Міністерство освіти і науки України
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерної інженерії

СУСЛІН Віталій Вікторович

« Алгоритм шифрування даних із використанням модульного експоненціювання для симетричних криптосистем / Data encryption algorithm using modular exposure for symmetric cryptosystems»

Студент групи КІм – 21
СУСЛІН Віталій Вікторович

_____

Науковий керівник
к.т.н., Ю.М. Батько

_____

Тернопіль – 2020

РЕЗЮМЕ

Кваліфікаційна робота на тему «Алгоритм шифрування даних із використанням модульного експоненціювання для асиметричних криптосистем» зі спеціальності 123 «Комп’ютерна інженерія» освітнього ступеня «магістр» написана обсягом 89 сторінок і містить 24 ілюстрації, 2 додатка та 54 джерела за переліком посилань.

Метою роботи є розробка алгоритму шифрування із використанням модульного експоненціювання для симетричних криптосистем.

Методи досліджень. Для досягнення даної мети виконано наступний ряд завдань, досліджено інформацію та її форми; проаналізовано засоби шифрування та дешифрування даних; проаналізовано синхронні та асинхронні криптосистеми; досліджено основні математичні операції шифрування та дешифрування; здійснено пошук покращеного та спрощеного методу операційного обчислення.

Результати дослідження: алгоритм пришвидшеної дії за допомогою спрощення обчислювальної складності вдосконаливши обчислювальння процесу шифрування даних, а саме модулярне експоненціювання.

Результати роботи можуть бути використані в таких сферах як: військова, урядова, фінансова та у звичайних системах контролю персональними даними.

Орієнтовані напрямки розвитку дослідження: розроблення програмних систем на основі розробленого алгоритму із вдосконаленою формою обчислення та пов’язувати із застосуванням розпаралелень, що дозволяє здійснити обчислення в різних потоках та зменшити обчислювальну складність.

КЛЮЧОВІ СЛОВА: АЛГОРИТМ, ДАНІ, ЗАХИСТ, БЕЗПЕКА, ЕКСПОНЕНЦІЮВАННЯ, ШИФРУВАННЯ, ДЕШИФРУВАННЯ, ПРОГРАМНА СИСТЕМА

RESUME

Qualification work of: “Data encryption algorithm using modular exposure for symmetric cryptosystems” in the specialty 123 “Computes Engineering” is written in 89 pages and contains 24 illustrations, 2 appendices and 54 sources from the list of references.

The aim is to develop an data encryption algorithm using modular exposure for symmetric cryptosystems.

Research methods: To achieve aim of qualification work, the following tasks is done: information an its forms were studied; data encryption and decryption tools are analyzed; synchronous and asynchronous cryptosystems are analyzed; the basic mathematical operations of encryption and decryption are investigated; and improved and simplified methods of operation calculation was searched.

Research result: accelerated action algorithm by simplifying computational complexity by improving the calculation of the data encryption process, namely modular exposition.

The result can be used in areas such as military, government, finance and conventional personal data control system.

Indicative directions of research development: develop software systems on the basis of the developed algorithm with the improved form of calculation and to connect with application of parallels that allows to carry out calculation in various streams and to reduce computational complexity; development of software libraries based on the developed algorithm for its use in other systems as a utility.

KEY WORDS: ALGORITHM, DATA, PROTECTION, SECURITY, EXPOSURE, ENCRYPTION, DECRYPTION, SOFTWARE SYSTEM.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

SSL(Secure Sockets Layer) – рівень захищеності сокетів.

TCP/IP – набір протоколів мережі, що складається із між мережевих протоколів і протоколу керування передаванням даних.

DES (Data Encryption Standard) – стандарт шифрування даних.

ВСТУП

Актуальність теми. Актульним є розв’язання проблематики сучасних крипто алгоритмів у швидкодії обчислення, які практично є досить громіздкими в обчисленні, що є досить затратними у використанні ресурсів також є створенням програмних систем, що здатні реалізовувати швидкодійну обробку застосовуючи програмні рішення для вдосконалення обчислень.

Значна частина стратегії захисту даних виконує роль швидкого відновлення даних після втрати або пошкодження. Захист даних від посторонніх та забезпечення конфіденційністю даних є іншими важливими компонентами захисту даних. У випадку коли персональні дані стають відомі постороннім особам, тоді це може призвести до втрати грошей, репутації, а також штрафи і тому важливо суворо дотримуватися правил захисту персональних даних. На сьогодні криптосистеми відіграють одну із лючових ролей під час захисту інформаційних потоків від стороннього доступу. До найбільш поширених криптосистем симетричні та асиметричні. Симетричні є швикодійними, проте вони зустрічаються в змішанні із асиметричними, де основними операціями які скаладють оючислювадьну складність модулярне множення, модулярне експоненціювання.

