јавни кључ и одговарајући асиметрични енкрипциони алгоритам (са одговарајућим параметрима), генерисати поруку коју ће само Боб моћи да декодира (због *one-way trap-door* карактеристике самог асиметричног алгоритма) користећи свој приватни кључ. Противник Ева ће имати приступ Бобовом јавном кључу и енкриптованој поруци, а сматрамо да поседује и информације о самом алгоритму који се користи. Због својстава асиметричног алгоритма за Еву је врло тешко да дође до Бобовог приватног кључа користећи његов јавни кључ (који би могла да примени у процесу декрипције, долазећи до оригиналне поруке), а истовремено, због особина *one-way trap-door* функције коришћене у алгоритму, Ева не може преко енкриптоване поруке и Бобовог јавног кључа доћи до оригиналне поруке. Дијаграм слања порука у супротном смеру – од Боба ка Алиси био би

еквивалентан, само што би Боб у оквиру енкрипције користио Алисин јавни кључ, док би она користила свој приватни кључ за декрипцију.



Слика 3.2 - Блок дијаграм комуникације користећи асиметрични алгоритам за енкрипцију порука – аутентификација/интегритет

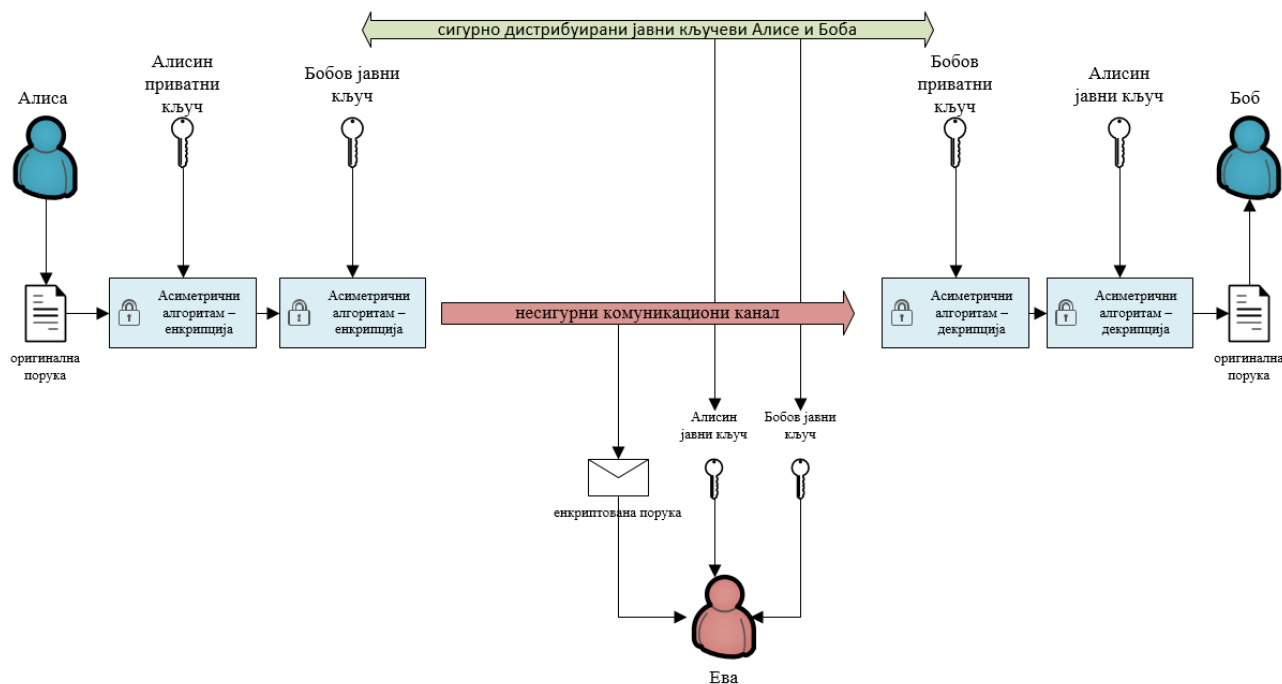
На слици 3.2 приказан је дијаграм комуникације који је врло сличан дијаграму са слике 3.1. Да би шему приказану на дијаграму било могуће реализовати претпоставља се да за коришћење асиметричног алгоритма важи последња особина наведена у оквиру дефиниције асиметричног алгоритма. Такође важе исте претпоставке као и за дијаграм са слике 3.1 – да је договорен асиметрични алгоритам као и његови параметри и да су сигурно размењени јавни кључеви.

У овом случају, захтевана су својства интегритета, аутентификације и непорецивости, а енкриптована порука служи као дигитални сертификат. Алиса користећи свој приватни кључ енкриптује поруку пре слања кроз несигурни канал, а затим ту поруку може да дешифрује свако користећи њен јавни кључ и да на тај начин потврди да је ту поруку заправо послала Алиса и да поруку нико није модификовао у току преноса (будући да само Алиса има приступ свом тајном кључу). Такође Алиса не може да тврди да је неко други послао поруку или да порука уопште није послата. Ева има приступ енкриптованој поруци и Алисином јавном кључу – што значи да ће и она моћи да дешифрује и прочита оригиналну поруку – својство тајности не важи код овакве шеме. Уколико би Ева хтела да модификује полазну поруку, морала би да дође до Алисиног приватног кључа користећи њен јавни кључ или да нађе слабост у оквиру алгоритма енкрипције, тако да порука коју она генерише може да се дешифрује користећи Алисин јавни кључ, што би Еви било изузетно тешко да уради у оба приступа.

На слици 3.3 је приказан дијаграм енкрипционе шеме у којој би била обезбеђена својства тајности, интегритета, аутентификације и непорецивости послатих порука, која представља комбинацију два претходно приказана дијаграма на сликама 3.1 и 3.2. У овом случају Ева неће

моћи да приступи садржају поруке и да верификује дигитални сертификат, будући да је порука енкриптована пре слања Бобовим јавним кључем.

Дијаграми са слика 3.2 и 3.3 такође би били еквивалентни у другом смеру преноса поруке, са заменом места актера у комуникацији и заменом имена приватних/јавних кључева.



Слика 3.3 - Блок дијаграм комуникације користећи асиметрични алгоритам за енкрипцију порука – аутентификација/интегритет и тајност

3.4. Криптоанализа асиметричних алгоритама

Из перспективе криптоанализе, дела криптографије који се бави проучавањем алгоритама и откривањем потенцијалних слабости, постоји више приступа који се могу предузети у процесу анализирања појединог алгорита и његових имплементација, које служе за откривање три основне класе проблема који могу компромитовати сигурност анализираних асиметричних алгоритама [8]:

- проблеми у основама алгорита – одговарајући проблем на ком је заснована *one-way trap-door* функција на основу које је имплементиран алгоритам није проблем одговарајуће сложености (пронађен је ефикасан алгоритам за рачунање инверзне функције без тајне, чиме је проблем на коме је посматрани асиметрични алгоритам заснован прешао из NP у P класу сложености). Наведено може да важи и за функцију генерисања кључа, али и за одређене поставке тј. параметре асиметричног алгорита, за које би проблем био једноставнији за решавање у неким или свим случајевима.
- проблеми у дизајну алгорита – комплексност и тежина извођења напада на асиметрични алгоритам није једнака комплексности и тежини самог проблема на коме је он заснован.

- имплементациони проблеми – нападање алгоритма користећи слабости у оквиру појединих имплементација датог алгоритма или коришћењем *side-channel* напада (напада на несигурно деалоцирану меморију, коришћењем напада базираних на времену извршавања криптографских операција, напада базираних на анализирању електромагнетног зрачења које се емитује из нападнутог уређаја итд.). Једна од најпознатијих сигурносних рањивости која је заснована на овом типу проблема позната је као *Heartbleed Bug*, у оквиру имплементације *heartbeat* функционалности *SSL/TLS*-а (коришћеног за заштиту веб саобраћаја) у широко коришћеној *OpenSSL* криптографској библиотеци. Експлоатисање овакве рањивости не би оставило трагове напада, а с обзиром да је тип напада *buffer-overflow* нападач би потенцијално могао да дође до *SSL/TLS* приватних кључева и да на тај начин стекне способност декрипције саобраћаја од и ка том веб сајту. [10]

У наставку су дефинисане основне сигурносне нотације, које су изузетно битне приликом процеса анализе и описа сигурности самих алгоритама (симетричних и асиметричних), будући да из перспектива више различитих типова нападача (противника у наставку, енг. *adversary*), који имају различите могућности и циљеве, испитују сигурност алгоритама (декларисаних као такмичари, енг. *challenger*) у облику различитих игара [11][9]:

- *OW-CPA* (енг. *One Way - Chosen Plaintext Attack*) представља једну од најосновнијих сигурносних нотација, у чијој игри сваки асиметричан алгоритам треба да победи да би био сматран имало сигурним. Код ове нотације игра почиње операцијом генерисања кључева такмичара, након чега такмичар пошаље свој јавни кључ противнику. Након овога, такмичар генерише насумичну поруку, коју енкриптује користећи свој јавни кључ и пошаље је противнику. Циљ противника у овој игри је да декриптује поруку, познајући алгоритам који се користи и јавни кључ од такмичара, алгоритмом у полиномијалном времену извршавања. У овој игри противник нема приступ операцијама ткзв. пророка.
- *OW-CCA* (енг. *One Way - Chosen Ciphertext Attack*) представља сигурносну нотацију која је слична *OW-CPA*, са тим да противник сад има приступ пророку, преко кога може да изврши произвољан број операција енкрипције користећи исти асиметрични алгоритам као и такмичар и са истим јавним кључем, пре него што добије енкриптовану поруку од такмичара, као и произвољан број декрипција порука које нису једнаке енкриптованој поруци коју је добио од такмичара, користећи приватни кључ од такмичара.
- *IND-CPA* (енг. *INDistinguishability under Chosen Plaintext Attack*) представља сигурносну нотацију где такмичар генерише пар приватних и јавних кључева, шаље свој јавни кључ противнику, а затим противник генерише две произвољне различите поруке једнаке дужине, и шаље их такмичару, који енкриптује насумично изабрану поруку од те две својим јавним кључем и пошаље противнику назад енкриптовану поруку. Након што добије назад енкриптовану поруку, циљ противника је да са што већом вероватноћом препозна од које је оригиналне поруке настала енкриптована порука, а да при томе има на располагању операције које се извршавају у полиномијалном времену пре слања порука/након примања енкриптоване поруке. Противник такође може произвољан број пута да се обрати пророку, који му враћа енкриптовану поруку, користећи исти алгоритам енкрипције и јавни кључ који користи такмичар. Уколико алгоритам не користи насумичне бројеве у процесу енкрипције, противник ће енкриптовањем обе поруке пре слања и једноставним

поређењем са добијеном енкриптованом поруком моћи једноставно да победи у овој игри.

- *IND-CCA1* (енг. *INDistinguishability under Chosen Ciphertext Attack*) представља сигурносну нотацију код које је игра формулисана на исти начин као код *IND-CPA* нотације, са тим да овде противник може да се обрати пророку и за декрипцију произвољних порука пре слања својих изабраних порука такмичару (где пророк користи приватни кључ од такмичара).
- *IND-CCA2* (енг. *INDistinguishability under adaptive Chosen Ciphertext Attack*) представља сигурносну нотацију код које је игра формулисана на исти начин као код *IND-CCA1* нотације, са тим да овде противник може да се обрати пророку за декрипцију произвољних порука и пре слања својих порука такмичару и после добијања енкриптоване поруке, са условом да се не може декриптовати порука која је добијена као одговор од такмичара.
- *NM-CPA* (енг. *Non Malleable - Chosen Plaintext Attack*) представља сигурносну нотацију код које игра започиње процесом генерације пара јавног и приватног кључа код такмичара, након чега се јавни кључ такмичара пошаље противнику. Противник затим може произвољан број пута да се обрати пророку ради енкрипције произвољне поруке користећи исти алгоритам и исти јавни кључ као такмичар и да изврши операције полиномијалне временске сложености. Након овога такмичар генерише насумичну поруку и енкриптује је користећи свој јавни кључ и прослеђује је противнику. Циљ противника у овој игри је да користећи добијену енкриптовану поруку генерише енкриптовану поруку чији је одговарајућа оригинална порука у „смисленој вези“ са поруком генерисаном од стране такмичара (нпр. наредна порука из одговарајућег скупа свих могућих порука), при чему противник и даље има приступ пророку за операције енкрипције.
- *NM-CCA1* (енг. *Non Malleable - Chosen Ciphertext Attack*) представља сигурносну нотацију код које је поставка и циљ исти као и код *NM-CPA*, само што додатно противник има приступ пророку за декрипцију енкриптованих порука користећи тајни кључ од такмичара, пре добијања енкриптоване поруке од такмичара.
- *NM-CCA2* (енг. *Non Malleable - adaptive Chosen Ciphertext Attack*) представља сигурносну нотацију која је дефинисана као и *NM-CCA1*, а противник додатно има приступ пророку за операције декрипције порука и пре и после добијања енкриптоване поруке од такмичара (при чему пророк не може да декриптује добијену поруку од такмичара).

Јачина сигурности криптографског алгоритма се дефинише као број који се повезује са количином рачунарског посла који је неопходан да се одради, са циљем извођења ефективног напада на алгоритам, којим би се поништила сигурност коју он пружа. Најчешће се исказује као број бита, где уколико се каже да је алгоритам сигуран са s бита то означава да је потребно приближно 2^s основних операција да се алгоритам успешно нападне, тј. да је алгоритам напада сложености $O(2^s)$. [12]

Користећи концепт јачине сигурности могуће је логички повезати и поредити сигурност између различитих симетричних и асиметричних енкрипционих алгоритама и алгоритама за размену кључева, алгоритама за дигитално потписивање и хеш функција, поређењем степена комплексности најефикаснијих алгоритама који се користе за напад на било који сигурносни аспект посматраних алгоритама.

Ниво сигурности	Еквивалентна сигурност	Сложеност напада
1	<i>AES128</i> (исцрпна претрага кључа)	$O(2^{64})$
2	<i>SHA256</i> (претрага за хеш колизијама)	$O(2^{85})$
3	<i>AES192</i> (исцрпна претрага кључа)	$O(2^{96})$
4	<i>SHA384</i> (претрага за хеш колизијама)	$O(2^{128})$
5	<i>AES256</i> (исцрпна претрага кључа)	$O(2^{128})$

Табела 3.2 – *NIST* сигурносни нивои за анализу пост-квантне криптографије

У оквиру табеле 3.2 приказани су нивои сигурности који се користе у оквиру *NIST*-овог процеса стандардизације пост-квантних алгоритама, где се приликом оцене сигурности анализираних пост-квантних алгоритама за поређење користе сигурности стандардизованих и широко коришћених алгоритама - *AES* и *SHA*. Процена сложености напада на поменуте алгоритме представља најбољи случај из погледа нападача, на основу квантних алгоритама изложених у [13] и [14], са адекватним квантним ресурсима на располагању. Сложеност датих напада се добија као $O(\sqrt{n})$ за *AES* и $O(\sqrt[3]{n})$ за *SHA*, где n представља величину простора свих могућих кључева (2^s , где је s битска дужина кључа) код *AES* алгоритма, односно величину простора свих могућих излаза хеш функције (2^s , где је s битска дужина излаза хеш функције) код *SHA* алгоритма. Додатно, поред сложености напада, треба узети у обзир и потребне рачунарске ресурсе (у овом случају класичне и квантне) за извођење напада, што прави разлику између 4. и 5. нивоа сигурности. У оквиру документације анализираних пост-квантних алгоритама се из непознатих разлога користе само нивои 1, 3 и 5.

У оквиру [50] дато је поређење јачина сигурности стандардних асиметричних и симетричних алгоритама, при чему дате јачине сигурности искључиво важе за нападе класичних рачунара.

4. Преглед стандардних асиметричних алгоритама

У овом поглављу биће приказани стандардни и тренутно коришћени асиметрични алгоритми – *Diffie-Hellman* и *ElGamal* алгоритам, *RSA*, као и алгоритми засновани на *ECC*.

4.1. *Diffie-Hellman* и *ElGamal* алгоритми

Diffie-Hellman алгоритам за размену кључева представља први јавно објављен асиметрични алгоритам, који су развили Ралф Меркле, Мартин Хелман и Витфилд Дифи 1976. године, који је практично дефинисао почетке развоја асиметричне криптографије и користи се дан-данас у бројним информационим системима где је потребна функционалност размене кључева. У почетку је био патентиран под ознаком *US4200770A*, а након што је патент истекао 1997. године алгоритам се налази у јавном власништву.

Сигурност *Diffie-Hellman* алгоритма почива на сигурности модуларне експоненцијације, тј. наведени проблем служи као *one-way* функција која се користи за генерисање и заштиту парова јавних и приватних кључева.

Уколико је дефинисан прост број p и коначно поље $GF(p)$, дефинише се примитивни корен број g броја p као број за који важи да су бројеви који се добијају као резултати операција $g^1 \bmod p$, $g^2 \bmod p$, ..., $g^{p-1} \bmod p$ сви међусобно различити и представљају наизглед насумичну пермутацију бројева од 1 до $p - 1$. Тада за произвољан број b из $GF(p)$ постоји одговарајући експонент i за који важи: $b = g^i \bmod p$. Дискретни логаритам представља операцију рачунања одговарајућег експонента i , када су дати бројеви b , g и p , тј. може се написати $i = dlog_{g,p}(b)$. Број g се назива и генератор у оквиру датог коначног поља. *Diffie-Hellman* алгоритам управо користи бројеве из поља $GF(p)$ и концепт генератора у $GF(p)$ у оквиру имплементације алгоритма. [5][11]

Основни, анономни *Diffie-Hellman* алгоритам размене кључа (који се добија из заједничке тајне) се може представити преко низа корака [5][11]:

- Генерација и размена параметара алгоритма – генеришу се велики прост број p и његов примитивни корен g или се усвоје њихове вредности на основу одговарајућег стандарда. Учесници у комуникацији Алиса и Боб морају да размене ове вредности да би обављали еквивалентне операције и да би на крају дошли до исте заједничке тајне.
- Генерисање приватних и јавних кључева – Алиса генерише свој приватни кључ као насумичан број $X_A < p$ и рачуна свој јавни кључ $Y_A = g^{X_A} \bmod p$. Еквивалентно, Боб генерише свој приватни кључ као насумичан број $X_B < p$ и рачуна свој јавни кључ $Y_B = g^{X_B} \bmod p$.
- Размена јавних кључева – Алиса и Боб међусобно пошаљу своје јавне кључеве.
- Процес калкулације заједничке тајне – Алиса заједничку тајну рачуна као $K = ((Y_B)^{X_A}) \bmod p$. Еквивалентно, Боб заједничку тајну рачуна као $K = ((Y_A)^{X_B}) \bmod p$. Може се доказати да су срачунате заједничке тајне еквивалентне тј. да важи $((Y_B)^{X_A}) \bmod p = ((Y_A)^{X_B}) \bmod p \Rightarrow ((g^{X_B})^{X_A}) \bmod p = ((g^{X_A})^{X_B}) \bmod p$. Генерисана заједничка тајна представља број који има исти број битова као и број p , тако да је неопходно да се примени хеш или *key derivation* функција одговарајуће

величине излаза да би се сигурно генерисао кључ одговарајуће величине за коришћење у оквиру договореног симетричног алгорита, за даљу комуникацију.

Да би алгоритам био сигуран, прост број p мора бити број са великим бројем цифара. На основу докумената *NIST*-а [15], тренутно се препоручују прости бројеви величине 2048 бита. Слична дужина кључа је препоручена у оквиру [16].

Уколико приватни кључ било које од страна у комуникацији буде компромитован, нападач ће имати приступ генерисаној тајни уколико је меморисао јавни кључ од друге стране, чији кључ није компромитован.

Будући да није могуће проверити порекло јавних кључева у оквиру анонимног *Diffie-Hellman* алгорита, то означава да је поменути алгоритам рањив на *man-in-the-middle* нападе, па коришћење истог није сигурно. Уместо њега најчешће се користе друге две верзије: статички *Diffie-Hellman* и *Ephemeral Diffie-Hellman*.