Таким чином постає проблематика у дослідженні алгоритмів і методів шифрування даних для зменшування обчислювальної складності та підвищити швидкодію алгоритмів для спеціалізованого програмного і апаратнго забезпечення при виконанні арифметични операцій в асиметричних криптосистемах. Розробка програмних систем дозволяє здійснювати користувачу обчислення набагато швидше, через реалізацію алгоритмів запрограмованим способом та застосовувати їх для потребах, без затрати часу на реалізацію явних математичних обчилень.

Метою роботи є розробка алгоритму шифрування на основі модулярного множення для асиметричних криптосистем.

Для досягнення даної мети потрібно виконати наступний ряд завдань:

- дослідити інформацію та її форми;
- проаналізувати засоби шифрування та дешифрування даних;
- проаналізувати синхронні та асинхронні криптосистеми;
- розробити вдосконалений алгоритм;
- здійснити реалізацію програмної системи із розробленим алгоритмом та виділити особливості;
- провести тестування та порівняння із аналогами;
- зробити висновок.

Об’єкт дослідження – процеш шифрування даних

Предмет дослідження – алгоритми і алгоритми шифрування даних.

Наукова новизна одержаних результатів зазначається наступним чином:

- Дослідження алгоритмів та методів шифрування та дешифрування даних у симетричних та асиметричнихї криптосистемах
- Розроблено алгоритм пришвидшеної дії за допомогою спрощення обчислюваноюї складності вдосконаливши обчислювальни процес шифрування даних, а саме модулярне експоненціювання.

Практичною цінністю дослідження є :

- розроблена програмна система на основі розробленого алгоритму із вдосконаленою формою обчислення та із застосуванням бібілотеки розпаралелень, що дозволяє здійснити обчислення в різних потоках і вирішує часову затратність
- розроблена програмна система для розв’язання проблематики шифрування даних для користувачів із вирішенням часових затрат на обробку та можливістю користуватися рішенням внезалежностівід платформи. Реалізація здійснена на мові програмування Java.
- Розроблено бібліотеку вдосконаленого алгоритму шифрування даних на основі модулярного експоненціонування у асиметричних криптосистем Криптолайт із можливістю подальшого застосунку її у інших програмних системах як утиліту.

Публікації та апробаія до випускної кваліфікаційної роботи . Отримавши результати підчас наукових досліджень, здійснених у випускній кваліфікаційній роботі, підготовано тези[1,2] доповіді «Алгоритм шифрування даних на основі модулярного експоненціонування» обсягом 1 сторінка на III Науково-практичній конференії молодих вчених і студентів «Інтелектуальні комп’ютерні системи та мережі», а також «Програмна система шифрування даних на основі алгоритму криптолайт» обсягом 1 сторінка на III Науково-практичній конференції молодих вчених і студентів «Інтелектуальні комп’ютерні системи та мережі».

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Поняття інформації та її форми

Інформація – абстраговане судження, що має багато значення взалежності від контекстку. Спроби формального визначення інформації відмінні в різних сферах людської діяльності. На інтуїтивному рівні інформація означає зміст того, про що отримувач довідався. Засобами масової інформації поширюють її а інформаційні технології виконують її обробкою. Прогнозується, що людство прямує до ідї інформаційного суспільства. Ще на початку XXI століття інформація посідала дедалі важливіше місце в житті людини, як член суспільства. Термін «інформація» є байдужим до істинності вмісту. Основою такого суспільства якраз є споживання й виробництво не товарів і послуг, а інформації[3].

Інформація може бути відвертою брехнею, хибною або правдивою. Від даних інформація відрізняється доступністю отримувачу. Для прикладу, нерозшифрований рукопис якогось невідомого китайського вченого, можливо, містить якийсь зміст, а отже, потенційно може стати інформацією, але до розшифрування він — лише набір символів і картинок, дані, над якими можуть працювати дешифрувальники. На сьогодні інформація є товаром, тобто купується та продаватися як і будь-який інший товар. Розвідувальні спец-служби здійснюють збір інформації про наявного або потенційного супротивника. Цінність інформації визначається корисністю та здатністю її забезпечити суб'єкта необхідними умовами для досягнення ним поставленої мети.

У компютерному світі вся інформація, що надходить ззовні, переводиться на найпростішу для комп'ютера мову, мову машин их кодів, тобто у двійкову форму. Літер у цій формі немає, а всьоголиш цифри 1 та 0. Кожен блок із громіздких двійкових форм може означати число в звичайній для людини десятковій системі діапазоном від 0 до 255.