Статички *Diffie-Hellman* представља модификовани анонимни *Diffie-Hellman*, где је јавни кључ серверске стране у комуникацији, заједно са параметрима алгорита – генератором и простим бројем, садржан у оквиру сертификата који је потписан од стране ауторитета за издавање сертификата, *CA* (енг. *Certificate Authority*), чиме је обезбеђено својство аутентификације сервера. Будући да генерисани сертификати имају одређени период важења у ком су валидни, за време тог периода није могуће променити јавни кључ или параметре без генерисања новог сертификата. Уколико се користи статички *Diffie-Hellman* и уколико дође до компромитације приватног кључа сервера, нападач који има могућност прислушкивања саобраћаја моћи ће да угрози својство тајности и да декриптује поруке (сачуване у прошлости) које су енкриптоване кључем генерисаним помоћу компромитованог пара *Diffie-Hellman* кључева (комуникација нема својство познато као *PFS*, енг. *Perfect Forward Secrecy*). [18]

Ephemeral Diffie-Hellman представља верзију *Diffie-Hellman* алгорита где серверска страна или обе стране у комуникацији користе два пара јавних и приватних кључева. Један пар се назива статички пар који је потписан и налази се у оквиру сертификата и тај пар кључева је сталан тј. не мења се често, док други пар представља *ephemeral* пар кључева који се користи само у оквиру једног процеса размене заједничке тајне. Статички кључеви се користе да се потпишу *ephemeral* јавни кључеви. У овом случају, уколико дође до компромитације приватног статичког кључа нападач ће моћи да компромитује само будућу комуникацију, где ће бити у прилици да генерише привремене парове кључева (важи својство *PFS*). [18]

У оквиру овог потпоглавља је приказан и *ElGamal*, асиметрични алгоритам за енкрипцију јавним кључем, који је заснован на *Diffie-Hellman* алгоритму, приказом низа његових функционалности [5]:

- Генерисање кључева – бирају се параметри p и g , који као и код *Diffie-Hellman* алгорита представљају велики прост број и његов генератор у пољу $GF(p)$. Боб генерише свој приватни кључ као насумични број X_B из поља $GF(p)$, тј. важи $1 < X_B < p - 1$. Примарни део јавног кључа Y_B се рачуна као и код *Diffie-Hellman* алгорита $Y_B = g^{X_B} \bmod p$, док цео јавни кључ поред Y_B садржи и параметре p и g те се приказује као (p, g, Y_B) . Након генерације кључева Боб ће свој јавни кључ на сигуран начин послати до Алисе.
- Енкрипција јавним кључем – ова функционалност се може представити дијаграмом са слике 3.1, где Алиса шаље поруку Бобу са претходно сигурно добијеним Бобовим јавним кључем (p, g, Y_B) , при чему порука M представља низ бита који је кад се представи као број припада у опсегу $0 \leq M \leq p - 1$. Веће поруке би морале да се

раставе на делове одговарајуће величине и да се енкриптују засебно, а након декрипције да се споје да би се добила оригинална порука. Алиса затим бира насумични број k , за који важи $1 \leq k \leq p-1$ и на основу њега конструише привремени тајни кључ који се користи за енкрипцију поруке $K = (Y_B)^k \bmod p$. Енкриптовани облик поруке се дефинише као пар целих бројева (C_1, C_2) , при чему C_1 представља број на основу ког Боб долази до тајног кључа који је коришћен у енкрипцији поруке M и рачуна се као $C_1 = g^k \bmod p$, док C_2 представља саму енкриптовану поруку која се рачуна као $C_2 = K \cdot M \bmod p$. Након процеса енкрипције Алиса шаље Бобу енкриптовану поруку (C_1, C_2) .

- Декрипција приватним кључем – ова функционалност је такође приказана на слици 3.1, где Боб прима поруку од Алисе енкриптовану својим јавним кључем. Боб ће применити свој приватни кључ X_B да израчуна тајни кључ K који је коришћен у енкрипцији поруке, преко формуле $K = (C_1)^{X_B} \bmod p$. Након овога, Боб ће доћи до поруке $M = (C_2 \cdot K^{-1}) \bmod p$, при чему K^{-1} представља мултипликативно инверзан број тајном кључу K .

Доказ за *ElGamal* потиче од доказа за *Diffie-Hellman* алгоритам, будући да се иста *one-way* функција користи за генерацију парова приватних и јавних кључева, док процес генерације и размене тајног кључа може упоредити са процесом размене и генерисања дељене тајне у оквиру *Diffie-Hellman* алгоритма, при чему *one-way trap-door* функција која се користи за енкрипцију поруке представља мултипликацију са тајним бројем K у оквиру $GF(p)$.

4.2. RSA алгоритам

RSA (Rivest Shamir Adleman) је асиметрични алгоритам за енкрипцију јавним кључем и за генерисање дигиталних потписа, који су развили Роналд Ривест, Ади Шамир и Леонард Ејдлман и који је јавно описан августа 1977. године у часопису *Scientific American*, а затим објављен фебруара 1978. године у оквиру рада [17]. Данас је један од најкоришћенијих асиметричних алгоритама. Патентиран је септембра 1983. године под ознаком *US4405829A*, а након шестог септембра 2000. године алгоритам се налази у јавном власништву.

Сигурност *RSA* алгоритма почива на сигурности модуларне експоненцијације, која се користи као *one-way trap-door* функција у процесу енкрипције/декрипције (чију инверзну функцију представљају дискретни логаритми, слично као код *Diffie-Hellman* алгоритма) и на сигурности факторизације целих бројева, *one-way* функције која се користи у оквиру процеса генерације парова приватних и јавних кључева.

RSA, као што је већ наведено, у потпуности испуњава дефиницију асиметричног алгоритма, при чему сам алгоритам користи бројеве из $\mathbb{Z}/n\mathbb{Z}$, мултипликативне групе бројева који су мањи и међусобно прости са бројем n , који је у наставку дефинисан као јавни модул и представља део јавног и приватног кључа. Укупан број чланова ове групе се рачуна помоћу Ојлерове фи функције као $|\mathbb{Z}/n\mathbb{Z}| = \varphi(n)$. Алгоритам се може дефинисати приказом низа његових функционалности [5][11]:

- Генерисање кључева – генеришу се два насумична различита велика проста броја p и q , који остају тајни. Након овога се рачуна вредност јавног модула $n = p \cdot q$ и тајна вредност Ојлерове фи функције од n , $\varphi(n) = (p-1) \cdot (q-1)$, која представља број позитивних целих бројева, мањих од n (без нуле), који су узајамно прости са n . Након

овога се бира вредност e , која представља јавни експонент, тако да важи $1 < e < \varphi(n)$ и $\gcd(e, \varphi(n)) = 1$, где \gcd представља ознаку за највећи заједнички делилац (обично се узимају вредности 3, 17 или 65 537). На крају је потребно да се израчуна приватни кључ d , као мултипликативно инверзан број у односу на јавни кључ e под модулом $\varphi(n)$, тј. важи $e \cdot d \equiv 1 \bmod \varphi(n)$. Приватни кључ представља пар (d, n) , док јавни кључ представља пар (e, n) .

- Енкрипција јавним кључем – ова функционалност се може представити дијаграмом са слике 3.1, где Алиса шаље поруку Бобу са претходно сигурно добијеним Бобовим јавним кључем (e, n) . Алиса ће у почетку поруку издвојити на блокове, где сваки за блок поруке M важи $M < n$. Алиса ће затим да енкриптује блок поруке M и да израчуна одговарајући енкриптовани облик од блока $C = M^e \bmod n$. Ове енкриптоване блокове ће Алиса конкатенирати и послати Бобу.
- Декрипција приватним кључем – ова функционалност је такође приказана на слици 3.1, где Боб прима поруку од Алисе енкриптовану својим јавним кључем, у облику више енкриптованих блокова. Боб ће да примени свој приватни кључ (d, n) да декриптује сваки блок поруке као $M = C^d \bmod n$ и да након тога обави операцију конкатенације декриптованих блокова, чиме добија целу почетну поруку. При томе, нико без мултипликативно инверзне вредности од вредности јавног експонента не може да декриптује поруку.
- Енкрипција приватним кључем – ова функционалност је приказана на слици 3.2, где Алиса, слично као код процеса енкрипције јавним кључем, користи свој приватни кључ (d, n) у циљу енкрипције блокова поруке означених са M . Шифровани облик блока се рачуна као $C = M^d \bmod n$. После енкрипције, блокови се конкатенирају и шаљу Бобу.
- Декрипција јавним кључем – ова функционалност је такође приказана на слици 3.2, у делу где Боб добија поруку енкриптовану Алисиним приватним кључем. У овом случају Боб ће да декриптује блокове поруке користећи Алисин јавни кључ (e, n) као $M = C^e \bmod n$. Након конкатенације свих блокова добија се оригинална порука. Боб је сигуран да је енкриптовану поруку могла да генерише само Алиса, будући да само она поседује одговарајући приватни експонент.

Доказ да је процес енкрипције и декрипције валидан може се показати доказом да је $D((d, n), E((e, n), M)) = D((e, n), E((d, n), M)) = M$, тј. да важи $M^{e \cdot d} \bmod n = M^{d \cdot e} \bmod n = M$. Валидност првог дела једнакости $M^{e \cdot d} \bmod n = M^{d \cdot e} \bmod n$ потиче из особине комутативности операције множења у оквиру групе $\mathbb{Z}/n\mathbb{Z}$. Такође будући да важи $e \cdot d \equiv 1 \bmod \varphi(n)$, може се написати $M^{e \cdot d} \bmod n = M^{k \cdot \varphi(n) + 1} \bmod n$, где k означава произвољни цели број. На основу Ојлерове теореме знамо да важи следећа једнакост $M^{\varphi(n)} \equiv 1 \bmod n$ за произвољан број M из $\mathbb{Z}/n\mathbb{Z}$, из чега следи једнакост $M^{k \cdot \varphi(n) + 1} \bmod n = (M^{\varphi(n)})^k \cdot M \bmod n = 1^k \cdot M \bmod n = M$, чиме је доказано својство енкрипције и декрипције користећи јавне и приватне кључеве. [17]

Да би алгоритам био сигуран поред напада на експоненцијалну модулацију, потребно је осигурати алгоритам од напада чији је циљ да дођу до приватног експонента d користећи јавни кључ (e, n) . Да би се ово обезбедило потребно је изабрати довољно велико n и одговарајуће просте бројеве p и q који су довољно удаљени како би процес факторизације n био готово немогућ за нападача. Препоручене величине кључева (јавног модула n) су 2048 битоа за

размену симетричних кључева, енкрипцију порука, као и за дигиталне сертификате, док се вредност од 3072 препоручује уколико се кључ користи за потписивање других јавних кључева (уколико га користи *CA*). [15] [16]

Уколико би размотрили основни сценарио напада *OW-CPA RSA* алгоритам би био сигуран против нападача, међутим приказани алгоритам, који је познат и као *textbook RSA*, није сасвим отпоран на нападе уколико би нападач имао додатне способности поред поседовања одговарајућег јавног кључа и знања о алгоритму. Уколико би нападач имао приступ пророку за операције декрипције и енкрипције, тј. уколико би нападач био *CCA* категорије, могао би чак и да декриптује поруку уколико нема одговарајућег *padding*-а.

Ово се може показати, уколико важи да множењем две енкриптоване поруке добијамо енкриптовану поруку која кад се декриптује даје производ две почетне поруке (ради једноставности усваја се да порука има само један блок M), тј. уколико важи: $E((e, n), M_1) \cdot E((e, n), M_2) = E((e, n), (M_1 \cdot M_2))$. Након што нападач добије енкриптовану поруку C чији садржај хоће да декриптује, он ће израчунати енкриптовану поруку $X = (C \bmod n) \cdot (2^e \bmod n) = (M^e \bmod n) \cdot (2^e \bmod n) = (2M)^e \bmod n$. Ово се ради да би се превазишла препрека пророка који не може директно да декриптује добијену поруку C . Након што пророк декриптује срачунату поруку X користећи одговарајући приватни кључ, нападач ће поседовати одговарајућу декриптовану поруку $Y = (2M) \bmod n$, преко које ће једноставно срачунати вредност M , чиме ефективно побеђује у *OW-CCA* игри (а самим тим и у *IND-CCA2* и *NM-CCA2* играма). Такође, важи да овакав алгоритам није *IND-CPA* безбедан, будући да се у оквиру процеса енкрипције не користе насумични бројеви – једна порука ће увек имати исти енкриптовани облик, па на основу тога нападач ће увек моћи да одреди која је послата порука енкриптована. [5] [11]

Да би се онемогућили напади ове врсте, потребно је да се користи *padding*, који би укључивао између осталог и коришћење насумичних бројева и хеш функција, са последицом да би се онда у том случају максимална ефективна величина блока за енкрипцију смањила. Постојало је више верзија *padding*-а за *RSA* алгоритам, а тренутно се препоручује коришћење *OAEP* (енг. *Optimal Asymmetric Encryption Padding*). [5]

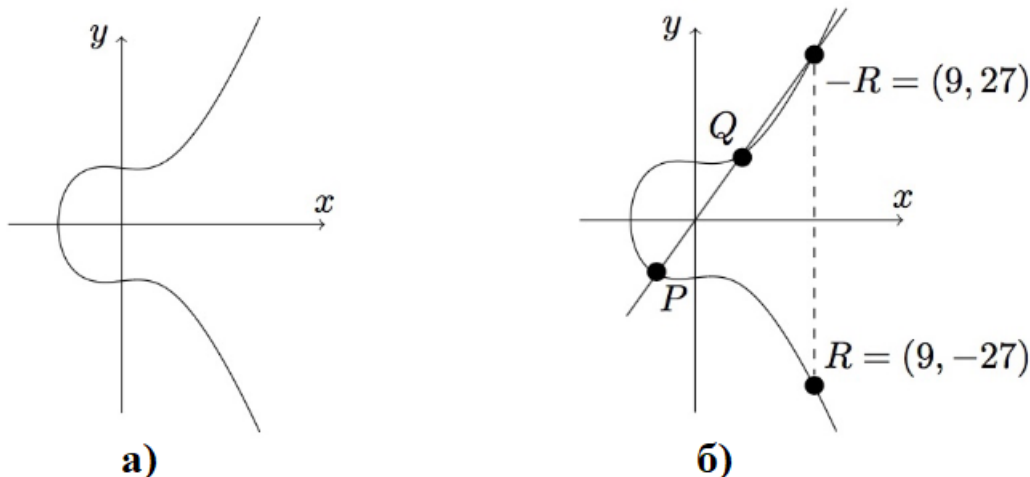
4.3. Алгоритми засновани на *ECC*

ECC (енг. *Elliptic Curve Cryptography*) представља део криптографије који се бави проучавањем примена елиптичних крива и операција над њима, што представља контраст од претходно изложених математичких концепта који су коришћени у оквиру *Diffie-Hellman* и *RSA* алгоритма, где су вршене операције модуларне експоненцијације у оквиру коначних поља или мултипликативних група. Разлог коришћења *ECC* криптографије представља последицу постојања алгоритма за рачунање дискретних логаритама на класичним рачунарима који је познат као генерално сито поља бројева (енг. *General Number Field Sieve, GNFS*), чије је време рада приближне експоненцијалне сложености $O(e^{(\log p)^{\frac{1}{3}} \cdot (\log \log p)^{\frac{2}{3}}})$, где p коришћени прост број величине $\lceil \log_2 p \rceil$ битова, којим је 2019. године решен проблем дискретног логаритма који је као модуо користио прост број p величине 795 битова. Као последица овога морају да се користе прости бројеви велике битске величине, углавном веће од 2048 битова, а будући да операције енкрипције, декрипције и генерације кључева нису линеарне сложености већ

полиномијалне, на уређајима мањих могућности поменуте операције могу бити изузетно временски захтевне. [1]

За разлику од претходно поменутог алгоритма за решавање дискретних логаритама у пољу целих бројева, најбољи алгоритам за решавање проблема дискретног логарита у оквиру групе тачака (сигурне) елиптичне криве величине q (прост број који дефинише поље елиптичне криве) има време извршавања $O(\sqrt{q})$. Ово значи да елиптична крива може да ради операције користећи просте бројеве од 256 битова, што је значајно брже у поређењу са операцијама које користе просте бројеве од 2048 битова, а да при томе оба алгоритма имају исти сигурносни ниво. [1]

Елиптичне криве се најчешће представљају у облику Вајерштрасове једначине $y^2 = x^3 + ax + b$, где су a и b реални бројеви. Поред ове једначине постоји још форми које се користе за приказ једначина крива – Монтгомери и Едвардсове криве, од којих су неке једноставније за рачунарске операције. На слици 4.1 а) приказан је пример елиптичне криве са Вајерштрасовом једначином $y^2 = x^3 - x + 9$.



Слика 4.1 – Пример елиптичне криве [1]

Елиптичне криве се јављају у више математичких области и као један од основних проблема за који је било потребно решење је налажење додатних решења једначине криве (тачка које имају x и y координате и налазе се на криви).

Једно од решења поменутог проблема је коришћење методе дужи, користећи две познате тачке које припадају криви. На слици 4.1 б) приказане су две тачке на криви $P = (-1, -3)$ и $Q = (1, 3)$, као и права $y = 3 \cdot x$ која је конструисана кроз њих. На слици се види да дата права сече елиптичну криву у новој, трећој тачки која је на слици означена са $-R$, чије координате се могу добити као $(9, 27)$ уколико решимо једначину која се добије кад убацимо смену $y = 3 \cdot x$ у оквиру једначине криве, чиме добијамо одговарајућу вредност x координате тачке. Приликом решавања једначине, након што се убаци одговарајућа вредност x , добијамо две могуће вредности за y координату, што означава да смо решавањем једначине добили две тачке, $-R$, која је означена као пресек праве и криве и тачку R , која има једнаке координате са $-R$, са разликом да је y координата другог знака.

Друго решење посматраног проблема подразумева коришћење методе тангенте, где се преко само једне тачке долази до нових решења једначине елиптичне криве. У овом случају би се у познатој тачки уцртала тангента на криву, која има једну тачку пресека са елиптичном кривом, а преко те тачке се може на исти начин (променом знака у координате) доћи до додатне тачке.

Две приказане методе преко којих долазимо до додатних тачака су дефинисане преко оператора \boxplus , који је терминолошки еквивалентан оператору експоненцијације у оквиру модуларних мултипликативних група и коначних поља (код *RSA* и *Diffie-Hellman* алгоритама). Уколико се оператор користи у облику $P \boxplus Q$, где су P и Q различите тачке, то означава да се користи метода дужи кроз две тачке, док уколико се оператор користи са истом тачком $P \boxplus P$ то означава коришћење методе тангенте. Такође вишеструко коришћење овог оператора на истој тачки P може се приказати као мултипликација $n \times P = (n - 1) \times P \boxplus P$. Додатно код овог оператора важе својства асоцијативности и комутативности. [5]

У оквиру криптографије, од значаја је углавном коришћење елиптичних крива над ограниченим пољима. Дефинишимо елиптичну криву E над пољем $GF(p)$ дефинисаним помоћу простог броја p , у нотацији $E/GF(p)$, као једначину $y^2 = x^3 + ax + b$, при чему $a, b \in GF(p)$, док решења једначине криве се представљају као скуп тачака чије координате такође припадају скупу $GF(p)$. У овај скуп такође додајемо тачку O која представља неутрални елемент за оператор \boxplus . Додатан услов који се усваја је да важи $4a^3 + 27b^2 \neq 0$. [1]

Основу сигурности у оквиру криптографије која користи елиптичне криве представља проблем дискретног логаритма. У овом случају проблем дискретног логаритма је дефинисан за операцију рачунања укупног броја коришћења оператора \boxplus , чиме се долази до тачке на криви $Q = n \times P$, при чему су нападачу познате почетна тачка P и крајња тачка Q , док број $n \in GF(p)$ остаје тајан. Овај проблем је изузетно компликован за решавање за адекватне криве, док рачунање крајње тачке једноставно уколико је познат број n .

Наравно, да би претходно наведени проблем био довољно тежак за решавање потребно је пронаћи сигурну криву са одговарајућим параметрима, што није лако за урадити, будући да постоји више различитих категорија напада. Зато се у оквиру стандарда везаних за *ECC*, поред алгоритама и шема дефинишу и параметри крива које су оцењене као довољно сигурне за коришћење у криптографији. Две најпознатије стандардизоване криве су *Curve25519* и крива *P256*.