Здатність інформації об'єктивно відбивати процеси та явища, що відбуваються в навколишньому світі називаєть достовірністю. Такою вважається, як правило інформація, яка несе у собі безпомилкові та істинні дані. Під безпомилковістю слід зрозуміти дані які не мають, прихованих або випадкових помилок. Випадкові помилки в даних обумовлені, як правило, ненавмисним спотвореннями змісту людиною чи збоями технічних засобів при переробці даних в інформаційних системах. Тоді як під істинними слід розуміти дані, зміст яких неможливо оскаржити або заперечити.

Обробка інформації є одержанням одних інформаційних об'єктів від інших фнормаційних об'єктів шляхом виконання спеціальних алгоритмів. Обробка є однією із основних операцій, які виконують над інформацією та головним способом збільшення обсягу й різноманітності. Засобами обробки інформації можуть бути впеціалізовані програмні ситеми та пристрої, що виконую алгоритмічні функції.

Людство займається обробкою інформації тисячі років. Так перші інформаційні технології грунтувалися на використання писемності та рахунків. Тільки століття тому, було здійснено науковий прогрес, що дав можливість здійснювати іонформатизованію швидше і розвивтаи її.

Актуальність – здатність інформації відповідати вимогам сьогодення.

Інформація поділяється на такі види:

1) За формою водавання інформація поділяється на такі наступні види:

– текстова – що передається у вигляді символів, призначених позначати лексикон мови та у відповідності до мови бути читабельною;

– числова – у вигляді цифрових знаків, що позначають математичні функції;

– графічна – у вигляді подій, зображень, графіків, предметів тощо;

– звукова – у вигляді запису або усна при передачі лексем мови акустичним шляхом.

2) За способом сприйняття для людини інформація поділяється на форми взалежності від типу рецепторів, що її сприймають.

– Зорова – відповідають органами зору. Ми бачимо все навколо.

– Слухова – відповідають органи слуху. Ми чуємо звуки навколо нас;

– Тактильна – відповідають рецептори дотику, тобто те що ми відчуваємо доторкаючись, або коли торкаються нас;

– Нюхова – відповідають рецепторами нюху Ми відчуваємо аромати нашим носом;

– Смакова – відповідають рецептори смаку. Ми відчуваємо смак.

3) За призначенням має такі форми як:

–масова – містить тривіальні відомості і оперує набором понять, зрозумілим більшій частині соціуму або конкретній групі осіб;

–спеціальна – містить специфічний набір понять, при використанні відбувається передача відомостей, які можуть бути не зрозумілі основній масі соціуму, але необхідні і зрозумілі в рамках вузької соціальної групи, де використовується дана інформація;

–особиста – набір відомостей про яку-небудь особистість, що визначає соціальний стан і типи соціальних взаємодій всередині популяції[3].

1.2. Шифрування та дешифрування як процес обробки

Криптографія – наука про математичні підходи та методи забезпечення конфінденційності, автентичності і цілісності інформації. Розвинулась з практичної потреби передавати важливі відомості захищеним способом. Для математичного аналізу криптографія використовує інструментарій вищої математики, алгебри та теорії ймовірностей[4].

Інформація, що може бути прочитана, зрозуміла і осмислена без яких-небудь спеціальних інструментів, називається відкритим текстом. Метод перетворення відкритого тексту таким чином, щоб сховати його суть, називається

зашифровуванням. Шифрування відкритого тексту приводить до його перетворення в незрозумілу форму, іменовану шифротекстом. Шифрування дозволяє сховати інформацію від тих, для кого вона не призначається, попри те, що вони можуть бачити сам шифротекст. Протилежний процес перетворення шифротексту в його вихідний вид називається розшифруванням.

До теперішніх часів криптографія виконувала виключно забезпечення конфідеційності повідомлень – перетворенням самого повідомлення із зрозумілої ворми в незрозумілу і зворотнє відновлення зі сторони отримувача, роблячи незрозумілим для злочинця, що має намір перехопити повідомлення для здійснення злочинних дій. На сьогодні постають криптографія розвинулася далеко в перед і дозволяє реалізовувати методи перевірки цілісності повідомлення, індетифікувати відправника та одержувача, реалізовувати цифрові підписи, інтерактивні підтвердження та технології безпечного спілкування.

Проблеми безпеки, зазвичай, можуть спричинити серйозні загрози для будь-якої системи, тому важливо знати її вразливі сторони. Найпривабливішими є бази даних, бо це є однією із перших цілей для кіберзлочинців, оскільки містять конфіденційну інформацію яка може мати велику ціну. Вона може варіюватися від фінансової або інтелектуальної власності до корпоративних і персональних даних користувачів. Кіберзлочинці можуть отримувати прибуток, проникаючи до серверів компаній і пошкоджувати бази даних. Таким чином, перевірка безпеки бази даних є вкрай обов'язковим[5].