Elliptic Curve Diffie-Hellman алгоритам је настао као проширење оригиналног *Diffie-Hellman* алгоритма, при чему се овде операције врше над тачкама одговарајуће елиптичне криве, на основу чега се добијају много боље перформансе за исти сигурносни ниво у односу на оригинални *Diffie-Hellman* алгоритам. У наставку је дефинисан алгоритам преко низа његових функционалности [5]:

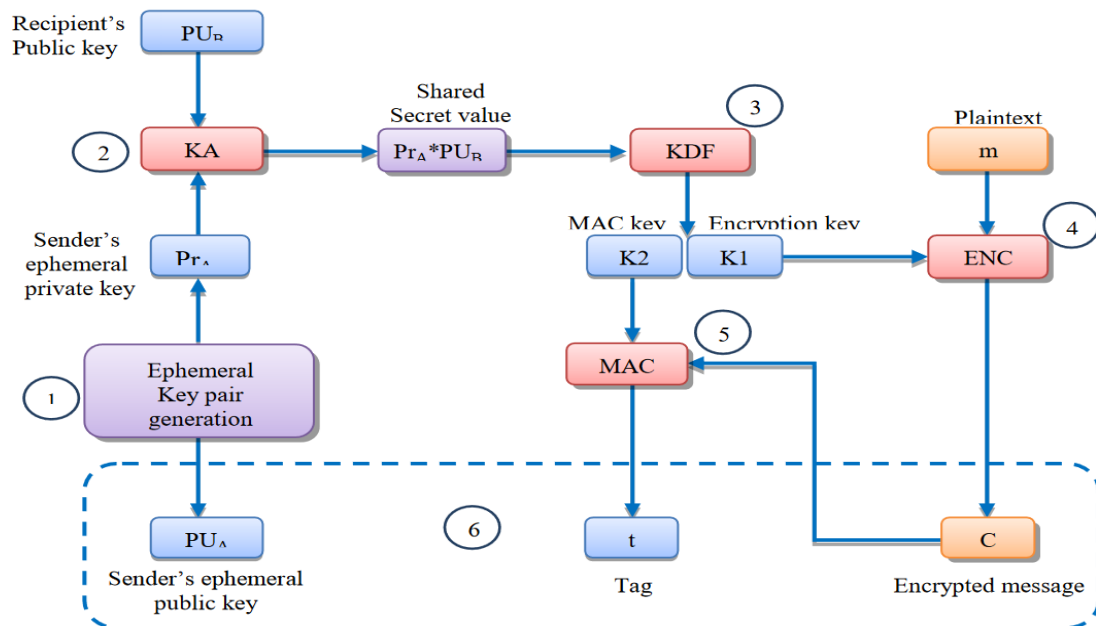
- Размена параметара алгоритма – бирају се параметри алгоритма – одговарајућа елиптична крива са параметрима a и b , прост број p и сигурну почетну тачку G и размене се са другом страном тако да обе стране користе једнаке параметре.
- Генерисање приватних и јавних кључева – Алиса генерише свој приватни кључ као насумичан број $n_A < p$ и рачуна свој јавни кључ $P_A = n_A \times G$. Еквивалентно, Боб генерише свој приватни кључ као насумичан број $n_B < p$ и рачуна свој јавни кључ $P_B = n_B \times G$.
- Размена јавних кључева – Алиса и Боб међусобно пошаљу своје јавне кључеве.

- Процес калкулације заједничке тајне – Алиса заједничку тајну рачуна као $K = n_A \times P_B$. Еквивалентно, Боб заједничку тајну рачуна као $K = n_B \times P_A$.

Дељена тајна у овом случају представља тачку K , при чему се најчешће узима њена x координата за дељену тајну, која се након тога користи у оквиру хеш функције или KDF за извођење тајног кључа симетричног алгорита.

Доказ исправности операција енкрипције и декрипције лежи у особини комутативности оператора \boxplus . Такође код овог алгорита важе исте напомене као и за *Diffie-Hellman* над коначним пољима.

ECIES (енг. *Elliptic Curve Integrated Encryption Scheme*) представља хибридну енкрипциону шему која користи елиптичне криве и врло подсећа на *ElGamal* алгорита, будући да користи *Elliptic Curve Diffie-Hellman* алгорита за размену тајног кључа, на исти начин као што то ради *ElGamal* са *Diffie-Hellman* алгоритмом. *ECIES* користи више типова криптографских функција – симетрични алгорита за енкрипцију података, *MAC* (енг. *Message Authentication Code*) и *key derivation* функцију (којом се из дељене јавне тајне генеришу кључеви за симетрични алгорита и *MAC*).

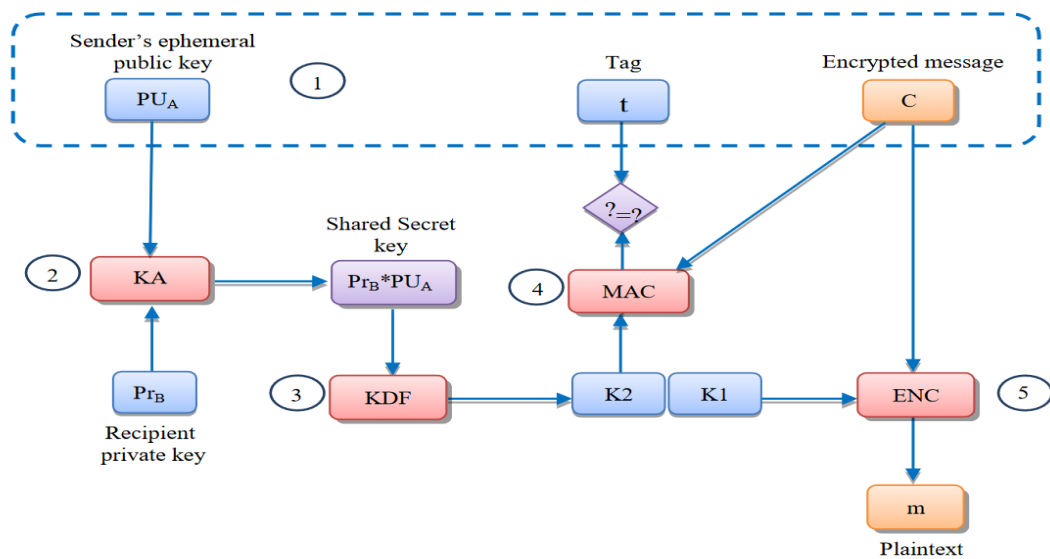


Слика 4.2 – *ECIES* шема за енкрипцију [19]

На слици 4.2 приказан је процес енкрипције поруке коју Алиса шаље Бобу користећи *ECIES*. У оквиру ове шеме црвеном бојом су означене криптографске функције, плавом бојом кључеви, а наранџастом бојом оригинална порука и енкриптована порука. У круговима су приказани кораци у оквиру алгорита енкрипције (претпоставља се да су Алиса и Боб разменили статичке јавне кључеве и параметре криве и почетну тачку, као и параметре *ECIES* алгорита – изабрани симетрични алгорита, *KDF* и *MAC* алгоритме) [19]:

1. Алиса генерише свој *ephemeral* пар приватног и јавног кључа, који су на слици 4.2 означени као PU_A и Pr_A , преко формуле $PU_A = Pr_A \times P$, где PU_A представља Алисину јавну тачку на криви, Pr_A Алисин тајни број, а P почетну тачку на криви. Посматрани пар приватног и јавног кључа користи се само једном.

- Алиса затим генерише дељену тајну (тачку на криви) $K = Pr_A \times PU_B$, користећи Бобову јавну тачку и исти *key agreement* као код *Elliptic Curve Diffie-Hellman* алгоритма.
- Дељена тајна K се затим користи у оквиру *KDF* алгоритма за генерисање кључева $K1$ и $K2$.
- Кључ $K1$ се користи у оквиру изабраног симетричног алгоритма за енкрипцију поруке m , чиме се обезбеђује својство тајности.
- Енкриптована порука се затим користи у оквиру *MAC* алгоритма, заједно са кључем $K2$, који генерише *tag* као излаз и обезбеђује својства аутентификације и интегритета.
- Формира се коначна порука која се шаље Бобу, која садржи енкриптовану поруку, *tag* и Алисин јавни кључ за дату поруку.



Слика 4.3 - ECIES шема за декрипцију [19]

На слици 4.3 приказан је процес декрипције [19]:

- Након што Боб прими енкриптовану *ECIES* поруку од Алисе, он је дели на њен јавни кључ PU_A , *tag* и на енкриптовани садржај поруке.
- Користећи Алисин јавни кључ, Боб ће да дође до заједничке тајне тачке на елиптичној криви $K = Pr_B \times PU_A$.
- Генерисана заједничка тачка K се затим користи у оквиру *KDF* за генерацију кључева $K1$ и $K2$, који су једнаки као кључеви које је генерисала Алиса и користе се у оквиру *MAC* функције и симетричног алгоритма.
- Пореди се *tag* из поруке са новим *tag*-ом који се рачуна преко кључа $K2$, енкриптоване поруке и алгоритма за *MAC*. Уколико су две вредности једнаке енкриптована порука је валидна, а у супротном се одбацује.
- На крају се декриптује енкриптована порука користећи одговарајући симетрични алгоритам и кључ $K1$.

5. Концепт квантних рачунара и Шоров алгоритам

Квантни рачунари концептуално представљају машине чији су прорачуни и рад засновани на принципима квантне физике и које могу да извршавају специјалне врсте алгоритама који решавају одређене категорије компликованих проблема много ефикасније од класичних рачунара.

Историја квантних рачунара почиње 1981. године, кад је на потребу за конструкцијом квантних рачунара указао амерички теоријски физичар Ричард Фајнман, будући да тадашњи класични рачунари нису могли на ефикасан начин да се искористе у експериментима и симулацијама различитих процеса и феномена који се јављају у природи а који су квантне природе. Интересовање за квантне рачунаре је знатно порасло 1994. године, кад је Петер Шор представио квантни алгоритам за ефикасно решавање проблема факторизације великих бројева и дискретних логаритама у оквиру рада [20] и 1996. године кад је представљен Гроверов алгоритам за ефикасно претраживање великог несортираног скупа података, који може да се искористи за убрзавање *brute-force* напада на симетричне алгоритме и за колизионе нападе на хеш функције [13]. Ова два алгоритма су указала на потенцијал примене квантних рачунара у криптографске сврхе. [21] [23]

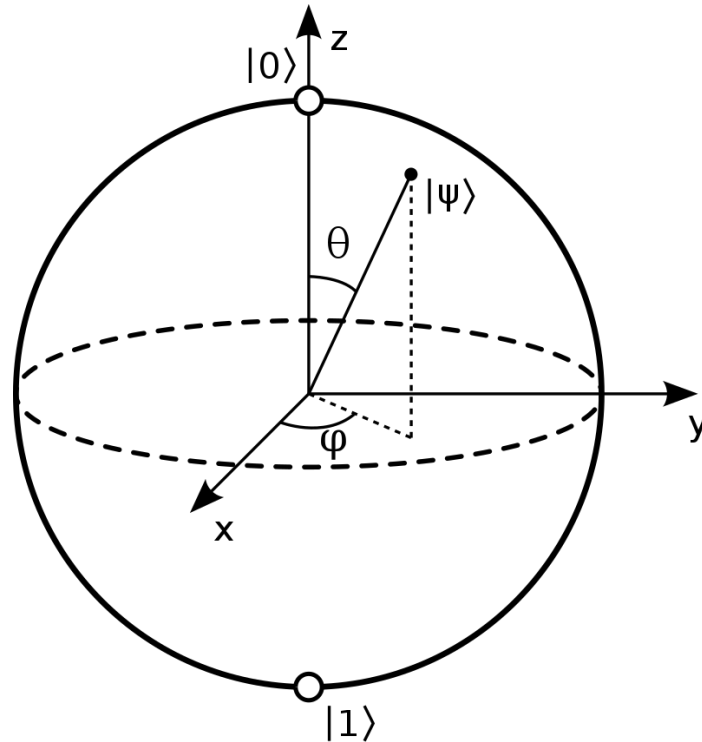
Први квантни рачунар, са 2 кубита, је експериментално приказан 1998. године. Након тога су се у процес развијања и истраживања квантних рачунара укључиле многе велике технолошке компаније као што су *IBM*, *Microsoft*, *Google*, *Intel* и *Xanadu*, а као резултат тога настао је велики број различитих типова квантних процесора [22], заснованих на различитим архитектурама. Данас многе од поменутих компанија пружају услугу приступа њиховим квантним рачунарима који се налазе *cloud*-у, где је могуће извршавање програма за квантне рачунаре користећи одговарајући *API*. [21]

Основна разлика између квантних и класичних рачунара се може видети у карактеристикама најмање количине информација у оквиру рачунарског система. Код класичних рачунара најмања количина информација је представљена битом, бројем бинарног система који може да има вредности 0 или 1, при чему у оквиру његове дефиниције не постоје вероватноће, тј. вредност уписана у дати бит остаће ту док се поново не промени уписом и уколико се у међувремену прочита вредност бита прочитаће се увек вредност која је последња уписана. Преко битова се енкодује свака информација на рачунару и чува у оквиру дискова, *RAM*-а или процесорских регистара, док се вредност битова мења у оквиру интеракције са логичким колима (која имплементирају битске операторе *AND*, *OR*, *XOR*, *NOT*, битско померање итд.). [23]

Кубит представља основну количину информација код квантних рачунара, чија физичка репрезентација представља квантни систем (најчешће се имплементирају користећи електроне, фотоне, јоне или суперпроводнике). Кубит се дефинише као вектор стања у Хилбертовом простору помоћу ткзв. бра-кет нотације $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$, при чему са $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ и $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ означавамо стања која су еквивалентна 0 и 1 у оквиру битова, док α и β представљају комплексне бројеве (који се не могу измерити) за које важи $|\alpha|^2 + |\beta|^2 = \alpha\bar{\alpha} + \beta\bar{\beta} = 1$. Једначина којом је описан кубит представља прво својство кубита – суперпозицију, која представља кубит као суму свих могућих стања у коме он може да буде. Бројеви $|\alpha|^2$ и $|\beta|^2$ представљају вероватноће да ће приликом мерења кубита измерена вредност бити 0 и 1 респективно, што илуструје прву разлику између бита и кубита. Такође кубит након што се

измери (долажењем у контакт са окружењем) губи своја квантна својства и понаша се као класични бит, а његова вредност постаје једнака једном од могућих стања – 0 или 1. [25]













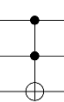
Кубит се такође може приказати као тачка на Блоховој сфери (чији је пречник једнак јединици), као што је приказано на слици 5.1, при чему је кубит одређен искључиво угловима θ и φ , преко формуле $|\psi\rangle = \cos \theta |0\rangle + e^{j\varphi} \sin \theta |1\rangle$. Уколико је тачка ближа неком од полова на сфери, већа је вероватноћа да ће кубит имати вредност тог пола приликом мерења. [23]



Слика 5.1 – Приказ кубита на Блоховој сфери [48]

Као и код класичних рачунара, где битове групишемо да би формирали већу логичку целину – меморијски регистар, исто се може чинити и са кубитима. Међутим овде се јавља битна разлика која илуструје значај квантних рачунара – уколико имамо N груписаних битова у оквиру регистра, код обичних рачунара они могу да чувају једну вредност од N битова, док би код квантних рачунара меморијски регистар од N кубита имао истовремено у оквиру себе 2^N могућих вредности, због својства суперпозиције, све док не извршимо мерење, када добијемо један од могућих резултата.

Наравно, да би кубити били корисни, неопходно је да постоје начини да се мењају, слично као што су код обичних рачунара коришћена логичка кола која извршавају операције Булове алгебре. За извршавање операција над кубитима користе се квантна логичка кола, која се математички представљају у облику унитарних матрица (матрице чија је конјуговано-транспонована матрица инверзна почетној матрици). Услов везан за матрице које репрезентују квантна логичка кола потиче од закона квантне физике да се информације не губе приликом транзиције из прошлости у будућност. Додатан детаљ који проистиче из овог правила је да матрице морају да одрже норму улазног вектора који представља сва могућа стања групе улазних кубита, тј. укупна вероватноћа свих могућих стања кубита мора увек бити једнака јединици. На слици 5.2 дат је преглед најпознатијих квантних логичких кола, њихових шема и матрица које описују њихову функцију. [23] [25]

Operator	Gate(s)	Matrix
Pauli-X (X)	 	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli-Y (Y)		$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Pauli-Z (Z)		$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Hadamard (H)		$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
Phase (S, P)		$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$
$\pi/8$ (T)		$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$
Controlled Not (CNOT, CX)		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
Controlled Z (CZ)	 	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$
SWAP	 	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Toffoli (CCNOT, CCX, TOFF)		$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$

Слика 5.2 – Најпознатија квантна логичка кола [49]

Шоров алгоритам је заснован на особини периодичности функције експоненцијалне модуларације $f(a) = x^a \bmod N$, при чему N представља једину улазну вредност алгоритма - број који је потребно факторисати, док $x < N$ је број који је међусобно прост са бројем N . Као један од главних циљева који се јављају у оквиру алгоритма је проналазак периода поменуте функције, који се може дефинисати као најмања ненулта вредност (будући да већ важи $x^0 \bmod N = 1$) броја r , за коју важи $x^r \equiv 1 \bmod N$. На обичним рачунарима процес рачунања ове вредности, уколико су познати x и N , представља проблем експоненцијалне сложености у зависности од броја бита броја N , док Шоров алгоритам за решавање овог проблема користи квантну Фуријеову трансформацију заједно са доступним хардвером квантног рачунара ради решавања овог проблема алгоритмом полиномијалне временске сложености.

Редукцију проблема факторизације броја N на проблем проналаска периода функције експоненцијалне модуларације која користи исти број N може се показати уколико нам је позната вредност периода - број r . Будући да важи $\left(x^{\frac{r}{2}}\right)^2 = x^r \equiv 1 \bmod N$, уколико је r паран број, може се написати да важи $\left(x^{\frac{r}{2}}\right)^2 - 1 \equiv 0 \bmod N$, након чега следи да важи $\left(x^{\frac{r}{2}} - 1\right)\left(x^{\frac{r}{2}} + 1\right) \equiv 0 \bmod N$, што по дефиницији означава да један од бројева $\left(x^{\frac{r}{2}} - 1\right)$ или $\left(x^{\frac{r}{2}} + 1\right)$ има нетривијални фактор који је заједнички са бројем N . [24]

Шоров алгоритам за факторизацију великих бројева се може приказати у низу корака [24] [26]:

1. Утврђује се да ли је број N прост број, парни број или број који је настао степеновањем простог броја цео број пута. Уколико нешто од наведеног важи алгоритам се овде зауставља. За утврђивање ових услова користе се постојећи алгоритми који се ефикасно извршавају на обичном рачунару.
2. Бира се насумични број q који представља степен двојке тако да припада опсегу $N^2 \leq q \leq 2N^2$. Овај део алгоритма може да се изврши на обичном рачунару.
3. Бира се насумични број x , који је међусобно прост са бројем N , тј. важи $\gcd(x, N) = 1$. Овај део алгоритма такође може ефикасно да се одради на обичном рачунару, коришћењем Еуклидовог алгоритма.
4. Креирамо квантни регистар одговарајуће дужине и поделимо га на два дела, при чему први део регистра мора да има довољно кубита да се представе бројеви до $q - 1$ (ширине $\lceil \log_2 q \rceil$ кубита), док други део регистра мора да има довољно кубита да се представе бројеви до $N - 1$ (ширине $\lceil \log_2 N \rceil$ кубита).
5. У оквиру првог дела регистра се учитају почетне вредности кубита тако да вредност овог регистра буде суперпозиција свих целих бројева $[0 ; q - 1]$. Вредност свих кубита у другом делу регистра се поставља на $|0\rangle$. Овај корак извршава искључиво квантни рачунар. Након што се изврши овај корак стање у квантном регистру износи $\frac{1}{\sqrt{q}} \cdot \sum_{a=0}^{q-1} |a, 0\rangle$.
6. Затим за сваки број a који се налази у првом делу регистра обављамо трансформацију $x^a \bmod N$ и чувамо резултат у другом делу регистра. Због квантног паралелизма ово се обавља у само једном кораку. Стање у квантном регистру након овог корака износи $\frac{1}{\sqrt{q}} \cdot \sum_{a=0}^{q-1} |a, x^a \bmod N\rangle$.
7. Измери се израчуната вредност k која се налази у другом делу регистра. Ова операција као последицу има колапс вредности које се налазе у оквиру првог регистра на све вредности a између 0 и $q - 1$ за које важи $x^a \bmod N = k$, при чему се са A означава скуп ових вредности, а са $\|A\|$ величина тог скупа. Након што се одради ова операција стање квантног регистра је $\frac{1}{\sqrt{\|A\|}} \cdot \sum_{a' \in A} |a', k\rangle$.
8. Примењује се операција дискретне Фуријеове трансформације на први регистар. Након овога вредност квантног регистра постаје: $\frac{1}{\sqrt{\|A\|}} \cdot \sum_{a' \in A} \frac{1}{\sqrt{q}} \sum_{c=0}^{q-1} |c, k\rangle \cdot e^{\frac{2\pi i a' c}{q}}$.
9. Измери се вредност из првог регистра m , за коју постоји висока вероватноћа да је умножак од броја $\frac{q}{r}$.
10. На обичном рачунару се, на основу вредности бројева m и q , различитим техникама пост-процесирања долази до вредности периода r . Уколико је r непаран број враћамо се на корак 2 од алгоритма. Исто, уколико важи $\left(x^{\frac{r}{2}} + 1\right) \equiv 0 \bmod N$, враћамо се у корак 2 од алгоритма. У супротном извршава се следећи корак.

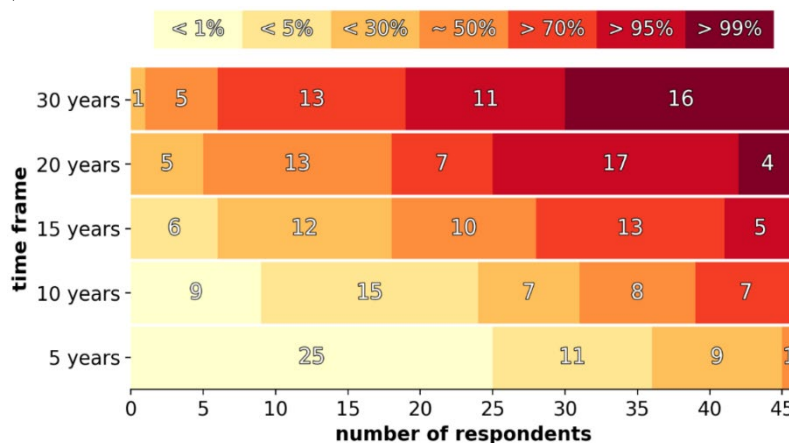
11. Користимо Еуклидов алгоритам на обичном рачунару да израчунамо вредност $\gcd(x^{\frac{r}{2}} - 1, N)$, која представља нетривијални фактор броја N , чиме се алгоритам завршава.