Розглянемо найпоширеніші пробле.Кіберзлочинець генеруючи вразливі та підроблені дані, які можуть генерувати або підробляти дані і відправляти їх до системи. Вони будуть зберігатися разом із дійсними. Для прикладу візьмемо вашу виробничу компанію. Вона використовує дані датчиків щоб виявляти несправні процесів виробництва, злочинці можуть проникнути до вашої системи та змусити датчики показувати підроблені результати, допстимо, неправильні пропускання газу, температури, вологість тощо. Такі дії, можуть дезінформувати контролюючого, так як він не помітить тривожні сигнали і пропустить можливість локалізувати проблему до того, як буде завдано серйозної шкоди. Від

таких вразливостей можна захиститись шляхом застосування підходу виявлення шахрайства, що дозволить усунути проблеми захисту із використанням криптографічного підходу.

Незважаючи на важливість шифрування інформації, цю вимогу безпеки часто ігнорується. Персональні дані, як правило, зберігаються в хмарі або інших носіях без будь-якого захисту шифруванням. Хоча шифрування є відомим засобом захисту конфіденційної інформації, проте не виключає із списку проблем безпеки даних. Причина такої безтурботності очевидна – постійні процеси шифрування та дешифрування навіть незначних частинок даних сповільнюють роботу, де втрачається час, який є дуже вагомим. Можливістю добування важливої інформації в системах зазвичай має застосування захисту на її межах.

Всі “пункти входу та виходу” захищені. Проте те, що роблять IT-спеціалісти у системах, залишається загадкою[6]. Така відсутність контролю у ваших рішеннх за конролю данними може дозволити вашим корумпованим IT-спеціалістам або злому бізнесу конкуренту добувати незахищені дані та продавати їх задля власної вигоди.

Для вашої компані, у свою чергу, можуть бути завдані великі збитки зазнати, якщо така інформація пов'язана із запуском нового продукту або послуги, фінансовими операціями компанії або особистою інформацією користувачів. Так, дані можна краще захистити, додавши додаткові параметри [7]. Окрім того, безпека системи може покращитися з анонімізацією. Якщо хтось намагається отримати персональні дані користувачів які мають перетворений вигляд, тобто у вигляді шифротексту, тоді можна і уникнути проблем злиття даних.

Зазвичай відсутня перевірка безпеки перед розгортанням систем. Найпоширенішою причиною слабкості баз даних є некомпетентність дотримання правил на стадії розгортання. Незважаючи на те, що функціональне тестування проводиться для забезпечення найвищої продуктивності, цей тип тесту не може показати вам, чи робить база даних те, що вона не повинна. Отже,

важливою є перевірка безпеки веб-сайтів різними типами тестів перед повним розгортанням, або застосувати відповідні програмі системи, що дозволяють здійснити перевірку безпеки. Одним із прикладів викрадення – це викрадена резервна копія бази даних. Існують два види загроз для баз даних – це зовнішній і внутрішній. Не рідко випадками бувають боротьби компаній із внутрішніми загрозами навіть більше, ніж із зовнішніми. Власники бізнесу зазвичай не можуть бути на сто відсотків впевнені в лояльності своїх найманих працівників, незалежно від того, яке програмне забезпечення для комп'ютерної безпеки вони використовують і наскільки відповідальними вони є. Кожен, хто має доступ до конфіденційних даних, може вкрасти його та продати його третім сторонам для отримання прибутку. Щоб якось себе забезпечити компанія укладає угоди із працівниками, до зазначаються суворі правила відповідальносі за злиття даних. Щоб уникнути проблеми, існує можливість усунути ризик –це зашифрувати архіви баз даних, застосовувати найсуворіші стандарти безпеки, вразі порушень застосовувати штрафи, використовувати програмне забезпечення кібербезпеки та постійно підвищувати обізнаність ваших команд через корпоративні зустрічі та особисті консультації. Безмежний доступ до адміністрування, відповідно, поганий захист даних[8].

Розумний поділ обов'язків між адміністратором і користувачем гарантує обмежений доступ тільки досвідченим командам. Таким чином, користувачі, які не беруть участь у процесі адміністрування баз даних, відчуватимуть більше труднощів, якщо вони намагатимуться вкрасти будь-які дані. Найкращим рішенням буде обмеження кількості облікових записів користувачів, оскільки кіберзлочинцю доведеться зіткнутися з більшими перешкодами в отриманні контролю над базою даних [9]. Такі випадоки можуть бути застосовані до будь-якого виду бізнесу, зазвичай таке відбувається у фінансовій сфері. Таким чином, добре не тільки дбати про те, хто має доступ до конфіденційних даних, але й виконувати тестування банківського програмного забезпечення, перш ніж випускати його.