Сложеност приказаног квантног алгоритма за факторизацију је полиномијална и износи $O((\log N)^2 \cdot \log \log N)$, док сложеност најефикаснијег алгоритма на обичном рачунару – *GNFS*, износи $O(e^{(\log N)^{\frac{1}{3}} \cdot (\log \log N)^{\frac{2}{3}}})$. [24]

Поред приказаног алгоритма за факторизацију целих бројева, у оквиру [20] дат је и алгоритам за решавање проблема дискретних логаритама у коначном пољу, а постоје и имплементације алгоритма које подржавају решавање проблема дискретних логаритама над елиптичним кривама, које су конструисане над ограниченим пољем [27]. Постојање ових алгоритама смешта поменуте проблеме у *BQP* класу комплексности, класу проблема које квантни рачунар са довољно ресурса може да реши у полиномијалном времену са одређеним степеном грешке, а самим тим асиметрични алгоритми који користе поменуте проблеме као *one-way* функције за генерацију кључа постају несигурни на нападе квантних рачунара.

Демонстрација овог алгоритма је одрађена на квантном рачунару са 7 кубита 2001. године [28], чиме је успешно факторисан број 15. Тренутни рекорд је факторизација броја 21 2012. године [29], док је 2019. постојао покушај факторизације броја 35 коришћењем *IBM Q System One* квантног рачунара [30].

Постоје различита мишљења о томе кад ће развој квантних рачунара довести до имплементације квантног рачунара са одговарајућим бројем кубита и квантних логичких кола, таквог да може да искористи Шоров алгоритам у пуном потенцијалу и да угрози све тренутно коришћене асиметричне алгоритме. У оквиру извештаја [31] обављена је анкета са 47 научника и експерата који се баве истраживањем у оквиру области квантних рачунара и алгоритама из више земаља, при чему једна од тема у оквиру анкете процена за који временски период ће бити развијен квантни рачунар који ће моћи да за период мањи од једног дана разбије сигурност често коришћеног *RSA* алгоритма са 2048-битном величином броја модула. На слици 5.3 је приказан график резултата поменуте анкете, при чему су испитаници бирали вероватноће развоја квантног рачунара који ће да разбије поменути алгоритам за различите периоде времена, а график приказује укупан број испитаника који су за дати временски период дали исте процене.



Слика 5.3 – Резултати анкете [31]

6. Пост-квантни асиметрични алгоритми

У оквиру овог поглавља биће изложена анализа алгоритама финалиста треће рунде *NIST*-овог такмичења за стандардизацију пост-квантне криптографије (енг. *Post-Quantum Cryptography, PQC*), у оквиру категорије алгоритама намењених за размену кључева (енг. *Key Encapsulation Mechanism, KEM* категорија) и енкрипцију јавним кључем: *Classic McEliece*, алгоритма заснованог на линеарним кодовима, као и *CRYSTALS-Kyber*, *NTRU* и *Saber*, алгоритама који су засновани на решеткама. У категорији алгоритама заснованих на решеткама, биће изабран максимално један алгоритам за процес стандардизације.

Поред наведених алгоритама финалиста, постоји и алтернативна категорија, у којој се налазе алгоритми за размену кључева и енкрипцију јавним кључем који користе другачије приступе у енкрипцији, при чему ће изабрани алгоритми из групе финалиста први бити стандардизовани. Слична групација на финалисте и алтернативне алгоритме се јавља и код кандидата алгоритама за генерисање дигиталних потписа.

6.1. *Classic McEliece*

McEliece криптосистем је први од асиметричних алгоритама за енкрипцију са јавним кључем који је заснован на линеарним кодовима (код овог алгоритма конкретно на насумичним бинарним *Goppa* кодовима), који је развијен и објављен 1978. године у оквиру рада [33] од стране Роберта Мекелиса и такође представља прву асиметричну шему која користи насумичне вредности у оквиру алгоритма за енкрипцију, коришћењем насумичне грешке која се додаје у оригиналну поруку. Иницијално није прихваћен за процес стандардизације и није био често коришћен, због великих кључева у односу на друге алгоритме (нпр. *RSA*). [34]

Првобитно *McEliece* алгоритам је дизајниран да буде безбедан против основног типа нападача *OW-CPA*. Сигурносни ниво *McEliece* алгоритма остао је стабилан кроз године, иако је објављено више радова који демонстрирају потенцијалне нападе на њега. Иницијални параметри су дизајнирани за сигурност дистрибуције кључа величине 64 бита, при чему се могу скалирати да би се постигла изузетна сигурност против напада класичних и квантних рачунара. [34]

Classic McEliece представља алгоритам који је надоградња на обичан *McEliece* алгоритам која пружа високу сигурност против нападача категорије *IND-CCA2*, који на располагању има обичан и квантни рачунар.

Бинарни *Goppa* код се дефинише користећи полином $g(x) = g_t x^t + \dots + g_2 x^2 + g_1 x^1 + g_0$ степена t над ограниченим пољем у ознаци \mathbb{F}_{2^m} (или $GF(2^m)$) и користећи вектор различитих вредности из поља $L = (\alpha_1, \alpha_2, \dots, \alpha_n)$, при чему за полином важи $g(x_i) \neq 0$ за свако $x_i \in \mathbb{F}_{2^m}$, у ознаци [35]:

$$\Gamma(g, L) = \left\{ c = (c_1, c_2, \dots, c_n) \in \mathbb{F}_{2^m}^n : \sum_{i=1}^n \frac{c_i}{x - \alpha_i} \equiv 0 \bmod g(x) \right\}$$

Полином $g(x)$ се такође назива *Goppa* полином, док је вектор L познат као носећи (енг. *support*) вектор. Бинарни *Goppa* кодови могу да исправе максимално t грешки које се догоде у току преноса података, док димензија *Goppa* кода износи $\dim_{\mathbb{F}_2^m}(\Gamma(g, L)) \geq n - m \cdot t$.

Вредност $\frac{1}{x - \alpha_i}$ или $(x - \alpha_i)^{-1}$ из дефиниције *Goppa* кода се рачуна као мултипликативно инверзан елемент елементу $(x - \alpha_i)$ у оквиру прстена $\frac{\mathbb{F}_2^m(x)}{g(x)}$ и та вредност износи $\left(\frac{g(x) - g(\alpha_i)}{x - \alpha_i}\right) \cdot g(\alpha_i)^{-1}$. На основу наведеног може се написати да кодна реч $c \in \Gamma(g, L)$ акко важи $\sum_{i=1}^n \left(\frac{g(x) - g(\alpha_i)}{x - \alpha_i}\right) \cdot g(\alpha_i)^{-1} \equiv 0 \mod g(x)$. Користећи ову особину се може извести матрица провере парности H [35]:

$$H = \begin{bmatrix} g(\alpha_1)^{-1} & g(\alpha_2)^{-1} & \dots & g(\alpha_n)^{-1} \\ \alpha_1 \cdot g(\alpha_1)^{-1} & \alpha_2 \cdot g(\alpha_2)^{-1} & \dots & \alpha_n \cdot g(\alpha_n)^{-1} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{t-1} \cdot g(\alpha_1)^{-1} & \alpha_2^{t-1} \cdot g(\alpha_2)^{-1} & \dots & \alpha_n^{t-1} \cdot g(\alpha_n)^{-1} \end{bmatrix}_{t \times n}$$

Уколико имамо дат бинарни *Goppa* код $\Gamma(g, L)$ (где је степен полинома g једнак t , док је величина вектора L једнака n , а димензија кода $\dim_{\mathbb{F}_2^m}(\Gamma(g, L)) = k$) и матрицу $G_{k \times n}$ која представља генераторску матрицу за дати бинарни *Goppa* код, енкодирање вектора поруке m дужине k се ради користећи операцију множења вектора поруке и генераторске матрице, тј. важи $c_{1 \times n} = m_{1 \times k} \cdot G_{k \times n}$ [35].

Уколико би се овакав вектор енкодоване поруке пренео путем канала који има шум, на пријему би добили вектор $y = (y_1, y_2, y_3, \dots, y_n) = (c_1, c_2, c_3, \dots, c_n) + (e_1, e_2, e_3, \dots, e_n) = c + e$, где c означава вектор оригиналне енкодоване поруке, док e представља вектор грешке. Да би пријемник успешно декодовао добијену поруку, мора прво да отклони све грешке из ње.

Да би пријемник поруке уопште био у могућности да исправи грешке број ненултих вредности у оквиру вектора грешке мора да буде мањи или једнак максималном броју грешки које код може да исправи (максимално t код бинарних *Goppa* кодова).

Уколико усвојимо да се у оквиру кода јавило r грешки, са скупом позиција грешки $B = \{i : 1 \leq i \leq n, e_i \neq 0\}$, у циљу исправљања грешки у коду, дефинисаћемо полином за лоцирање грешке $\sigma(x)$ степена r , полином за евалуацију вредности грешке $w(x)$ степена вредности $r - 1$ и синдром примљеног кода $S(y)$, који детектује да ли је уопште дошло до грешке [35]:

$$\begin{aligned} - \sigma(x) &= \prod_{i \in B} (x - \alpha_i) \\ - w(x) &= \sum_{i \in B} e_i \prod_{j \in B, j \neq i} (x - \alpha_j) \\ - S(y) &= \sum_{i=1}^n \frac{y_i}{x - \alpha_i} \mod g(x) = \sum_{i=1}^n \frac{c_i}{x - \alpha_i} \mod g(x) + \sum_{i \in B} \frac{e_i}{x - \alpha_i} \mod g(x) = \\ &= \sum_{i \in B} \frac{e_i}{x - \alpha_i} \mod g(x) \end{aligned}$$

За исправљање грешки код бинарних *Goppa* кодова се користи Патерсонов алгоритам за детекцију грешки, (чији су улаз бинарни *Goppa* код $\Gamma(g, L)$ и примљени вектор поруке y , а излаз вектор грешке e) који се извршава у више корака [35]:

1. Израчунати вредност синдрома $S(y)$ који представља полином у прстену $\frac{\mathbb{F}_2^m(x)}{g(x)}$.
2. Рачуна се полином $T(x) = S(y)^{-1} \bmod g(x)$. Овај корак се неће успешно извршити уколико је $S(y) = 0$, будући да тај полином нема инверзан елемент у прстену $\frac{\mathbb{F}_2^m(x)}{g(x)}$, чиме се долази до закључка да није дошло до грешке при преносу (тј. да су сви чланови вектора e једнаки нули) и завршава се алгоритам.
3. Рачуна се полином $P(x) = \sqrt{T(x)} + x \bmod g(x)$.
4. Рачунају се полиноми $u(x)$ и $v(x)$ користећи проширени Еуклидов алгоритам, при чему мора да важи $u(x) = v(x) \cdot P(x) \bmod g(x)$.
5. Рачуна се вредност полинома за лоцирање грешке $\sigma(x) = u(x)^2 + x \cdot v(x)^2$
6. Пронађу се решења полинома $\sigma(x)$, чиме се проналази вектор грешке e и алгоритам се завршава.

Након што се пронађу и елиминишу грешке у оквиру добијеног вектора кодиране поруке, потребно је декодирање вектора да би дошли до оригиналне поруке, рачунањем решења система једначина, при чему $G^T_{n \times k}$ представља транспоновану генераторску матрицу [35]:

$$\left[G^T \mid \begin{array}{c} c_1 \\ \vdots \\ c_n \end{array} \right]$$

McEliece алгоритам за *one-way trap-door* функцију користи проблем декодирања генералног линеарног кода, при чему су нападачу познати коришћени линеарни код $[n, k]$ \mathcal{C} , цео број t (који представља максималан број грешки који дати код може да исправи) и вектор енкодване поруке са грешкама c . Нападач на основу познатих података треба да пронађе одговарајућу кодну реч $x \in \mathcal{C}$, при чему мора да важи $d(x, c) \leq t$, где d представља Хемингову дистанцу две кодне речи.

McEliece алгоритам се може дефинисати приказом низа његових функционалности са дијаграма енкрипције јавним кључем и декрипције приватним кључем [35] [34]:

- Генерисање кључева: Боб генерише свој пар приватног и јавног кључа бирањем насумичног бинарног *Goppa* кода $\Gamma(g, L)$, са генераторском матрицом $G_{k \times n}$, који има могућност да исправи t грешки, при чему је t степен полинома g . Затим се генеришу насумична инвертибилна бинарна матрица $S_{k \times k}$ и пермутациона матрица $P_{n \times n}$. Након тога се рачуна матрица $\hat{G}_{k \times n} = S_{k \times k} \cdot G_{k \times n} \cdot P_{n \times n}$, при чему је Бобов јавни кључ (\hat{G}, t) , који се на сигуран начин прослеђује Алиси, док је Бобов приватни кључ (S, G, P) .
- Енкрипција јавним кључем: Алиса на основу добијеног Бобовог јавног кључа (\hat{G}, t) треба да енкриптује поруку m дужине k и пошаље је Бобу. Прво ће да генерише насумичан вектор z дужине n бита са Хеминговом тежином t , који представља вектор насумичне грешке. Након тога, Алиса рачуна енкриптовану поруку $c = m \cdot \hat{G} + z$ и проследи је Бобу.
- Декрипција приватним кључем: Након што добије енкриптовану поруку c , Боб користи свој приватни кључ да израчуна инверзну пермутациону матрицу P^{-1} , на основу које рачуна $c \cdot P^{-1} = m \cdot S \cdot G + z \cdot P^{-1}$. Након тога Боб користи Патерсонов алгоритам да детектује и исправи грешку и на крају реши систем једначина чиме добија вредност вектор $\hat{m}_{1 \times k} = m_{1 \times k} \cdot S_{k \times k}$, из кога добијамо вредност оригиналне поруке $m = \hat{m} \cdot S^{-1}$.

Classic McEliece представља пост-квантни асиметрични алгоритам за размену кључева који користи *Niederreiter*-ову дуалну верзију *McEliece* алгоритма са бинарним *Goppa* кодовима, при чему алгоритам се може приказати низом функционалности, при чему се подразумева да је поље \mathbb{F}_{2^m} познато Алиси и Бобу [35] [34]:

- Генерисање кључева: Боб бира насумични бинарни *Goppa* код $\Gamma(g, L)$ над пољем \mathbb{F}_{2^m} генерисањем полинома $g(x)$ и вектора L . Затим се рачуна матрица парности $\tilde{H}_{t \times n} = \{h_{i,j}\}$, при чему елемент $h_{i,j} = \alpha_j^{i-1} g(\alpha_i)^{-1}$, $i \in [1, t]$, $j \in [1, n]$. Након овога врши се замена елемената матрице $\tilde{H}_{t \times n}$ са векторима из поља \mathbb{F}_2^m , који су организовани по колонама матрице, користећи особину изоморфизма између поља \mathbb{F}_2^m и \mathbb{F}_{2^m} , чиме се добија матрица $\hat{H}_{mt \times n}$. На ову матрицу се примењује метода Гаусове елиминације, чиме се добија матрица система $H_{mt \times n} = (I_{mt} | T_{mt \times (n-mt)})$. На крају је потребно генерисати насумичан вектор s величине n битова. Бобов јавни кључ је сад матрица T , док је његов приватни кључ $(s, \Gamma(g, L))$.
- Енкрипција јавним кључем: Алиса користи Бобов јавни кључ T да израчуна своју матрицу H , а затим на основу матрице и насумично генерисаног вектора грешке дужине n битова са Хеминговом тежином t рачуна вредност првог дела енкриптоване поруке, вектор дужине mt $C_0 = H \cdot e$ у пољу $\mathbb{F}_{2^m}^{mt}$. Такође рачуна вредност $C_1 = \text{SHAKE256}(2, e)$, при чему *SHAKE256* представља хеш функцију, а први параметар се представља као бајт. Бобу прослеђује енкриптована порука $C = (C_0, C_1)$ укупне дужине $mt + 256$ битова, а Алиса рачуна тајни кључ као $K = \text{SHAKE256}(1, e, C)$.
- Декрипција приватним кључем: Боб добија енкриптовану поруку C од Алисе, и дели је на два дела C_0 и C_1 . Након овога се поставља вредност броја b на 1 и екстендује величина вектора C_0 до дужине n битова, додавањем $n - mt$ нултих битова на крај вектора, чиме добијамо вектор $v = (C_0, 0, \dots, 0) \in \mathbb{F}_2^n$. Након овога је потребно да се у оквиру бинарног *Goppa* кода $\Gamma(g, L)$ пронађе јединствена кодна реч c за коју важи $d(c, v) \leq t$ користећи *Niederreiter*-ов процес декодирања, при чему уколико се не нађе кодна реч алгоритам се поново извршава. Након што је пронађена кодна реч c , рачуна се вектор грешке $e = v + c$. Уколико не важе услови да је Хемингова тежина вектора e једнака t и да важи $C_0 = H \cdot e$, вектор грешке e постаје једнак вектору s , док b добија вредност 0 (ово се ради у оквиру имплементације алгоритма да би се избегли *side-channel* напади). Након провере услова, рачуна се вредност $C'_1 = \text{SHAKE256}(2, e)$, која се након тога пореди са добијеном вредношћу C_1 и уколико не важи једнакост $C_1 = C'_1$, вектор e постаје једнак вектору s , а број b добија вредност 0. На крају се рачуна вредност тајног кључа као $K = \text{SHAKE256}(b, e, C)$. Уколико није било потешкоћа у преносу поруке Алиса и Боб долазе до истог тајног кључа, што могу додатно да верификују слањем и поређењем хеш вредности својих генерисаних тајних кључева.

Постоји више параметара који су предложени за коришћење, са различитим нивоима процењене безбедности задатим од стране *NIST*-а, што је приказано у оквиру табеле 6.1.