Буває і неадекватне управління ключами. Хорошою практикою будев шифрування конфіденційних даних, але також важливо, щоб увагу була звернена на того, хто матиме доступ до ключів.

Оскільки ключі часто зберігаються на чиємусь жорсткому диску, це, очевидно, легка ціль для того, хто хоче їх вкрасти. Краще не залишити такі важливі засоби безпеки програмного забезпечення незахищеними, знайте, що це робить вашу систему вразливою до атак. Шифрування даних – це спосіб захисту, в якому інформація перетворюється в шифрований текст і може бути відновлена або розшифрована лише користувачем із правильним ключем дешифрування. Зашифровані дані, також відомі як шифротекст, виглядають зашифрованими або нечитабельними для особи або суб'єкта, які здійснюють доступ без дозволу. Тож для захисту даних організовують схему обробки даних, як зображено на рисунку 1.1.

Рисунок 1.1. – Приклад організації захисту даних

Процесом перетворення інформації у незрозумілу форму для будь-кого окрім призначеного отримувача називаєтсья – шифрування. Дешифрування є оберненим процесом шифруваня, тобто процес перетворення інформації назад

до зрозумілої форми. Криптографічний алгоритм, який називають шифром, тобто математичниа функція, яка використовується, щоб зашифрувати чи дешифрувати. В переважній більшості випадків, використовують дві пов'язані функції, одну для шифрування та іншу для дешифрування[10].

Застосовуюси сучасні методи криптографії, можливості зберігати зашифровану інформацію у таємниції бере основу не тільки на криптографічному алгоритмі, що є доступним публіці, а на числі, яке називаєть ключем, що мусить бути використаним в алгоритмі для створення зашифрованого результату або для дешифрування уже зашифрованої інформації. Маючи правильний ключк дешифрування є досить простим. Дешифрувати без відомого ключа досить складний процес, що практично є неможливим з практичної точки зору.

Шифрування даних застосовується для запобігання доступу до важливих конфіденційних даних від зловмисника або небажаної особи. Важливим аспеком захисту в архітектурі кібербезпеки, є шифрування, що робить використання перехоплених даних, досить складним, практично не можливим. Застосовується до всіх видів потреб у захищенності даних, починаючи від військових даних на передовій, секретних документів уряду та країни, до операцій для особистих кредитних карток та персональних телефонів. Програмні системи для реалізації шифрування даних, маючи всобі реалізовані алгоритми криптосистем, використовується для розробки схеми та методів шифрування, яке теоретично можна обійти тільки з великими обсягами обчислювальної потужності.

Застосовується шифрування для зберігання важливої інформації в ненадійних джерелах та її передачі по незахищеним каналам зв'язку. Така передача даних представляє із себе два взаємно зворотних процесу: Перед відправленням даних по лінії зв'язку або перед приміщенням на зберігання вони піддаються зашифровуванню.

Використання шифрування присутнє не тільки для передачі в Інтернеті, а й в мобільних телефонах чи транзакціях банкоматів. Доречі, дані все ще можуть бути перехоплені, проте вони будуть незрозумілими і тому не корисними для

хакерів або шпигунів. Різноманіття пристрої в різних мережах зашифровують повідомлення, якими обмінюються. Криптосистеми захищають всі передані дані. Для відновлення вихідних даних із зашифрованих до них можна застосувати процедуру розшифрування. Шифром називається пара алгоритмів, що реалізують кожне із зазначених перетворень. Ці алгоритми застосовуються до даних з використанням ключа. Ключі для шифрування та розшифрування можуть як відрізнятися, так і бути однаковими. Секретність другого з них робить дані недоступними для несанкціонованого читання, а таємність першого не дозволяє вносити неправдиві дані[11].

У перших способах шифрування використовувалися однакові значення ключей, однак в 1976 році були відкрито алгоритми із виеористанням різних ключів. Збереження таких ключів в секретності і правильний їх поділ між адресатами ставить важливе завданням для збереження конфіденційності інформації, що передається. Такі завдання досліджуються в теорії управління ключами.

На даний момент запропоновано величезну кількість методів шифрування. В основному поділяють на симетричні та асиметричні методи, в залежності від структури використовуваних ключів. Крім того, методи шифрування можуть мати різну криптостійкість і по-різному обробляти вхідні дані – блокові шифри і потокові шифри. Всіма цими методами, їх створенням і аналізом займається наука криптографія. Ключ є фундаментальною частиною для захисту конфіденційності інформації, повідомлення або частини даних. Сам процес шифрування та дешифрування може бути ініційований тільки за допомогою ключа.