Назив сета параметара	m	n	t	Ниво сигурности
<i>mcEliece348864</i>	12	3488	64	1
<i>mcEliece348864f</i>	12	3488	64	1
<i>mcEliece460896</i>	13	4608	96	3
<i>mcEliece460896f</i>	13	4608	96	3
<i>mcEliece6688128</i>	13	6688	128	5
<i>mcEliece6688128f</i>	13	6688	128	5
<i>mcEliece6960119</i>	13	6960	119	5
<i>mcEliece6960119f</i>	13	6960	119	5
<i>mcEliece8192128</i>	13	8192	128	5
<i>mcEliece8192128f</i>	13	8192	128	5

Табела 6.1 – Параметри Classic McEliece алгоритма [34]

6.2. CRYSTALS-Kyber

CRYSTALS-Kyber представља пост-квантни асиметрични алгоритам за размену кључева и енкрипцију јавним кључем који је део *CRYSTALS* (енг. *Cryptographic Suite for Algebraic Lattices*) пакета пост-квантних алгоритама, у које је укључен и *Dilithium*, пост-квантни алгоритам за генерисање дигиталних потписа и представља рад више аутора. Алгоритам се састоји из два дела – *Kyber.CPAPKE*, *IND-CPA* сигурног алгоритма за енкрипцију јавним кључем као и *Kyber.CCAKEM IND-CCA2* сигурног алгоритма за размену кључева који представља модификацију претходно поменутог алгоритма, додавањем *Fujisaki-Okamoto (FO)* трансформације. *CRYSTALS-Kyber* у оквиру обе имплементације за *one-way trap-door* функцију користи учење са грешкама. [36]

Учење са грешкама (енг. *Learning With Errors, LWE*) представља један од често коришћених проблема који се јавља у оквиру криптографске примене решетки. Уколико имамо дефинисана два позитивна цела броја d и q , тајни вектор полинома $s \in \mathbb{Z}_q^d$, $LWE_{d,q,\psi}$ узорак се дефинише као пар $(a, \frac{1}{q}\langle a, s \rangle + e \bmod q)$, при чему $\langle a, s \rangle$ представља операцију унутрашњег производа вектора a и s , где се a бира из униформне дистрибуције над \mathbb{Z}_q^d , док e представља вредност полинома грешке која се генерише насумично користећи дистрибуцију ψ . Дефинишимо проблем претраге, где је потребно на основу произвољног броја парова узорака из $LWE_{d,q,\psi}$ доћи до тајног вектора s . Додатни проблем који постоји код учења са грешком је ткзв. проблем одлуке, при чему се тражи начин да се разликују произвољни узорци из $LWE_{d,q,\psi}$ од истог броја насумичних узорака генерисаних из униформне дистрибуције, при чему је овај проблем лакши за решавање од проблема претраге. Проблем се може пресликати на математику решетки, уколико усвојимо да је A матрица основе решетке (са базним векторима у колонама), док множењем са тајним вектором s добијамо једну тачку у оквиру

решетке тј. један вектор, на који додајемо грешку, чиме проблем претраге постаје проблем налажења најближег вектора (CVP). [37]

У оквиру обе имплементације се такође користи и *NTT* алгоритам (енг. *Number Theoretic Transform*), који представља ефикасан алгоритам за мултипликацију полинома чији су коефицијенти из ограниченог поља или прстена. Користећи *NTT* функције могуће је на ефикасан начин израчунати производ $f \cdot g$ два елемента $f, g \in \mathbb{Z}_q[x]/(x^n + 1)$, у облику $f \cdot g = NTT^{-1}(NTT(f) \circ NTT(g)) = NTT^{-1}(\hat{f} \circ \hat{g})$, где \hat{f} и \hat{g} представљају два полинома у облику погодном за рачунање користећи *NTT* алгоритам. Такође се користе и функције *Compress_q* и *Decompress_q*, којима се одбацују нижи битови енкриптоване поруке који немају значаја у тачности енкрипције, чиме се постиже мања величина енкриптоване поруке, и функције *Encode* и *Decode*, којима се врши серијализација (из полинома у низ бајтова) и десеријализација (из низа бајтова у полином) вектора полинома и полинома у оквиру алгоритма. [36]

Kyber.CPAPKE ће бити приказан у оквиру низа функционалности при чему важе следеће дефиниције R као прстен $\mathbb{Z}[x]/(x^n + 1)$, R_q као прстен $\mathbb{Z}_q[x]/(x^n + 1)$, док параметри n , n' и q износе $n = 256$, $n' = 9$, $q = 3329$, а параметри k, d_u, d_v, η_1 и η_2 зависте од усвојеног сета параметара [36]:

- Генерисање кључева: Боб бира насумични број d величине 32 бајта, који се користи у оквиру хеш функције G за генерисање бројева ρ и σ ($\rho, \sigma = G(d)$). Након овога се генерише матрица $\hat{A} \in R_q^{k \times k}$ у *NTT* домену, чији елементи представљају полиноме и генеришу се као $\hat{A}[i][j] = Parse(XOF(\rho, j, i))$, где *XOF* представља или симетричан алгоритам или хеш функцију, док *Parse* представља методу за бирање узорка из R_q тако да ефективно симулира узимање елемената из насумичне униформне расподеле. Након овога се генеришу вектор полинома $s \in R_q^k$, који је представља приватни кључ и вектор полинома $e \in R_q^k$ који представља вектор грешке на основу кога се генерише јавни кључ, преко формула $s[i] = CBD_{\eta_1}(PRF(\sigma, i))$, $e[i] = CBD_{\eta_1}(PRF(\sigma, i + k))$, при чему *CBD* представља центрирану биномску дистрибуцију, док *PRF* представља хеш функцију или симетричан алгоритам за енкрипцију. Затим се генеришу вредности за *NTT* алгоритам $\hat{s} = NTT(s)$ и $\hat{e} = NTT(e)$, након чега се врши генерација вредности јавног кључа $\hat{t} = \hat{A} \circ \hat{s} + \hat{e}$. Бобов јавни кључ представља енкодвана вредност (\hat{t}, ρ) , док вредност приватног кључа износи енкодвана вредност \hat{s} .
- Енкрипција јавним кључем: Алиса генерише насумични број података d величине 32 бајта и добија вредности \hat{t} и ρ из Бобовог јавног кључа. Затим Алиса генерише матрицу $\hat{A}^T \in R_q^{k \times k}$ у *NTT* домену, $\hat{A}^T[i][j] = Parse(XOF(\rho, i, j))$. Након овога се генеришу вектори полинома грешке $r \in R_q^k$ и $e_1 \in R_q^k$, користећи формуле $r[i] = CBD_{\eta_1}(PRF(d, i))$ и $e_1[i] = CBD_{\eta_2}(PRF(d, i + k))$, након чега се генерише додатни полином који исто представља грешку $e_2 = CBD_{\eta_2}(PRF(d, 2 \cdot k))$. На основу вектора r конструишемо $\hat{r} = NTT(r)$. На крају Алиса израчуна вектор полинома u и полином v , користећи формуле $u = NTT^{-1}(\hat{A}^T \circ \hat{r}) + e_1$ и $v = NTT^{-1}(\hat{t}^T \circ \hat{r}) + e_2 + Decompress_q(Decode(m), 1)$, где m представља полином поруке коју Алиса жели да енкриптује. Делови енкриптоване поруке c_1 и c_2 се рачунају као $c_1 =$

$Encode(Compress_q(u, d_u))$ $c_2 = Encode(Compress_q(v, d_v))$. Енкриптована порука представља пар (c_1, c_2) .

- Декрипција приватним кључем: Након што добије енкриптовану поруку од Алисе, Боб је дели на делове c_1 и c_2 и рачуна вектор u и полином v преко формула $u = Decompress_q(Decode(c_1), d_u)$ и $v = Decompress_q(Decode(c_2), d_v)$. Након овога се рачуна вредност оригиналне поруке m коју је Алиса послала, $m = Encode(Compress_q(v - NTT^{-1}(\hat{s} \circ NTT(u)), 1))$.

Генерална идеја иза функције декрипције код *Kyber.CPAPKE* је да се из израза у наставку $v - s \cdot u = t^T \cdot r + e_2 + m - s \cdot (A^T \cdot r + e_1) = m - s \cdot e_1 + e_2 + r \cdot (t^T - s \cdot A^T)$ извуче полином енкодоване поруке m коришћењем функције $Compress_q$ којом се врши корекција грешки заокруживањем. Ово је могуће будући да се у оквиру процеса енкрипције јавним кључем, при рачунању вредности полинома v порука m претвара из низа бајтова у полином користећи $Decode$ функцију, а затим се над датим полиномом поруке примени функција $Decompress_q$, којом се креира толерантност на грешку, тако што енкодује сваки бит који је једнак нули у оквиру поруке на 0, док се битови једнаки јединици енкодују на вредност $\left\lfloor \frac{q}{2} \right\rfloor$, чиме се додаје функционалност заокруживања у оквиру функције $Compress_q$ – одређивања да ли је коефицијент ближи 0 или $\left\lfloor \frac{q}{2} \right\rfloor$. Коришћењем заокруживања за корекцију *LWE* грешке уводи се и особина која није постојала код класичних алгоритама или код *McEliece* алгоритама – постоји мала вероватноћа да ће декрипција поруке бити неуспешна, тј. да генерисани тајни кључеви неће бити једнаки у случају размене кључа. Ово представља одређени компромис, будући да енкрипционе шеме засноване на *LWE* проблему код којих није могућа грешка имају мању сигурност од напада. [36]

У наставку приказане су функције *PRF*, *XOF*, *H*, *G* и *KDF* које су коришћене у оквиру алгоритама, које важе за сетове параметара чији се назив не завршава са 90s [36]:

- за функцију *PRF*(s, b) користи се хеш функција *SHAKE256*($s||b$)
- за функцију *XOF* користи се хеш функција *SHAKE128*
- за функцију *H* користи се хеш функција *SHA3_256*
- за функцију *G* користи се хеш функција *SHA3_512*
- за функцију *KDF* користи се хеш функција *SHAKE256*.

Уколико алгоритам користи сет параметара чији се назив завршава са 90s користе се следеће функције [36]:

- за функцију *PRF*(s, b) користи се *AES256* у *CTR* моду, при чему је s коришћен као кључ, док се b допуњује до 12-бајтног *nonce*-а, а бројач *CTR* мода се иницијализује на нулу
- за функцију *XOF*(p, i, j) користи се *AES256* у *CTR* моду, при чему је p коришћен као кључ, док се $i||j$ допуњује до 12-бајтног *nonce*-а, а бројач *CTR* мода се иницијализује на нулу
- за функцију *H* користи се хеш функција *SHA256*
- за функцију *G* користи се хеш функција *SHA512*

- за функцију KDF користи се хеш функција $SHA256$.

$Kyber.CCAKEM$ користи функције генерисања кључа, енкрипције јавним кључем и декрипције приватним кључем $Kyber.CPAPKE$ алгоритма које се могу означити са $KeyGen()$, $Enc()$ и $Dec()$ респективно, при чему се примењује *Fujisaki-Okamoto* трансформација са малим изменама. Алгоритам се може приказати преко функционалности [36]:

- Генерисање кључева: Боб генерише насумични број z величине 32 бајта, након чега користи функцију генерације кључа којом добија $(pk, sk') = KeyGen()$, при чему се коначни приватни кључ облика $sk = (sk', pk, H(pk), z)$. Јавни кључ остаје исти као и код $Kyber.CPAPKE$ алгоритма.
- Енкрипција јавним кључем: Након што добије Бобов јавни кључ, Алиса генерише насумични број m величине 32 бајта, коме промени вредност користећи хеш функцију $m = H(m)$. Након овога се рачунају $(\bar{K}, d) = G(m || H(pk))$, након чега се позива функција за енкрипцију јавним кључем преко које добијамо енкриптовану поруку $c = Enc(pk, m, d)$, са изменом да се користи насумичан вектор података d и да се исти не генерише поново у оквиру енкрипционе функције. Генерисани дељени тајни кључ се на крају рачуна као $K = KDF(\bar{K} || H(c))$, док се генерисана енкриптована порука прослеђује Бобу.
- Декрипција приватним кључем: Боб на основу свој приватног кључа $(sk', pk, H(pk), z)$ и на основу добијене енкриптоване поруке c генерише $m' = Dec(sk', c)$ и $h = H(pk)$, након чега рачуна $(\bar{K}', d') = G(m' || h)$, након чега рачуна своју верзију енкриптоване поруке $c' = Enc(pk, m', d')$. Уколико су две енкриптоване поруке c и c' једнаке, Бобов тајни кључ се генерише као $K = KDF(\bar{K}' || H(c))$ и он је једнак кључу који је генерисала Алиса. У супротном се генерише као $K = KDF(z || H(c))$, при чему је Алисин кључ различит од Бобовог и неће моћи да успоставе комуникацију, тј. Алиса би поново треба да генерише кључ и пошаље нову енкриптовану поруку.

У оквиру табеле 6.2 приказани су различити сетови параметара које користи $Kyber.CCAKEM$, при чему параметар означен са δ означава вероватноћу да ће се при заокруживању у оквиру процеса декрипције десити грешка.

Назив сета параметара	k	η_1	η_2	(d_u, d_v)	δ	Ниво сигурности
$Kyber512$	2	3	2	(10, 4)	2^{-139}	1
$Kyber768$	3	2	2	(10, 4)	2^{-164}	3
$Kyber1024$	4	2	2	(11, 5)	2^{-174}	5
$Kyber512-90s$	2	3	2	(10, 4)	2^{-139}	1
$Kyber768-90s$	3	2	2	(10, 4)	2^{-164}	3
$Kyber1024-90s$	4	2	2	(11, 5)	2^{-174}	5

Табела 6.2 – Параметри $Kyber$ алгоритма [36]

6.3. NTRU

NTRU представља асиметрични пост-квантни алгоритам заснован на решеткама чија је прва верзија објављена у раду [38]. У почетку су постојале две имплементације – *NTRUEncrypt*, који је коришћен за енкрипцију јавним кључем и *NTRUSign*, који је коришћен за генерисање дигиталних потписа. Од наведених алгоритама, *NTRUEncrypt* је патентиран 2002. године под америчким патентним бројем 6.081.597, док је прешао у јавно власништво 2017. године, док је *NTRUSign* и даље патентиран али може да се користи под *GPL* лиценцом. [40]

Верзија алгоритма која је предложена за *PQC* стандардизацију представља *IND-CCA2* сигуран алгоритам за размену кључева који је настао као комбинација ранијих верзија – *NTRUEncrypt* и *NTRU-HRSS-KEM*, при чему је сам алгоритам размене кључева заснован на *OW-CPA* сигурној детерминистичкој шеми за енкрипцију јавним кључем *DPKE* (енг. *Deterministic Public Key Encryption Scheme*). [39]

NTRU алгоритам, тј. његова *DPKE* шема, за *one-way* функцију користи проблем факторизације полинома у прстенима, у случају кад је полином који је потребно факторисати настао као конволуциони производ два полинома са малим коефицијентима. Анализа поменутог проблема је врло слична анализи проблема који се користе у оквиру решетки, на основу чега се алгоритам сврстава у категорију алгоритама заснованих на решеткама.

Као параметри у оквиру *NTRU* алгоритма користе се позитивни цели бројеви (n, p, q) који су међусобно прости, простори или скупови из којих се бирају полиноми f и g који су потребни за генерацију кључева, полином грешке (или ткзв. *blinding* полинома) r који је потребан за процес енкрипције поруке и простор за генерисање полинома порука $(\mathcal{L}_f, \mathcal{L}_g, \mathcal{L}_r, \mathcal{L}_m)$, као и инјективно пресликавање $Lift: \mathcal{L}_m \rightarrow \mathbb{Z}[x]$, при чему постоје два типа параметара – *NTRU-HPS* и *NTRU-HRSS*, који су добили назив по ауторима и који се разликују по типовима простора који се користе, као и по вредности броја q . [39]

У наставку су приказани описи симбола и операција које су коришћене у оквиру алгоритма:

- Φ_1 представља полином $x - 1$
- Φ_n представља полином $\frac{x^n - 1}{x - 1} = x^{n-1} + x^{n-2} + \dots + 1$
- $a = b \bmod(q, \Phi_n)$ представља операцију којом се за дати улазни полином $a \in \mathbb{Z}[x]$ рачуна јединствени полином $b \in \mathbb{Z}[x]$ чији степен максимално може бити једнак $n - 2$, са коефицијентима у скупу $\{-\frac{q}{2}, -\frac{q}{2} + 1, \dots, \frac{q}{2} - 2, \frac{q}{2} - 1\}$
- $a = b \bmod(p, \Phi_n)$ представља операцију којом се за дати улазни полином $a \in \mathbb{Z}[x]$ рачуна јединствени полином $b \in \mathbb{Z}[x]$ чији степен максимално може бити једнак $n - 2$, са коефицијентима у скупу $\{-1, 0, 1\}$
- $a = b \bmod(p, \Phi_1 \Phi_n)$ представља операцију којом се за дати улазни полином $a \in \mathbb{Z}[x]$ рачуна јединствени полином $b \in \mathbb{Z}[x]$ чији степен максимално може бити једнак $n - 1$, са коефицијентима у скупу $\{-\frac{q}{2}, -\frac{q}{2} + 1, \dots, \frac{q}{2} - 2, \frac{q}{2} - 1\}$
- \mathcal{T} представља скуп ненултих тернарних полинома (полиноми са коефицијентима у скупу $\{-1, 0, 1\}$) максималног степена $n - 2$

- \mathcal{T}_+ представља подскуп скупа \mathcal{T} , где сви полиноми имају особину ненегативне корелације (за сваки тернарни полином $v \in \mathcal{T}_+$ облика $v = \sum_i v_i \cdot x^i$ важи релација $\sum_i v_i \cdot v_{i+1} \geq 0$)
- $\mathcal{T}(d)$ представља подскуп скупа \mathcal{T} са полиномима који имају тачно $\frac{d}{2}$ коефицијената једнаких -1 и тачно $\frac{d}{2}$ коефицијената једнаких $+1$.

У наставку су дефинисане операције које се јављају у оквиру *DPKE* шеме коју користи *NTRU*, при чему се подразумева да су Алиса и Боб претходно разменили параметре алгоритма [39]:

- Генерисање кључева: Боб генерише два тернарна полинома f и g из простора \mathcal{L}_f и \mathcal{L}_g на насумичан начин, након чега генерише полином $f_q = f^{-1} \bmod (q, \Phi_n)$. Након овога Боб генерише вредност јавног кључа $h = p \cdot g \cdot f_q \bmod (q, \Phi_1 \Phi_n)$ и делова приватног кључа $h_q = h^{-1} \bmod (q, \Phi_n)$, $f_p = f^{-1} \bmod (p, \Phi_n)$. Бобов јавни кључ представља полином h , док је Бобов приватни кључ (f, f_p, h_q) .
- Енкрипција јавним кључем: Алиса користи договорену функцију пресликавања да од поруке m која представља полином из простора порука \mathcal{L}_m дође до полинома $m' = \text{Lift}(m)$. Након овога Алиса користећи Бобов јавни кључ h генерише енкриптовану поруку $c = (r \cdot h + m') \bmod (q, \Phi_1 \Phi_n)$, при чему r представља тајни полином грешке са насумичним малим коефицијентима.
- Декрипција приватним кључем: Боб након што је примио енкриптовану поруку c проверава да ли важи једнакост $c = 0 \bmod (q, \Phi_1)$ и уколико не важи овде се зауставља процес декрипције и избацује се грешка, а уколико важи рачуна се привремени полином $a = (c \cdot f) \bmod (q, \Phi_1 \Phi_n) = ((r \cdot (p \cdot g \cdot f_q) + m') \cdot f) \bmod (q, \Phi_1 \Phi_n) = (p \cdot g \cdot r + m' \cdot f) \bmod (q, \Phi_1 \Phi_n)$, након чега се добија послата порука $m' = (a \cdot f_p) \bmod (p, \Phi_n)$, будући да важи једнакост $(p \cdot g \cdot r) = 0 \bmod (p, \Phi_n)$. На основу m' Боб може да дође до оригиналног полинома поруке m и да из њега извуче поруку. Такође на основу m' Боб може да дође и до полинома грешке r користећи формулу $r = ((c - m') \cdot h_q) \bmod (q, \Phi_n)$. Након овога се проверава да ли полиноми m и r припадају просторима \mathcal{L}_m и \mathcal{L}_r и уколико не припадају избацује се грешка.

У наставку су приказане функционалности *NTRU* алгоритма за размену тајних кључева који је предложен за стандардизацију, користећи функције генерисања кључева, енкрипције јавним кључем и декрипције приватним кључем *DPKE* шеме које су означене као *KeyGen()*, *Enc()*, *Dec()* [39]:

- Генерисање кључева: Боб користи функцију за генерисање кључа *DPKE* шеме $((f, f_p, h_q), h) = \text{KeyGen}()$, након чега генерише насумичан број од 256 бита s који се користи уместо декриптоване поруке за генерисање тајног кључа због сигурности уколико је дошло до грешке приликом декрипције, исто као и код *Classic McEliece* и *Kyber* алгоритма. За приватни кључ Боб узима (f, f_p, h_q, s) док је Бобов јавни кључ h .

- Енкрипција јавним кључем: Алиса генерише насумичан број од 256 бита, који користи да генерише полиноме грешке и поруке r и m . Након овога Алиса генерише енкриптовану поруку $c = Enc(h, r, m)$ и тајни кључ $K = SHA3_256(r, m)$, након чега се прослеђује енкриптована порука Бобу.
- Декрипција приватним кључем: Боб након што прими енкриптовану поруку покушава да је декриптује користећи свој приватни кључ $(r, m) = Dec((f, f_p, h_q), c)$. Уколико је декрипција успешна тајни кључ се генерише као $K = SHA3_256(r, m)$ и овај кључ ће бити једнак кључу који је генерисала Алиса. У супротном, уколико је дошло до грешке при декрипцији Боб генерише тајни кључ као $K = SHA3_256(s, c)$, при чему тајни кључеви Алисе и Боба неће бити једнаки и процес размене кључа мора да се одради од почетка.

Сетови параметара који су предложени у оквиру процеса стандардизације су приказани у табели 6.3. У оквиру *NTRU-HPS* параметара за скупове полинома важи $\mathcal{L}_f = \mathcal{T}, \mathcal{L}_g = \mathcal{T}(\frac{q}{8} - 2), \mathcal{L}_r = \mathcal{T}, \mathcal{L}_m = \mathcal{T}(\frac{q}{8} - 2)$ а функција *Lift* има облик $m \mapsto m$, док за *NTRU-HRSS* скупови имају облик $\mathcal{L}_f = \mathcal{T}_+, \mathcal{L}_g = \{\Phi_1 \cdot v : v \in \mathcal{T}_+\}, \mathcal{L}_r = \mathcal{T}, \mathcal{L}_m = \mathcal{T}$, а функција *Lift*: $m \mapsto \Phi_1 \cdot (\frac{m}{\Phi_1} \bmod (p, \Phi_n))$.