Ключ шифрування та дешифрування можна порівнювати із звичайним паролем – наприклад, з тим, що ви використовуєте для свого смартфона, електронної пошти тощо. Ключ є невід’ємною частинкою обчислювального процесу кодування і декодування даних. Ключі шифрування зроблені таким чином, щоб вони були абсолютно унікальні, використовуючи набір різних алгоритмів. Ключ шифрування використовується для пкриптографічного

процесу оброби даних. Тоді це означатиме, що ключ шифрування здатний змінювати дані в незрозумілий набір символів та повернути ці символи назад до оригінального відкритого тексту відповідно до ключів дешифратором.

Як правило, ключ є випадковим двійковим або фактичним паролем. Ключ «розповідає» алгоритму про те, які шаблони необхідно дотримуватися, щоб перетворити відкритий текст на зашифрований текст (і навпаки). У зв’язку з тим, що алгоритми є загальнодоступними і доступні для будь-кого, якщо хакер отримає ключ шифрування, зашифровані дані легко розшифруються до відкритого тексту

В криптосистемах, що використовують шифрування симетричними ключами, ключ шифрування може бути вирахуваний за допомогою ключа дешифрування та навпаки. В більшості алгоритмів з симетричними ключами один і той же ключ використовується як для шифруванння, так і для дешифрування, як показано на рисунку 1.2.

Рисунок 1.2 – Робота симетричної криптосистеми

Використовуючи шифрування із симетричними ключами може бути найефективнішим, отже користувачі не відчуваютимуть жодної значної затримки в часі в результаті підчас шифрування та дешифрування. автентифікації використовує шифрування з симетричними ключами, що також забезпечує певний рівень безпеки, тому що інформація зашифрована одним симетричним ключем не може бути дешифрована яимось іншим симетричним ключем. Таким чином, доки симеричний ключ зберігаєтиметься в таємниці двома сторонами, що

використовують його для шифрування обміну інформацією, то кожна сторона може бути впевнена, що обмінюється інформацією з іншою, так само як і до того часу доки дешифровані повідомлення матимуть зміст[12].

Ефективним шифрування з симетричними ключами можна вважати лише тільки тоді, коли симетричний ключ зберігається в таємниці двома залученими сторонами. Проте будь-хто інший може дізнатися ключ, що вплине на конфіденційність та автентифікацію. Особа із неавторизованим симетричним ключем може не тільки дешифрувати повідомлення, відправлені з використанням даного ключа, а й шифрувати нові і віпправляти їх ніби вони були відправлені однією з двох сторін, які від початку користувались ключем, тобто є можливість дезінформації, або відпрвлення хибних даних.

В протоколі SSL зазвичай використовує шифрування з симетричними ключами, і даний протокол використовується для автентифікації, виявлення шифрування та фальсифікації через мережі TCP/IP. SSL також використовує техніки шифрування з використанням публічного ключа, яке описане в наступному розділі.

Найчастіше використовується реалізації шифрування з використанням публічного ключа, що базуються на алгоритмах, запатентованих RSA Data Security. Тому цей розділ описує підхід RSA до шифрування з використанням публічного ключа.

Шифрування з використанням публічного ключа, або ще асиметричне шифрування, залучає пару ключів – публічний ключ та приватний ключ, що пов'язані з об'єктом, який потрібно автентифікувати електронно, зашифрувати чи підписати. З середини 1970-тих активно почали з'являтися дослідження асиметрични криптосистем разом із появою відкритої сертифікації DES національного Бюро Стандартів США та публікацією Діфф-Хелмана та оприлюднення алгоритму RSA. З того часу криптографія перетворилася на широкий і загальний інструмент для передачі даних в комп’ютерних мережах і захист інформації взагалі[13].

Кожен публічний ключ є у вільному доступі, а відповідний приватний ключ зберігається в таємиці. Дані зашифровані ваши публічним ключем можуть бути дешифровані лише вашим приватним ключем. Такі криптосистеми зазвичай використовують для реалізації цифрового підпису. Зазвичай для надійсноті асиметричне шифрування ще доповнюють симетричними алгоритмами для кращої надійності системи. Асиметричні криптосистеми зазвичай ще можуть використовуватися у стандартних програмних системах, що здійснюють процес шифрування і зберігати публічні ключі.

Рисунок 1.3 ілюструє спрощений вигляд того, як працює шифрування з використанням публічного ключа.

Рисунок 1.3 – Робота асиметричної криптосистеми

Схема зображена на рисуноку 1.3 дозволяє вільно розповсюджувати публічний ключ. Лише власник приватного клбюча зможе прочитати зашифровану інформацію. В загальному, щоб відправити зашифровані дані комусь, ви зашифруєте інформацію публічним ключем і особа, що отримує зашифровану інформацію та використовує дешифрування її відповідним приватним ключем.