Назив сета параметара	n	p	q	Ниво сигурности
<i>NTRU-HPS-2048-509</i>	509	3	2048	1
<i>NTRU-HPS-2048-677</i>	677	3	2048	3
<i>NTRU-HPS-4096-821</i>	821	3	4096	5
<i>NTRU-HPS-4096-1229</i>	1229	3	4096	5
<i>NTRU-HRSS-701</i>	701	3	8192	3
<i>NTRU-HRSS-1373</i>	1373	3	16384	5

Табела 6.3 – Параметри *NTRU* алгорита [39]

6.4. *Saber*

Saber је пост-квантни асиметрични алгоритам који спада у породицу алгоритама заснованих на решеткама. Састоји се од *Saber.PKE* - *IND-CPA* шеме за енкрипцију јавним кључем, која се затим трансформише у *Saber.KEM* - *IND-CCA2* сигуран алгоритам за енкапсулацију кључева, користећи верзију *Fujisaki-Okamoto* трансформације. [41]

За основу сигурности у оквиру *Saber.PKE* алгоритма користи се *one-way trap-door* функција која је заснована на проблему учења са заокруживањем – *LWR* (енг. *Learning With Rounding*). Као и код *LWE* дат нам је насумични јавни вектор полинома $a \in \mathbb{Z}_q^d$ и насумични приватни вектор полинома $s \in \mathbb{Z}_q^d$, при чему се сада генеришу парови облика $(a, \lfloor \langle a, s \rangle \bmod q \rfloor_p)$, где други члан представља унутрашњи производ вектора a и s по модулу q , који се након тога детерминистички заокружује на $p < q$ битова. Овиме се практично елементи из прстена \mathbb{Z}_q^d деле на p суседних и непрекидних интервала, користећи функцију заокруживања $\lfloor \cdot \rfloor_p: \mathbb{Z}_q^d \mapsto \mathbb{Z}_p^d$. Уколико је грешка из *LWE* e довољно мала а однос $\frac{q}{p}$ довољно велики (нпр. већи за експоненцијални фактор од грешке из *LWE* e), важиће једнакост $(a, \lfloor \langle a, s \rangle \bmod q \rfloor_p) \approx^{stat} (a, \lfloor \langle a, s \rangle + e \bmod q \rfloor_p)$. *LWR* такође се може описати проблемима претраге и одлуке. *LWR* има предност у односу на *LWE*, будући да не захтева генерисање насумичне грешке из одређене расподеле, што смањује број коришћених битова и време извршавања. [42]

У наставку је дат преглед нотације и симбола који су коришћени у оквиру алгоритма [41]:

- R_q представља прстен $\mathbb{Z}_q[x]/(X^n + 1)$, при чему n представља параметар алгоритма
- $R_q^{l \times k}$ представља прстен матрица димензија $l \times k$ над прстеном R_q , чији су елементи полиному у оквиру R_q
- \ll и \gg представљају операције битског померања улево и удесно.

Параметри *Saber* алгоритма су описани у наставку [41]:

- $n, l - n = 256$ представља степен прстена полинома $\mathbb{Z}_q[x]/(X^n + 1)$, док l представља ранк модула који одређује димензије еквивалентног проблема у решеткама у облику $l \cdot n$
- q, p, T – ови параметри представљају модуле који се користе у оквиру криптографске шеме и сви су степени броја 2, у облику $q = 2^{\varepsilon_q}, p = 2^{\varepsilon_p}, T = 2^{\varepsilon_T}$, при чему мора да важи $\varepsilon_q > \varepsilon_p > \varepsilon_T$
- μ – представља параметар који утиче на коефицијенте вектора, будући да се коефицијенти у оквиру полинома тајних вектора s и s' бирају из центриране биномске дистрибуције $\beta_\mu(R_q^{l \times 1})$
- $\mathcal{F}, \mathcal{G}, \mathcal{H}$ - представљају хеш функције које се користе у оквиру алгоритма, при чему је за \mathcal{F} и \mathcal{G} усвојена имплементација хеш функције *SHA3_256*, док је \mathcal{H} имплементирана користећи *SHA3_512* хеш функцију
- gen – представља функцију која је коришћена у оквиру алгоритма за генерисање псеудонасумичне матрице $A \in R_q^{l \times l}$ на основу ткзв. *seed* вредности, која је имплементирана користећи *SHAKE128* хеш функцију.

Алгоритам такође користи и константе које се користе у оквиру имплементације алгоритма за замену операције заокруживања једноставним померањем битова [41]:

- константан полином $h_1 \in R_q$, чији су сви коефицијенти једнаки $2^{\varepsilon_q - \varepsilon_p - 1}$
- константан вектор полинома $h \in R_q^{l \times l}$ при чему је сваки полином у вектору једнак h_1
- константан вектор полинома $h_2 \in R_q$ чији су сви коефицијенти једнаки вредности $2^{\varepsilon_p - 2} - 2^{\varepsilon_p - \varepsilon_T - 1} + 2^{\varepsilon_q - \varepsilon_p - 1}$.

Saber.PKE алгоритам приказаћемо помоћу низа корака, при чему се подразумева да су обе стране претходно размениле одговарајуће параметре алгоритма [41]:

- Генерисање кључева: Боб у почетку генерише број $seed_A$ дужине 256 бита насумично из униформне расподеле. Након тога овај број се користи за генерисање матрице $A \in R_q^{l \times l}$ $A = gen(seed_A)$. Боб затим генерише насумични број r дужине 256 бита, који представља $seed$ који се користи за генерисање приватног вектора полинома $s = \beta_\mu(R_q^{l \times 1}, r)$ из центриране биномске дистрибуције. На крају се генерише вектор полинома $b = ((A^T \cdot s + h) \bmod q) \gg (\varepsilon_q - \varepsilon_p)$, при чему важи $b \in R_q^{l \times 1}$. Бобов јавни кључ је пар $(seed_A, b)$, док је Бобов приватни кључ вектор полинома s .
- Енкрипција јавним кључем: Алиса након што прими Бобов јавни кључ, енкодује поруку у полином дужине n $m \in R_2$, након чега генерише матрицу полинома A користећи $seed_A$ из Бобовог јавног кључа $A = gen(seed_A)$. Уколико број r од 256 бита није дат као аргумент генерисање се насумично. Затим Алиса генерише приватни вектор полинома $s' = \beta_\mu(R_q^{l \times 1}, r)$, након чега генерише јавни вектор полинома $b' = ((A^T \cdot s' + h) \bmod q) \gg (\varepsilon_q - \varepsilon_p)$, при чему важи $b' \in R_q^{l \times 1}$. Након овога Алиса рачуна привремено коришћен вектор $v' = b'^T \cdot (s' \bmod p) \in R_p$, који се користи у даљем рачунању енкриптоване поруке $c_m = (v' + h_1 - 2^{\varepsilon_p - 1} \cdot m \bmod p) \gg (\varepsilon_p - \varepsilon_T) \in R_T$. Енкриптована порука се представља као пар (c_m, b') .
- Декрипција приватним кључем: Боб након што прими енкриптовану поруку (c_m, b') од Алисе рачуна полином $v = b'^T (s \bmod p) \in R_p$, након чега долази до полинома оригиналне поруке $m' = ((v - 2^{\varepsilon_p - \varepsilon_T} \cdot c_m + h_2) \bmod p) \gg (\varepsilon_p - 1) \in R_2$.

Функције *Saber* алгоритма за енкапсулацију кључева – *Saber.KEM* дате су у наставку при чему се користе функције за генерисање кључева, енкрипцију и декрипцију из *Saber.PKE* шеме означене са *KeyGen()*, *Enc()*, *Dec()* [41]:

- Генерисање кључева: Боб из *KeyGen()* функције добија вредности броја $seed_A, b$ и s , при чему јавни кључ представља пар $pk = (seed_A, b)$ и додатно се генерише вредност $pkh = \mathcal{F}(pk)$, која представља излаз хеш функције за јавни кључ. Након овога се генерише насумични 256-битни број z који служи за генерисање кључа у случају да декрипција не успе. Бобов приватни кључ има облик (z, pkh, pk, s) .
- Енкрипција јавним кључем: Алиса генерише насумични 256-битни број m након чега из хеш функције добија вредности $(r, \hat{K}) = \mathcal{G}(\mathcal{F}(pk), m)$, које користи да би затим конструисала енкриптовану поруку $c = Enc(pk, m, r)$ и тајни кључ $K = \mathcal{H}(\mathcal{H}(c), \hat{K})$.
- Декрипција приватним кључем: Боб након што прими енкриптовани кључ од Алисе рачуна вредност оригиналне поруке $m' = Dec(s, c)$, након чега на основу хеш вредности од свог јавног кључа и добијене поруке рачуна $(r, \hat{K}') = \mathcal{G}(\mathcal{F}(pk), m')$, након чега Боб енкриптује добијену поруку и рачуна $c' = Enc(pk, m', r')$. На крају Боб

поређи енкриптовану поруку c' са примљеном енкриптованом поруком од Алисе и уколико нису једнаке кључ рачуна као $K = \mathcal{H}(\mathcal{H}(c), z)$, чиме размена кључева није успела. У супротном генерише се исти тајни кључ као Алисин $K = \mathcal{H}(\mathcal{H}(c), \hat{K}')$.

Као и код *Kyber* алгоритма, будући да се ради заокруживање, постоји врло мала вероватноћа да се правилно енкриптована порука не може дешифровати успешно.

У оквиру табеле 6.4 приказани су предложени сетови параметара за стандардизацију, као и вероватноћа да ће доћи до грешке (у табели дата као δ).

Назив сета параметара	l	n	q	p	T	μ	δ	Ниво сигурности
<i>LightSaber</i>	2	256	2^{13}	2^{10}	2^3	10	2^{-120}	1
<i>Saber</i>	3	256	2^{13}	2^{10}	2^4	8	2^{-136}	3
<i>FireSaber</i>	4	256	2^{13}	2^{10}	2^6	6	2^{-165}	5

Табела 6.4 – Параметри *Saber* алгоритма [39]

7. Апликација за тестирање перформанси асиметричних алгоритама

У оквиру овог поглавља биће приказана реализована апликација за тестирање и поређење перформанси различитих класичних и пост-квантних асиметричних алгоритама. Биће изложени опис функционалности, имплементациони детаљи, подржани алгоритми, инструкције за превођење и покретање апликације, приказ структуре апликације, као и приказани резултати рада апликације.

7.1. Опис функционалности

Апликација је имплементирана као конзолна апликација која се извршава у оквиру једне нити и која извршава низ тестова за подржане класичне и пост-квантне асиметричне алгоритме, при чему се у оквиру сваког теста симулира функционалност размене кључева и мере перформансе алгоритама. Апликација мери стандардне девијације времена и просечна времена потребна за генерисање кључева, рачунање дељене тајне и времена енкрипције и декрипције (за оне алгоритме који подржавају енкрипцију јавним кључем) у више (кориснички задатих) итерација. Времена се мере у милисекундама користећи имплементиране тајмере. Поред временских параметара алгоритама, такође се приказују и величине приватних и јавних кључева, величине генерисаног енкриптованог текста (за алгоритме који подржавају енкрипцију јавним кључем) и величине дељене тајне вредности. Додатно, исписује се и модел процесора на коме се апликација извршава, имајући у виду да од брзине процесора зависе сва времена која се добијају као резултати.

Излаз апликације се приказује на стандардном излазу, а истовремено постоји и опција да се излаз апликације сачува у оквиру текстуалног фајла, при чему уколико већ постоји такав фајл брише се његов садржај.

Процес симулације размене кључева се одвија на два начина, у зависности од тога да ли тренутни асиметрични алгоритам има само функционалност размене кључева или може да се користи и за енкрипцију јавним кључем.

У случају алгоритама за размену кључева (анонимни и *Ephemeral Diffie-Hellman* и *Elliptic Curve Diffie-Hellman*), обе стране генеришу своје јавне и приватне кључеве (при чему се мери брзина генерисања кључева од само једне стране), затим се обави размена јавних кључева и генерисање дељене тајне, и тајног *AES* 256-битног кључа уз помоћ *SHA-256* хеш функције (при чему је коришћење хеш функције урачунато у време генерисања дељене тајне).

У случају алгоритама за енкрипцију јавним кључем (сви остали подржани алгоритми), само једна страна генерише свој пар кључева и дистрибуира свој јавни кључ. Затим се генерише насумично тајни *AES* 256-битни кључ који се енкриптује са јавним кључем, при чему се мери време енкрипције одговарајућим тајмером. Друга страна дешифрује добијени енкриптовани кључ, при чему се мери време потребно за дешифрину. Време генерисања дељене тајне се у овом случају рачуна као просечна вредност времена енкрипције и дешифрине, ради поређења са алгоритмима размене кључева.

У оба случаја се такође и врши провера којом се верификује да су два добијена кључа једнака, која није укључена у процес мерења времена, чиме се верификује успешност размене кључева. Уколико се догоди грешка апликација исписује поруку грешке и престаје са радом.

7.2. Имплементациони детаљи

Апликација је реализована користећи C++ програмски језик у оквиру окружења *Visual Studio 2022* [46], при чему су у оквиру имплементације коришћене библиотеке отвореног кода *Crypto++* (верзија 8.60) [44] и *liboqs* (верзија 0.7.2) [45], заједно са *liboqs* екстензијама за C++ (верзија 0.7.1). Апликација је искључиво намењена за 64-битну архитектуру процесора (x64/x86_64), будући да библиотека *liboqs* тренутно подржава само тај инструкцијски сет.

Crypto++ библиотека је коришћена за имплементацију тестирања класичних асиметричних алгоритама, док је *liboqs* библиотека коришћена за имплементацију тестирања пост-квантних алгоритама. Додатно је потребно нагласити да су у оквиру анонимног и *Ephemeral Diffie-Hellman*, као и *ElGamal* алгорита коришћени предложени параметри дати у оквиру стандарда [43] [51].

7.3. Подржани алгоритми

У наставку је дата листа подржаних алгоритама:

- *Anonymous Diffie-Hellman* (подржане величине кључева: 1024, 2048, 4096, 8192 бита)
- *Ephemeral Diffie-Hellman* (подржане величине кључева: 1024, 2048, 4096, 8192 бита)
- *Elliptic Curve Diffie-Hellman* са кривом *Curve25519*
- *RSA* са *OAEP padding* шемом
- *Elliptic Curve Integrated Encryption Scheme* са кривом *secp256r1*
- *ElGamal* (подржане величине кључева: 1024, 2048, 4096, 8192 бита)
- *Classic-McEliece-348864*
- *Classic-McEliece-348864f*
- *Classic-McEliece-460896*
- *Classic-McEliece-460896f*
- *Classic-McEliece-6688128*
- *Classic-McEliece-6688128f*
- *Classic-McEliece-6960119*
- *Classic-McEliece-6960119f*
- *Classic-McEliece-8192128*
- *Classic-McEliece-8192128f*
- *Kyber512*
- *Kyber768*
- *Kyber1024*
- *Kyber512-90s*
- *Kyber768-90s*
- *Kyber1024-90s*
- *NTRU-HPS-2048-509*
- *NTRU-HPS-2048-677*
- *NTRU-HPS-4096-821*
- *NTRU-HPS-4096-1229*
- *NTRU-HRSS-701*
- *NTRU-HRSS-1373*
- *LightSaber-KEM*
- *Saber-KEM*
- *FireSaber-KEM*

7.4. Структура апликације

Организација фајлова по директоријумима је приказана у наставку:

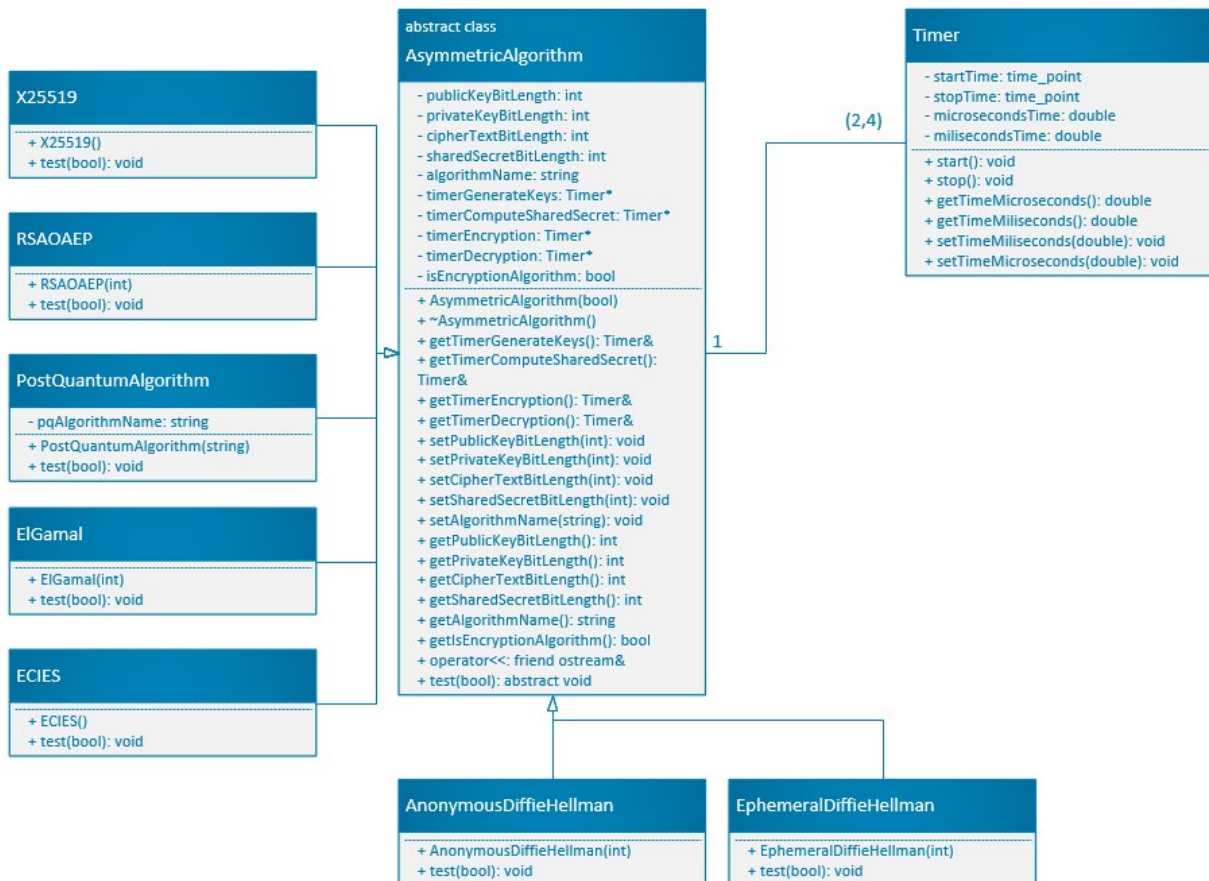
- апликација – главни директоријум апликације
 - cryptopp860 – директоријум *Crypto++* библиотеке
 - изворни код библиотеке - .cpp и .h фајлови
 - x64\Output\Release\cryptlib.lib – преведена статичка библиотека *Crypto++* за x64 архитектуру
 - liboqs – директоријум *liboqs* библиотеке
 - docs – директоријум који садржи документацију за библиотеку (може се приказати отварањем фајла html\index.html)
 - include – директоријум који садржи фолдер oqs у коме се налазе сви потребни *header* фајлови
 - lib – директоријум који садржи фајл oqs x64.lib – преведену статичку библиотеку *liboqs* за x64 архитектуру
 - TestingPerformanceOfAsymmetricAlgorithms – директоријум који садржи пројекат апликације и изворни код
 - TestingPerformanceOfAsymmetricAlgorithms.sln – пројектни фајл апликације
 - x64 – директоријум у коме се налази преведена апликација
 - TestingPerformanceOfAsymmetricAlgorithms – директоријум у коме се налази изворни код

Листа фајлова изворног кода дата је у наставку:

- Timer.h – *header* фајл који садржи декларацију класе тајмера
- Timer.cpp – *source* фајл који садржи имплементацију класе тајмера
- AsymmetricAlgorithm.h – *header* фајл који садржи декларацију апстрактне класе асиметричног алгорита
- AsymmetricAlgorithm.cpp – *source* фајл који садржи имплементацију метода апстрактне класе асиметричног алгорита
- AnonymousDiffieHellman.h – *header* фајл који садржи декларацију класе за тестирање анонимног *Diffie-Hellman* алгорита
- AnonymousDiffieHellman.cpp - *source* фајл који садржи имплементацију класе за тестирање анонимног *Diffie-Hellman* алгорита
- EphemeralDiffieHellman.h – *header* фајл који садржи декларацију класе за тестирање *Ephemeral Diffie-Hellman* алгорита
- EphemeralDiffieHellman.cpp – *source* фајл који садржи имплементацију класе за тестирање *Ephemeral Diffie-Hellman* алгорита
- X25519.h – *header* фајл који садржи декларацију класе за тестирање *Elliptic Curve Diffie-Hellman* алгорита са кривом *Curve25519*
- X25519.cpp – *source* фајл који садржи имплементацију класе за тестирање *Elliptic Curve Diffie-Hellman* алгорита са кривом *Curve25519*
- RSAOAEP.h – *header* фајл који садржи декларацију класе за тестирање *RSA* алгорита са *OAEP padding* шемом
- RSAOAEP.cpp – *source* фајл који садржи имплементацију класе за тестирање *RSA* алгорита са *OAEP padding* шемом

- EG.h – *header* фајл који садржи декларацију класе за тестирање *ElGamal* алгоритма
- EG.cpp – *source* фајл који садржи имплементацију класе за тестирање *ElGamal* алгоритма
- ECIES.h – *header* фајл који садржи декларацију класе за тестирање *Elliptic Curve Integrated Encryption Scheme* алгоритма користећи криву *secp256r1*
- ECIES.cpp – *source* фајл који садржи имплементацију класе за тестирање *Elliptic Curve Integrated Encryption Scheme* алгоритма користећи криву *secp256r1*
- PostQuantumAlgorithm.h – *header* фајл који садржи декларацију класе за тестирање пост-квантних асиметричних алгоритама из *liboqs* библиотеке
- PostQuantumAlgorithm.cpp – *source* фајл који садржи имплементацију класе за тестирање пост-квантних асиметричних алгоритама из *liboqs* библиотеке
- TestingPerformanceOfAsymmetricAlgorithms.cpp – *source* фајл који представља главни фајл апликације у коме се налази *main* метода из које се покрећу тестови
- common.h - *header* фајл из *liboqs* екстензије за C++
- oqs_cpp.h - *header* фајл из *liboqs* екстензије за C++
- rand.h - *header* фајл из *liboqs* екстензије за C++

Дијаграм класа у оквиру апликације је приказан на слици 7.1.