У порівнянні із шифруванням симетричним ключем, шифрування із використанням публічного ключа потребує білльшого обсягу ресурсів для обчислення і тому не завжди використовується для великих обсягів інформації. Існує можливість використовувати шифрування із використанням публічного ключа для відправки симетричного ключа, що потім може бути застосований для шифрування додаткових даних. На даний момент протокол SSL використовує цей

підхід. Як це буває, зворотній порядок схеми, показаної на рисунку 1.3 також працює: дані зашифровані приватним ключем можуть бути дешифровані лише публічним ключем. Це не може бути бажаним шляхом зашифрування надсекретної інформації, тоді будь-хто маючи публічний ключ, має змогу дешифрувати дані та мати доступ до небажаної інформації.

Шифрування із використанням приватного ключа є досить корисним, адже це означає, що ви можете використовувати ваш приватний ключ щоб підписати дані вашим цифровим підписом, що є важливою вимогою для електронної комерції та інших комерційних сторін використовуючи вимоги криптографії. Програмне забезпечення для клієнта, для прикладу Firefox може пізніше використати ваш публічний ключ для підтвердження того, що повідомлення було підписане вашим приватним ключем і не було підроблене з моменту підписання. Правила цифрових підписів описують як працює процес підтвердження.

Цифровий підпис є видом електронного підпису врезультаті криптографічного перетворення набору даних, що дозволяє логічно математичним методом підтвердити цілісність та ідентифікувати підписувача. Підпис накладається особистим ключем та є здатність перевірити його відкритим ключем. При підписанні електронних документів початковий зміст не змінюється, а додається блок даних, отримання якого можна розділити на два етапи.

На першому етапі за допомогою програмного забезпечення і спеціальної математичної функції обчислюється і називається “Відбитком”. Відбиток має такі властивості:

- фіксована довжина, внезалежності від довжини повідомлення;
- унікальність відбитку для кожного повідомлення;
- неможливість відновити повідомлення без його відбитка.

При модифікації документа і зміниться його відбиток, що відобразиться при перевірці Цифрового підпису. На другому етапі відбиток документа шифрується за допомогою програмного забезпечення та персонального ключа автора.

Розшифрування цифрового підпису та одержання відбитка, який відповідає документу, можна тільки використовуючи Сертифікат відкритого ключа автора. Таким чином, обчислення відбитку захищає документ від модифікації сторонніми особами після підписання, а ще є підтвердження автора.

Пошук ключа для доступу до зашифрованих даних в звичайному тексті називать процесом взлому алгоритму шифрування. Щоб здійснити взлом симетричного алгоритму, означає спробу визначити ключ, який використовувався при шифруванні тексту. Проуес взлому алгоритму із публічним ключем, означатиме отримання секретної інформації, що була поширена між двома отримувачими.

Найпоширенішим підходом взлому симетричного алгоритму шифрування є простим підбором відомих ключів в межах цілого алгориму, допоки не знайдеться вірний. Для алгоритмів з використанням публічного ключа, так як одна половина ключів є в доступі завчасно, інша половина (приватний ключ) може бути вирахувана використовуючи публічний, через комплекс складних математичних розрахунків [14] може дати результат, проте є дуже клопіткою роботою. Ще можна застосувати метод грубої сили «брутфорс атаку», яка є ручним пошуком ключа для взлому алгоритму.

Для симетричних ключів, міцність шифрування часто описується в вигляді розміру чи довжини ключів, що використовуються для виконання шифрування: загалом, довші ключі надають кращий рівень захищеності. Довжина ключа вимірюється у бітах. Візьмемо для прикладу, 128-бітні ключі, що використовуються для шифрування із симетричними ключами, як RC4, що використовується у протоколі SSL надають значно кращий захист упорівнянні із 40-бітними ключами, що використовуються подібним шифром. Можна сказати, що 128-бітне RC4 шифрування в 3x102 разів захищенішим, упорівнянні із 40-бітним RC4 шфруванням. Ключ шифрування вважається повністю захищеним, якщо найкращий відомий метод атаки з метою взлому не є швидшим ніж спроба атаки методом грубої сили, для тестування кожного можливого ключа[15].

Взломи алгоритму знайомить із ризиками перехоплення чи навіть розкриття важливої фнормації та підробки підтвердження. Також повідомляє нас про ризик перехоплення та розкриття приватної інформації. Міцність ключа алгоритму вираховується способом знаходження найшвидшого методу для взлому цього ж алгоритму і порівннюючи цей спосіб з методом грубої сили.

Різні шифри можуть потребувати ключі різноманітної довжни для отримання одного рівня захищеності. Наприклад є шифри, що використовуються для симетричного шифрування, є можливість використовувати всі можливі значення для ключа даної довжини, а не підмножину цих значень. Криптосистема RSA, яка використовується для шифрування із застосуванням публічного ключа, наприклад, може використовувати тільки підмножину всіх можливих значень для ключа даної довжини, все через природу матиматичної проблеми, на якій вона базується. Шифрування зазвичай використовується змішане і може бути здійснено етапно на різних частинах системи, що дозволяє зловмиснику отримати невірний шифротекст, який повністю не відповідатиме оригінальному значенню, так як підчас перехоплення він отримає тільки його частину.