Слика 7.1 – Дијаграм класа апликације

7.5. Инструкције за превођење и покретање апликације

Прво је потребно отворити пројекат отварањем фајла апликација\TestingPerformanceOfAsymmetricAlgorithms\TestingPerformanceOfAsymmetricAlgorithms.sln у оквиру *Visual Studio 2022* окружења (и старије верзије могу да се користе) коришћењем опције *File → Open → Project/Solution* и бирањем поменутог фајла.

Након овога потребно је поставити подешавања за библиотеке да би се апликација успешно превела, при чему је потребно ући у подешавања пројекта коришћењем опције *Solution Explorer → TestingPerformanceOfAsymmetricAlgorithms*, и затим кликнути десни клик и изабрати *Properties*:

- у *Configuration Properties → C/C++ → General* кликнути на *Additional Include Directories* и ту додати путање до директоријума апликација\liboqs\include и апликација\cryptopp860
- у *Configuration Properties → Linker → General* кликнути на *Additional Library Directories* и ту додати путање до директоријума апликација\liboqs\lib и апликација\cryptopp860\x64\Output\Release.

Након овога је могуће покренути процес превођења апликације користећи опцију *Build → Build Solution*, при чему ће се апликација након завршетка процеса превођења налазити у директоријуму апликација\TestingPerformanceOfAsymmetricAlgorithms\x64.

У оквиру апликације (*source* фајла TestingPerformanceOfAsymmetricAlgorithms.cpp) се такође могу променити следећа подешавања, након чега је потребно поново превести апликацију:

- број итерација у којима се извршава сваки алгоритам (чиме се побољшава прецизност рачунања времена), променом вредности макроя NUMBER_OF_ITERATIONS на 19. линији који је подразумевано подешен на 100.
- да ли да се резултати рада апликације чувају у фајлу, променом вредности макроя SAVE_RESULTS на 20. линији који је подразумевано подешен на *true*
- име фајла у коме се чувају резултати рада апликације, променом вредности макроя RESULTS_FILE на 21. линији који је подразумевано подешен на *string "results.txt"*.

Након што се покрене преведена апликација, на стандардном излазу биће приказан сваки алгоритам након што је завршено његово тестирање, заједно са резултатима који иду уз њега. Након завршетка рада корисник добија поруку да може да притисне било који тастер, чиме се затвара апликација.

7.6. Резултати рада апликације

У наставку су приказани резултати рада апликације који се добијају на процесору *Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz* за 100 итерација.

У оквиру табеле 7.1 дате су величине јавних и приватних кључева, величина дељене генерисане тајне вредности (која се касније користи као тајни кључ или се претвара у тајни кључ одговарајуће величине (256 бита) користећи хеш функцију *SHA256*) и нивое сигурности (који се могу применити само на пост-квантне алгоритме и њихове сетове параметара). Табела је сортирана растуће према величини јавног кључа, будући да та величина представља врло битну информацију зато што корисници морају да размене јавне кључеве пре него што започну комуникацију. Класични алгоритми у оквиру табеле имају ознаку \times за ниво сигурности будући да они не припадају ниједном сигурносном нивоу који је дефинисао *NIST*, јер нису резистивни на нападе квантних рачунара. Величина генерисане тајне вредности износи 32 бајта (256 бита) за алгоритме који подржавају енкрипцију јавним кључем и ова тајна вредност након декрипције може директно да се користи као тајни кључ за симетричне алгоритме, док је код *Diffie-Hellman* алгоритма тајна вредност већа (будући да се генерише као број из опсега чија величина зависи од параметара алгоритма – величине кључева), па је потребно применити хеш функцију (*SHA256*) да би се тајна вредност користила као тајни кључ у оквиру симетричног алгоритма.

Асиметрични алгоритам	Величина јавног кључа (B) ↑	Величина приватног кључа (B)	Величина генерисане тајне вредности (B)	Ниво сигурности
<i>ECIES with secp256r1</i>	32	32	32	\times
<i>ECDH with curve25519</i>	32	32	32	\times
<i>Anonymous DH (1024 b)</i>	128	20	128	\times
<i>Ephemeral DH (1024 b)</i>	128	20	256	\times
<i>RSA with OAEP (1024 b)</i>	128	128	32	\times
<i>ElGamal (1024 b)</i>	128	20	32	\times
<i>Anonymous DH (2048 b)</i>	256	32	256	\times
<i>Ephemeral DH (2048 b)</i>	256	32	512	\times
<i>RSA with OAEP (2048 b)</i>	256	256	32	\times
<i>ElGamal (2048 b)</i>	256	32	32	\times
<i>Anonymous DH (4096 b)</i>	512	512	512	\times
<i>Ephemeral DH (4096 b)</i>	512	512	1024	\times
<i>ElGamal (4096 b)</i>	512	512	32	\times
<i>RSA with OAEP (4096 b)</i>	512	512	32	\times
<i>LightSaber-KEM</i>	672	1568	32	1
<i>NTRU-HPS-2048-509</i>	699	935	32	1
<i>Kyber512</i>	800	1632	32	1
<i>Kyber512-90s</i>	800	1632	32	1
<i>NTRU-HPS-2048-677</i>	930	1234	32	3
<i>Saber-KEM</i>	992	2304	32	3
<i>RSA with OAEP (8192 b)</i>	1024	1024	32	\times

<i>Anonymous DH (8192 b)</i>	1024	1024	1024	×
<i>Ephemeral DH (8192 b)</i>	1024	1024	2048	×
<i>ElGamal (8192 b)</i>	1024	1024	32	×
<i>NTRU-HRSS-701</i>	1138	1450	32	3
<i>Kyber768</i>	1184	2400	32	3
<i>Kyber768-90s</i>	1184	2400	32	3
<i>NTRU-HPS-4096-821</i>	1230	1590	32	5
<i>FireSaber-KEM</i>	1312	3040	32	5
<i>Kyber1024</i>	1568	3168	32	5
<i>Kyber1024-90s</i>	1568	3168	32	5
<i>NTRU-HPS-4096-1229</i>	1842	2366	32	5
<i>NTRU-HRSS-1373</i>	2401	2983	32	5
<i>Classic-McEliece-348864</i>	261120	6452	32	1
<i>Classic-McEliece-348864f</i>	261120	6452	32	1
<i>Classic-McEliece-460896</i>	524160	13568	32	3
<i>Classic-McEliece-460896f</i>	524160	13568	32	3
<i>Classic-McEliece-6688128</i>	1044992	13892	32	5
<i>Classic-McEliece-6688128f</i>	1044992	13892	32	5
<i>Classic-McEliece-6960119</i>	1047319	13908	32	5
<i>Classic-McEliece-6960119f</i>	1047319	13908	32	5
<i>Classic-McEliece-8192128</i>	1357824	14080	32	5
<i>Classic-McEliece-8192128f</i>	1357824	14080	32	5

Табела 7.1 – Величине кључева анализираних алгоритама

У оквиру табеле 7.2 приказане су величине енкриптоване поруке, тј. енкриптованог тајног кључа, за алгоритме који подржавају енкрипцију јавним кључем.

Асиметрични алгоритам	Величина енкриптоване поруке (B) ↑	Ниво сигурности
<i>ElGamal (1024 b)</i>	32	×
<i>ElGamal (2048 b)</i>	64	×
<i>ECIES with secp256r1</i>	117	×
<i>ElGamal (4096 b)</i>	128	×
<i>RSA with OAEP (1024 b)</i>	128	×
<i>Classic-McEliece-348864</i>	128	1
<i>Classic-McEliece-348864f</i>	128	1
<i>Classic-McEliece-460896</i>	188	3
<i>Classic-McEliece-460896f</i>	188	3
<i>Classic-McEliece-6960119</i>	226	5
<i>Classic-McEliece-6960119f</i>	226	5
<i>Classic-McEliece-6688128</i>	240	5
<i>Classic-McEliece-6688128f</i>	240	5
<i>Classic-McEliece-8192128</i>	240	5
<i>Classic-McEliece-8192128f</i>	240	5
<i>ElGamal (8192 b)</i>	256	×

<i>RSA with OAEP (2048 b)</i>	256	×
<i>RSA with OAEP (4096 b)</i>	512	×
<i>NTRU-HPS-2048-509</i>	699	1
<i>LightSaber-KEM</i>	736	1
<i>Kyber512</i>	768	1
<i>Kyber512-90s</i>	768	1
<i>NTRU-HPS-2048-677</i>	930	3
<i>RSA with OAEP (8192 b)</i>	1024	×
<i>Kyber768</i>	1088	3
<i>Kyber768-90s</i>	1088	3
<i>Saber-KEM</i>	1088	3
<i>NTRU-HRSS-701</i>	1138	3
<i>NTRU-HPS-4096-821</i>	1230	5
<i>FireSaber-KEM</i>	1472	5
<i>Kyber1024</i>	1568	5
<i>Kyber1024-90s</i>	1568	5
<i>NTRU-HPS-4096-1229</i>	1842	5
<i>NTRU-HRSS-1373</i>	2401	5

Табела 7.2 – Величине енкриптоване поруке

У оквиру табела 7.3, 7.4, 7.5 и 7.6 дат је приказ времена и стандардних девијација потребних за генерацију кључева и за рачунање дељеног тајног кључа (табела је сортирана по просечном времену потребном за рачунање тајног кључа) за стандардне алгоритме као и пост-квантне алгоритме са различитим нивоима сигурности, при чему ово време код алгоритма који врше енкрипцију јавним кључем представља просечно време између енкрипције и декрипције, приказано формулом $t_{shared_secret_time} = \frac{t_{enc} + t_{dec}}{2}$ и представља параметар по коме се могу поредити перформансе алгоритма за размену кључева и алгоритма који врше енкрипцију јавним кључем.

Асиметрични алгоритам	Просечно време за генерисање кључа (ms)	Стандардна девијација времена за генерисање кључа (ms)	Просечно време за рачунање дељеног тајног кључа (ms) ↑	Стандардна девијација времена за рачунање дељеног тајног кључа (ms)
<i>ECDH with curve25519</i>	0.3009	0.0399	0.0856	0.0124
<i>Anonymous DH (1024 b)</i>	0.7643	0.6049	0.2061	0.0526
<i>RSA with OAEP (1024 b)</i>	13.5394	5.3056	0.2487	0.0399
<i>ElGamal (1024 b)</i>	0.6064	0.0530	0.2782	0.0201
<i>Ephemeral DH (1024 b)</i>	0.7903	0.2546	0.3827	0.0355
<i>RSA with OAEP (2048 b)</i>	79.1534	42.0699	1.0038	0.0735
<i>Anonymous DH (2048 b)</i>	1.5529	0.0993	1.0368	0.1121
<i>ElGamal (2048 b)</i>	1.6383	0.1115	1.4412	0.0780

<i>ECIES with secp256r1</i>	1.1686	0.1566	1.4875	0.1648
<i>Ephemeral DH (2048 b)</i>	2.3432	0.2327	2.0896	0.2020
<i>Anonymous DH (4096 b)</i>	4.2910	0.7419	5.1093	0.9641
<i>RSA with OAEP (4096 b)</i>	950.0690	621.2480	6.4776	0.4752
<i>Ephemeral DH (4096 b)</i>	6.9041	0.3352	9.1653	0.5763
<i>Anonymous DH (8192 b)</i>	15.3013	1.5532	17.9367	2.8746
<i>Ephemeral DH (8192 b)</i>	27.9989	2.1759	34.8259	2.8685
<i>RSA with OAEP (8192 b)</i>	10770.6000	6975.1800	38.5006	1.1402
<i>ElGamal (4096 b)</i>	36.6614	1.0724	55.2167	1.5739
<i>ElGamal (8192 b)</i>	230.2630	4.2878	346.6150	3.5147

Табела 7.3 – Времена генерисања кључа и рачунања дељеног тајног кључа – стандардни алгоритми

Асиметрични алгоритам	Просечно време за генерисање кључа (ms)	Стандардна девијација времена за генерисање кључа (ms)	Просечно време за рачунање дељеног тајног кључа (ms) ↑	Стандардна девијација времена за рачунање дељеног тајног кључа (ms)
<i>Kyber512</i>	0.2888	0.0351	0.1354	0.0172
<i>NTRU-HPS-2048-509</i>	4.2248	0.1981	0.1742	0.0156
<i>Kyber512-90s</i>	0.3256	0.0287	0.1758	0.0144
<i>LightSaber-KEM</i>	0.4208	0.0351	0.1767	0.0184
<i>Classic-McEliece-348864</i>	174.6350	36.8920	0.3921	0.0963
<i>Classic-McEliece-348864f</i>	144.0390	4.1179	0.4168	0.1083

Табела 7.4 – Времена генерисања кључа и рачунања дељеног тајног кључа – 1. ниво сигурности

Асиметрични алгоритам	Просечно време за генерисање кључа (ms)	Стандардна девијација времена за генерисање кључа (ms)	Просечно време за рачунање дељеног тајног кључа (ms) ↑	Стандардна девијација времена за рачунање дељеног тајног кључа (ms)
<i>Kyber768</i>	0.3217	0.0249	0.1782	0.0154
<i>NTRU-HRSS-701</i>	7.7509	0.4082	0.2531	0.0271
<i>Kyber768-90s</i>	0.4137	0.0571	0.2599	0.0295
<i>NTRU-HPS-2048-677</i>	7.3069	0.3007	0.2600	0.0276
<i>Saber-KEM</i>	0.4971	0.0238	0.2748	0.0165
<i>Classic-McEliece-460896f</i>	424.9040	7.8727	0.7325	0.1431
<i>Classic-McEliece-460896</i>	547.6470	130.2900	0.7616	0.1865

Табела 7.5 – Времена генерисања кључа и рачунања дељеног тајног кључа – 3. ниво сигурности

Асиметрични алгоритам	Просечно време за генерисање кључа (ms)	Стандардна девијација времена за генерисање кључа (ms)	Просечно време за рачунање дељеног тајног кључа (ms) ↑	Стандардна девијација времена за рачунање дељеног тајног кључа (ms)
<i>Kyber1024</i>	0.3717	0.0280	0.2320	0.0196
<i>NTRU-HPS-4096-821</i>	10.5686	0.3805	0.3296	0.0290
<i>Kyber1024-90s</i>	0.5015	0.0552	0.3602	0.0321
<i>FireSaber-KEM</i>	0.6105	0.0357	0.4084	0.0310
<i>NTRU-HPS-4096-1229</i>	23.0654	0.5156	0.6183	0.0338
<i>NTRU-HRSS-1373</i>	29.4697	0.5438	0.8631	0.1250
<i>Classic-McEliece-6960119f</i>	549.7360	13.4158	1.0128	0.1348
<i>Classic-McEliece-6960119</i>	790.1480	281.8520	1.0147	0.1937
<i>Classic-McEliece-6688128</i>	885.4360	368.7460	1.0172	0.1935
<i>Classic-McEliece-6688128f</i>	576.9000	5.5432	1.0417	0.1824
<i>Classic-McEliece-8192128f</i>	601.9900	4.9415	1.1089	0.1555
<i>Classic-McEliece-8192128</i>	1189.5500	422.4230	1.1104	0.1554

Табела 7.6 – Времена генерисања кључа и рачунања дељеног тајног кључа – 5. ниво сигурности

У оквиру табела 7.7, 7.8, 7.9 и 7.10 дат је приказ просечних времена енкрипције (параметар коришћен за сортирање табеле) и декрипције као и стандардне девијације ових времена за алгоритме који врше енкрипцију јавним кључем, за стандардне алгоритме и различите пост-квантне алгоритме са различитим сетовима параметара, груписаним по нивоу сигурности.

Асиметрични алгоритам	Просечно време енкрипције (ms) ↑	Стандардна девијација времена енкрипције (ms)	Просечно време декрипције (ms)	Стандардна девијација времена декрипције (ms)
<i>RSA with OAEP (1024 b)</i>	0.0473	0.0158	0.4396	0.0660
<i>RSA with OAEP (2048 b)</i>	0.0745	0.0122	1.9229	0.1446
<i>RSA with OAEP (4096 b)</i>	0.1513	0.0318	12.7900	0.9346
<i>ElGamal (1024 b)</i>	0.2788	0.0234	0.2763	0.0239
<i>RSA with OAEP (8192 b)</i>	0.3411	0.0311	76.6476	2.2746
<i>ElGamal (2048 b)</i>	1.5736	0.1061	1.3066	0.1087

<i>ECIES with secp256r1</i>	1.7271	0.2335	1.2448	0.1387
<i>ElGamal (4096 b)</i>	72.0066	2.4714	38.4207	1.3003
<i>ElGamal (8192 b)</i>	456.4500	4.6047	236.7730	5.1509

Табела 7.7 – Времена енкрипције и декрипције – стандардни алгоритми

Асиметрични алгоритам	Просечно време енкрипције (ms) ↑	Стандардна девијација времена енкрипције (ms)	Просечно време декрипције (ms)	Стандардна девијација времена декрипције (ms)
<i>Kyber512</i>	0.1862	0.0238	0.0832	0.0126
<i>NTRU-HPS-2048-509</i>	0.1961	0.0188	0.1493	0.0155
<i>LightSaber-KEM</i>	0.2219	0.0228	0.1294	0.0162
<i>Kyber512-90s</i>	0.2272	0.0181	0.1226	0.0122
<i>Classic-McEliece-348864</i>	0.5091	0.1715	0.2705	0.0365
<i>Classic-McEliece-348864f</i>	0.5573	0.2040	0.2715	0.0374

Табела 7.8 – Времена енкрипције и декрипције – 1. ниво сигурности

Асиметрични алгоритам	Просечно време енкрипције (ms) ↑	Стандардна девијација времена енкрипције (ms)	Просечно време декрипције (ms)	Стандардна девијација времена декрипције (ms)
<i>NTRU-HRSS-701</i>	0.2256	0.0277	0.2770	0.0334
<i>Kyber768</i>	0.2271	0.0223	0.1277	0.0119
<i>NTRU-HPS-2048-677</i>	0.2495	0.0232	0.2673	0.0344
<i>Kyber768-90s</i>	0.3108	0.0360	0.2071	0.0296
<i>Saber-KEM</i>	0.3120	0.0218	0.2357	0.0197
<i>Classic-McEliece-460896f</i>	0.8225	0.2676	0.6372	0.0662
<i>Classic-McEliece-460896</i>	0.8911	0.3509	0.6270	0.0563

Табела 7.9 – Времена енкрипције и декрипције – 3. ниво сигурности

Асиметрични алгоритам	Просечно време енкрипције (ms) ↑	Стандардна девијација времена енкрипције (ms)	Просечно време декрипције (ms)	Стандардна девијација времена декрипције (ms)
<i>Kyber1024</i>	0.2801	0.0248	0.1821	0.0213
<i>NTRU-HPS-4096-821</i>	0.2952	0.0296	0.3603	0.0344
<i>Kyber1024-90s</i>	0.4101	0.0344	0.3082	0.0388
<i>FireSaber-KEM</i>	0.4430	0.0445	0.3718	0.0316
<i>NTRU-HPS-4096-1229</i>	0.4551	0.0171	0.7774	0.0607
<i>NTRU-HRSS-1373</i>	0.5279	0.0385	1.1942	0.2467
<i>Classic-McEliece-6688128</i>	1.3254	0.3797	0.7037	0.0494
<i>Classic-McEliece-6960119f</i>	1.3280	0.2511	0.6919	0.0869

<i>Classic-McEliece-6960119</i>	1.3381	0.3462	0.6855	0.0769
<i>Classic-McEliece-6688128f</i>	1.3701	0.3636	0.7079	0.0542
<i>Classic-McEliece-8192128f</i>	1.5004	0.2907	0.7111	0.0759
<i>Classic-McEliece-8192128</i>	1.5075	0.2724	0.7075	0.0845

Табела 7.10 – Времена енкрипције и декрипције – 5. ниво сигурности

Може се приметити да су величине приватних и јавних кључева коришћених у оквиру пост-квантних алгоритама прилично веће од величина кључева класичних алгоритама, нарочито од алгоритама који су засновани на елиптичним кривама. Ово не би требало да представља проблем, осим у случају *Classic McEliece* где у појединим применама на системима са малом количином меморије може доћи до прекорачења.

Време потребно за генерисање пара приватног и јавног кључа се такође прилично разликује – од пост-квантних алгоритама најбоље перформансе имају *Kyber* и *Saber*, док генерално најгоре перформансе имају *Classic McEliece* и *RSA* за најсигурније сетове параметара. Овде је такође битно скренути пажњу на имплементациони детаљ – без усвојених параметара из стандарда време генерације кључева и параметара код *DH*, *ECDH* и *ElGamal* алгоритама би трајало много дуже (за неколико редова величине).

Као последица приказаних података, може се уочити да *Kyber* има најбоље перформансе од приказаних пост-квантних алгоритама заснованих на решеткама, које блиско прати *Saber*. Због овога али и због већег броја радова и анализа *LWE* проблема који представља основу сигурности *Kyber* алгорита у односу на *LWR* проблем који се користи у оквиру *Saber* алгорита, *NIST* је предложио стандардизацију *Kyber* алгорита после треће рунде, док *NTRU* и *Saber* неће бити укључени у процес стандардизације. [47]

Classic McEliece алгоритам представља посебан случај, где се користе много веће величине јавних кључева (за одређене параметре чак преко 1 MB), док алгоритам има пристојне брзине енкрипције и декрипције и такође најмању величину енкриптоване поруке од свих осталих разматраних пост-квантних алгоритама. Будући да је *NIST* прилично сигуран у безбедност *Classic McEliece* алгорита, а да и даље није јасно да ли је неки од алтернативних алгоритама бољи за случајеве кад је потребна мала величина енкриптоване поруке и кад се парови кључева не генеришу и размењују превише често, *Classic McEliece* ће бити додатно анализиран у оквиру четврте рунде, заједно са алтернативним алгоритмима за размену кључева. [47]

8. Закључак

Развој квантних рачунара и алгоритама који их користе наставиће да представља један од значајнијих сегмената развоја у оквиру информатике, будући да се могу искористити за решавање проблема који не могу на једноставан и ефикасан начин да се реше применом класичних рачунара.

Наравно, будући да квантни рачунари представљају претњу тренутно стандарним асиметричним алгоритмима потребна је нова класа пост-квантних алгоритама који су отпорни на нападе и класичних и квантних алгоритама. Истраживање и развој пост-квантних алгоритама је од изузетног значаја за одржавање својстава тајности и интегритета података у оквиру рада интернета и бројних сервиса. Време потребно за истраживање, развој, стандардизацију и широко прихватање пост-квантних алгоритама мора да буде краће од времена развоја квантних рачунара да би се гарантовала поменута својства и спречили активни напади, при чему ће сигурност стандардизованих пост-квантних алгоритама и њихових сетова параметара морати такође да прође и тест времена. Нажалост пост-квантни алгоритми неће бити у могућности да спрече пасивне нападе – нападач који има способност прислушкивања саобраћаја и довољне капацитете да тај саобраћај сачува биће у могућности да дође до раније послатих шифрованих података уколико буде поседовао адекватни квантни рачунар.

Тема овог рада је анализа пост-квантних алгоритама, њихових перформанси и меморијских захтева, као и њихов однос према тренутно широко коришћеним и стандардним асиметричним алгоритмима, кроз имплементацију апликације која користи библиотеке отвореног кода. Један од видова унапређења могао би да представља додавање и кандидата из алтернативне категорије алгоритама за размену кључева, што се на основу тренутне имплементације може лако урадити као и налажење начина за тестирање величине меморије неопходне за сваку имплементацију алгоритма у зависности од платформе и опција за превођење.

Литература

- [1] Dan Boneh, Victor Shoup, *A Graduate Course in Applied Cryptography*, January 2020.
- [2] Whitfield Diffie, Martin E. Hellman, *New Directions in Cryptography*, **IEEE Transactions on Information Theory**, November 1976.
- [3] Claude E. Shannon, *Communication Theory of Secrecy Systems*, **Bell System Technical Journal**, October 1949.
- [4] Whitfield Diffie, Martin E. Hellman, *Multiuser cryptographic techniques*, June 1976.
- [5] William Stallings, *Cryptography and Network Security: Principles and Practice 5th Edition*, January 2010.
- [6] Andrej Bogdanov, Luca Trevisan, *Average-Case Complexity*, October 2006, Revised August 2021.
- [7] S.N.Ghosh, Deepak T Biradar, Ganesh C Shinde, Sarika D Bhojaned, Manojkumar R Shirapure, *Performance Analysis of AES, DES, RSA And AES-DES-RSA Hybrid Algorithm for Data Security*, **International Journal of Innovative and Emerging Research in Engineering Volume 2, Issue 5**, January 2015.
- [8] Phong Q. Nguyen, *Public-Key Cryptanalysis*, January 2009.
- [9] M. Bellare, A. Desai, D. Pointcheval, P. Rogaway, *Relations Among Notions of Security for Public-Key Encryption Schemes*, June 2001.
- [10] *The Heartbleed Bug*, <https://heartbleed.com> (приступљено августа 2022.)
- [11] Nigel P. Smart, *Cryptography Made Simple*, 2016.
- [12] Elaine Barker, John Kelsey, *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*, **NIST Special Publication 800-90A Revision 1**, June 2015.
- [13] Lov K. Grover, *A fast quantum mechanical algorithm for database search*, July 1996.
- [14] Gilles Brassard, Peter Hoyer, Alain Tapp, *Quantum Algorithm for the Collision Problem*, May 1996.
- [15] Elaine Barker, Quynh Dang, *Recommendation for Key Management*, **NIST Special Publication 800-57 Part 3 Revision 1**, January 2015.
- [16] *Cryptographic Mechanisms: Recommendations and Key Length*, **Federal Office for Information Security, Germany**, January 2022.
- [17] R. L. Rivest, A. Shamir, L. Adleman, *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, **Communications of the ACM**, February 1978.
- [18] *OpenSSL Wiki Diffie-Hellman*, https://wiki.openssl.org/index.php/Diffie_Hellman (приступљено августа 2022.)
- [19] Salim Ali Abbas, Amal Abdul Baqi Maryoosh, *Data Security for Cloud Computing based on Elliptic Curve Integrated Encryption Scheme (ECIES) and Modified Identity based Cryptography (MIBC)*, **International Journal of Applied Information Systems**, March 2016

- [20] Peter Shor, *Algorithms for quantum computation: discrete logarithms and factoring*, November 1994.
- [21] *Timeline of quantum computing and communication*, https://en.wikipedia.org/wiki/Timeline_of_quantum_computing_and_communication (приступљено августа 2022.)
- [22] *List of quantum processors*, https://en.wikipedia.org/wiki/List_of_quantum_processors (приступљено августа 2022.)
- [23] Zion Elani, *Qubit, Quantum Entanglement and all that: Quantum Computing Made Simple*, January 2020.
- [24] Matthew Hayward, *Quantum Computing and Shor's Algorithm*, February 2005.
- [25] Quentin Truong, *Introduction to Quantum Programming*, August 2015. <https://towardsdatascience.com/introduction-to-quantum-programming-a19aa0b923a9> (приступљено августа 2022.)
- [26] Samuel J. Lomonaco, *A lecture on Shor's quantum factoring algorithm*, October 2000.
- [27] Martin Roetteler, Michael Naehrig, Krysta M. Svore, and Kristin Laute, *Quantum Resource Estimates for Computing Elliptic Curve Discrete Logarithms*, October 2017.
- [28] Lieven M.K. Vandersypen, Matthias Steffen, Gregory Breyta, Costantino S. Yannoni, Mark H. Sherwood, Isaac L. Chuang, *Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance*, December 2001.
- [29] Enrique Martin-Lopez, Anthony Laing, Thomas Lawson, Roberto Alvarez, Xiao-Qi Zhou, Jeremy L. O'Brien, *Experimental realisation of Shor's quantum factoring algorithm using qubit recycling*, October 2012.
- [30] Mirko Amico, Zain H. Saleem, Muir Kumph, *An Experimental Study of Shor's Factoring Algorithm on IBM Q*, May 2019.
- [31] Michele Mosca, Marco Piani, *2021 Quantum Threat Timeline Report*, **Global Risk Institute**, January 2022.
- [32] Daniele Micciancio, Oded Regev, *Lattice-based Cryptography*, November 2008.
- [33] R. J. McEliece, *A Public-Key Cryptosystem Based On Algebraic Coding Theory*, February 1978.
- [34] Martin R. Albrecht, Daniel J. Bernstein et al., *Classic McEliece: conservative code-based cryptography*, October 2020.
- [35] Harshdeep Singh, *Code based Cryptography: Classic McEliece*, May 2020.
- [36] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, Damien Stehlé, *CRYSTALS-Kyber (version 3.02) – Submission to round 3 of the NIST post-quantum project*, August 2021.
- [37] Katharina Boudgoust, Corentin Jeudy, Adeline Roux-Langlois, Weiqiang Wen, *On the Hardness of Module Learning With Errors with Short Distributions*, April 2022.
- [38] Jeffrey Hoffstein, Jill Pipher, Joseph H. Silverman, *NTRU: A new high speed public key cryptosystem*, August 1996.

- [39] Cong Chen, Oussama Danba, Jeffrey Hoffstein et al., *NTRU: Algorithm Specifications And Supporting Documentation*, September 2020.
- [40] *NTRU*, <https://en.wikipedia.org/wiki/NTRU> (приступљено августа 2022.)
- [41] Andrea Basso, Jose Maria Bermudo Mera, Jan-Pieter D’Anvers, Angshuman Karmakar, Sujoy Sinha Roy, Michiel Van Beirendonck, Frederik Vercauteren, *SABER: Mod-LWR based KEM (Round 3 Submission)*, October 2020.
- [42] Joël Alwen, Stephan Krenn, Krzysztof Pietrzak, Daniel Wichs, *Learning with Rounding, Revisited: New Reduction, Properties and Application*, February 2013.
- [43] *RFC5114 - Additional Diffie-Hellman Groups for Use with IETF Standards*, January 2008. <https://datatracker.ietf.org/doc/html/rfc5114>
- [44] *Crypto++ Library*, <https://cryptopp.com/> (приступљено августа 2022.)
- [45] *Open Quantum Safe – liboqs*, <https://openquantumsafe.org/liboqs/> (приступљено августа 2022.)
- [46] *Microsoft Visual Studio 2022*, <https://visualstudio.microsoft.com/vs/> (приступљено августа 2022.)
- [47] Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, Thinh Dang et al., *Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process*, July 2022.
- [48] *Bloch sphere* https://en.wikipedia.org/wiki/Bloch_sphere (приступљено августа 2022.)
- [49] *Quantum logic gates* https://en.wikipedia.org/wiki/Quantum_logic_gate (приступљено августа 2022.)
- [50] *RFC3766 - Determining Strengths For Public Keys Used For Exchanging Symmetric Keys*, April 2004. <https://datatracker.ietf.org/doc/html/rfc3766>
- [51] *RFC3526 - More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)*, May 2003. <https://www.rfc-editor.org/rfc/rfc3526>

Списак скраћеница

<i>CIA trpujada</i>	<i>Confidentiality, Integrity, and Availability</i>
<i>SSL</i>	<i>Secure Sockets Layer</i>
<i>TLS</i>	<i>Transport Layer Security</i>
<i>PGP</i>	<i>Pretty Good Privacy</i>
<i>IPsec</i>	<i>IP security</i>
<i>S/MIME</i>	<i>Secure/Multipurpose Internet Mail Extensions</i>
<i>OTR</i>	<i>Off-The-Record Messaging</i>
<i>NIST</i>	<i>National Institute of Standards and Technology</i>
<i>RSA</i>	<i>Rivest–Shamir–Adleman</i>
<i>ECC</i>	<i>Elliptic-Curve Cryptography</i>
<i>AES</i>	<i>Advanced Encryption Standard</i>
<i>ECIES</i>	<i>Elliptic Curve Integrated Encryption Scheme</i>
<i>ECDSA</i>	<i>Elliptic Curve Digital Signature Algorithm</i>
<i>MITM</i>	<i>Man-In-The-Middle</i>
<i>OW-CPA</i>	<i>One Way - Chosen Plaintext Attack</i>
<i>OW-CCA</i>	<i>One Way - Chosen Ciphertext Attack</i>
<i>IND-CPA</i>	<i>INDistinguishability under Chosen Plaintext Attack</i>
<i>IND-CCA1</i>	<i>INDistinguishability under Chosen Ciphertext Attack</i>
<i>IND-CCA2</i>	<i>INDistinguishability under adaptive Chosen Ciphertext Attack</i>
<i>NM-CPA</i>	<i>Non Malleable - Chosen Plaintext Attack</i>
<i>NM-CCA1</i>	<i>Non Malleable - Chosen Ciphertext Attack</i>
<i>NM-CCA2</i>	<i>Non Malleable - adaptive Chosen Ciphertext Attack</i>
<i>SHA</i>	<i>Secure Hash Algorithm</i>
<i>CA</i>	<i>Certificate Authority</i>
<i>PFS</i>	<i>Perfect Forward Secrecy</i>
<i>gcd</i>	<i>Greatest Common Divisor</i>
<i>OAEP</i>	<i>Optimal Asymmetric Encryption Padding</i>
<i>MAC</i>	<i>Message Authentication Code</i>
<i>KDF</i>	<i>Key Derivation Function</i>
<i>BQP</i>	<i>Bounded-error Quantum Polynomial time</i>

<i>PQC</i>	<i>Post-Quantum Cryptography</i>
<i>KEM</i>	<i>Key Encapsulation Mechanism</i>
<i>CRYSTALS</i>	<i>Cryptographic Suite for Algebraic Lattices</i>
<i>PKE</i>	<i>Public Key Encryption</i>
<i>FO</i>	<i>Fujisaki-Okamoto трансформација</i>
<i>LWE</i>	<i>Learning With Errors</i>
<i>CVP</i>	<i>Closest vector problem</i>
<i>NTT</i>	<i>Number Theoretic Transform</i>
<i>DPKE</i>	<i>Deterministic Public Key Encryption scheme</i>
<i>LWR</i>	<i>Learning With Rounding</i>
<i>EG</i>	<i>El Gamal</i>
<i>X25519</i>	<i>Elliptic Curve Diffie-Hellman алгоритам са кривом Curve25519</i>
<i>ECDH</i>	<i>Elliptic Curve Diffie-Hellman</i>
<i>DH</i>	<i>Diffie-Hellman</i>
<i>pk</i>	<i>public key</i>
<i>sk</i>	<i>secret key</i>
<i>API</i>	<i>Application Programming Interface</i>

Списак слика

Слика 2.1 – Блок дијаграм комуникације користећи симетрични алгоритам за енкрипцију порука.....	5
Слика 3.1 - Блок дијаграм комуникације користећи асиметрични алгоритам за енкрипцију порука - тајност	10
Слика 3.2 - Блок дијаграм комуникације користећи асиметрични алгоритам за енкрипцију порука – аутентификација/интегритет	11
Слика 3.3 - Блок дијаграм комуникације користећи асиметрични алгоритам за енкрипцију порука – аутентификација/интегритет и тајност	12
Слика 4.1 – Пример елиптичне криве [1]	21
Слика 4.2 – ECIES шема за енкрипцију [19]	23
Слика 4.3 - ECIES шема за дешифрацију [19]	24
Слика 5.1 – Приказ кубита на Блоховој сфери [48]	26
Слика 5.2 – Најпознатија квантна логичка кола [49].....	27
Слика 5.3 – Резултати анкете [31]	29
Слика 7.1 – Дијаграм класа апликације	47

Списак табела

Табела 3.1 – Асиметрични алгоритми и њихове функционалности	7
Табела 3.2 – NIST сигурносни нивои за анализу пост-квантне криптографије.....	15
Табела 6.1 – Параметри Classic McEliece алгоритма [34]	34
Табела 6.2 – Параметри Kyber алгоритма [36]	37
Табела 6.3 – Параметри NTRU алгоритма [39]	40
Табела 6.4 – Параметри Saber алгоритма [39]	43
Табела 7.1 – Величине кључева анализираних алгоритама	50
Табела 7.2 – Величине енкриптоване поруке	51
Табела 7.3 – Времена генерисања кључа и рачунања дељеног тајног кључа – стандардни алгоритми.....	52
Табела 7.4 – Времена генерисања кључа и рачунања дељеног тајног кључа – 1. ниво сигурности	52
Табела 7.5 – Времена генерисања кључа и рачунања дељеног тајног кључа – 3. ниво сигурности	52
Табела 7.6 – Времена генерисања кључа и рачунања дељеног тајног кључа – 5. ниво сигурности	53
Табела 7.7 – Времена енкрипције и декрипције – стандардни алгоритми	54
Табела 7.8 – Времена енкрипције и декрипције – 1. ниво сигурности	54
Табела 7.9 – Времена енкрипције и декрипције – 3. ниво сигурности	54
Табела 7.10 – Времена енкрипције и декрипције – 5. ниво сигурности	55

А. Увод у математику решетки

Решетка (енг. *lattice*) \mathcal{L} се дефинише као скуп вектора свих целобројних комбинација n линеарно независних вектора који припадају \mathbb{R}^n , из скупа $\{b_1, b_2, \dots, b_n\}$, који представља основу решетки, тј. важи:

$$\mathcal{L}(b_1, b_2, \dots, b_n) = \left\{ \sum_{i=1}^n x_i \cdot b_i : x_i \in \mathbb{Z} \right\}.$$

Основа решетки се такође може приказати и у матричном облику, као матрица $B \in \mathbb{R}^{n \times n}$, у којој су вектори основе распоређени у колонама, при чему се у овом случају решетка дефинише у облику $\mathcal{L}(B) = \{B \cdot x : x \in \mathbb{Z}^n\}$, где x представља вектор димензије n , а $B \cdot x$ представља операцију мултипликације матрице са вектором. Битна особина која важи код решетки је да уколико имамо дефинисану унимодуларну матрицу U (квадратну целобројну матрицу чија је детерминанта једнака ± 1), основе B и $B \cdot U$ генеришу исте решетки, тј. важи $\mathcal{L}(B) = \mathcal{L}(B \cdot U)$. Ова особина означава да за једну решетку постоји више истих основа преко којих се може формирати, што представља изузетну особину за примену решетки у криптографији. [32]

Такође, будући да решетка представља структуру дискретног карактера, може се дефинисати и дужина најмањег ненулног елемента у оквиру ње као $\lambda_1(\mathcal{L}) = \min\{\|x\| : x \in \mathcal{L}, x \neq 0\}$. Такође дефинисаћемо и $\lambda_i(\mathcal{L})$ као најмању дужину r за коју је могуће формирати n -димензионалну лопту полупречника r у којој се налази i линеарно независних тачака решетки.

У криптографији се генерално користе три основна проблема која се јављају код решетки великих димензија: [11]

- проблем одређивања најкраћег вектора (енг. *Shortest Vector Problem, SVP*)
- проблем одређивања најближег вектора (енг. *Closest Vector Problem, CVP*)
- проблем декодирања у ограниченој дистанци (енг. *Bounded-Distance Decoding Problem, BDD*)

SVP представља један од најпознатијих проблема који се јављају код решетки, где је потребно пронаћи најкраћи вектор у оквиру решетки. Овај проблем долази у више варијација чије су дефиниције приказане у наставку [11]:

- основна варијанта *SVP*, где је потребно пронаћи вектор x у оквиру дате решетки \mathcal{L} да важи $\|x\| < \|y\|$, за све ненулта векторе $y \in \mathcal{L}$, тј. пронаћи вектор x такав да важи $x = \lambda_1(\mathcal{L})$
- апроксимативна варијанта *SVP*, где је потребно пронаћи вектор x у оквиру дате решетки \mathcal{L} и за дату константу γ за који важи $\|x\| \leq \gamma \cdot \lambda_1(\mathcal{L})$
- γ -јединствен *SVP*, где је потребно пронаћи вектор x у оквиру дате решетки \mathcal{L} и за дату константу γ за коју важи $\lambda_2(\mathcal{L}) > \gamma \cdot \lambda_1(\mathcal{L})$, тако да важи $\|x\| = \lambda_1(\mathcal{L})$.

CVP је други најважнији проблем код решетки, где је за дату основу решетки \mathcal{L} у матричном облику B и за задати вектор $x \in \mathbb{R}^n$ који не припада решетки потребно пронаћи вектор $y \in \mathcal{L}$, тако да важи $\|x - y\| \leq \|x - z\|$ за сваки вектор $z \in \mathcal{L}$. Такође и за овај проблем

постоји апроксимативна верзија где је је потребно пронаћи вектор $y \in \mathcal{L}$ за који важи $\|x - y\| \leq \gamma \cdot \|x - z\|$ за сваки вектор $z \in \mathcal{L}$, где γ представља константу. [11]

BDD представља проблем који је сличан проблему *CVP*, при чему је потребно за дату матрицу основе B решетке \mathcal{L} , за вектор $x \in \mathbb{R}^n$ и за дати број λ пронаћи вектор $y \in \mathcal{L}$ за који важи $\|x - y\| \leq \lambda \cdot \lambda_1(\mathcal{L})$. [11]