1.3. Відомі програмні засоби для шифрування

На сьогоднішній день у вільному доступі є джерела для організації програмних систем для шифрування даних із можливістю самостійно розробити, або запрограмувати, систему із наявними джерелами алгоритмів та особливостей розробки. Також є наявні готові бібліотеки які можна додавати до програмних продуктів, які дозволяють без глибоких математичних знань, швидко підключити та застосувати її.

Однією із доступною бібліотекою у якій представлений великий спектр функцій з області крипторагіфі “Bouncy Castle”. Дана бібліотека має реалізацію на двох мовах програмування Java і C#, що дозволяє застосувати її на багатьох

платформах. В основі архітектури лежить широкий набір низькорівневої реалізації API.

Доступною для безплатного використання, написаною на мові програмування C++ із відкритим вихідним кодом є Crypto++. Бібліотека має підтримку 32 і 64 розрядних архітектурних систем, особливо таких як Android, Apple, Linux та Windows. Генерування ключів відбувається за допомогою генерування випадкових чисел. Бібліотека не містить обмежень на довжину ключів RSA та DH, вибирається із адекватних для захисту системи симетричних ключів підчас обміну.

Ще однією бібліотекою із відкритим кодом є “Themis”. Має доступну реалізацію на багатьох мовах програмування і часто застосовується як одна із стандартних основ організації захисту даних у програмних системах. Наприклад рідна бібліотека для мови програмування Go використовує дану біботеку із змішаним підбором OpenSSL, BoringSSl. Найпростішим алгоритмом роботи даної бібліотеки є застосування асиметричного крипто алгоритму, що дозволяє обробляти дані із можливістю зашифрувати їх парою ключем і поширювати їх у відповідності до реалізації користувачем.

Програмна система KeePass, розроблена Домініком Райхлом для операційної системи Windows, дозволяє зберігати інформацію про паролі і коди доступу. . На рисунку 1.4 зображено інтерфейс програмної системи.

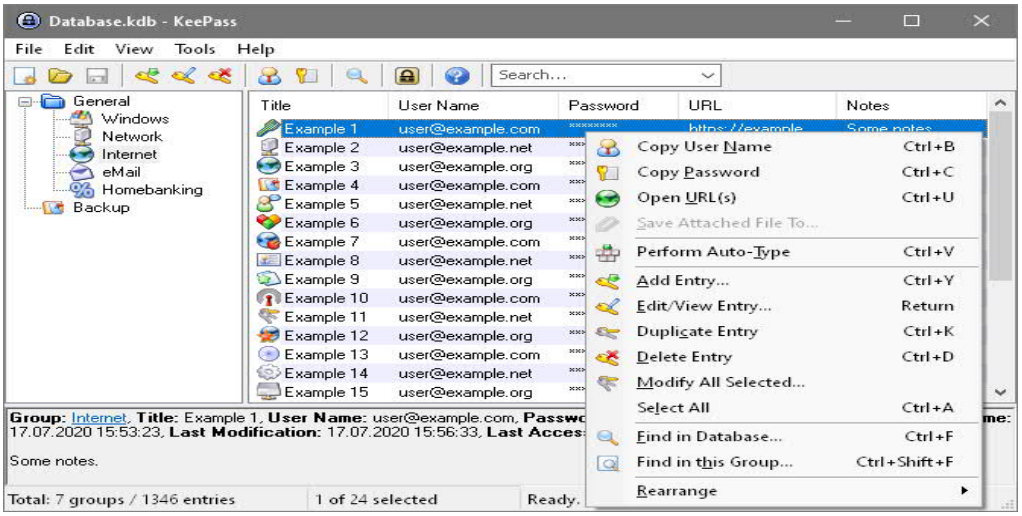

Рисунок 1.4 – Інтерфейс програмної систми KeyPass

Для здійснення безпеки застосовує алгоритм Twofish і Advanced Standard. Одними із переваг, як зазначають виробники, є портативність та зручність, відсутність додаткових необхідних установок та експорт даних у різні формати.

Однією із доступних програм і вільним у доступі є програма “GEAT”. Програмна система реалізована добровольцями із однофменного форуму. Програма реалізована на мові програмування Java, тобто є надія котувача, що дана программа буде працювати на різних платформах. Програмна система надає можливість користувачу генерувати ключ, потім обрати файл який бажається зашифруватися. На рисуноку 1.5 зображено фнрфейс програмної систему GEAT

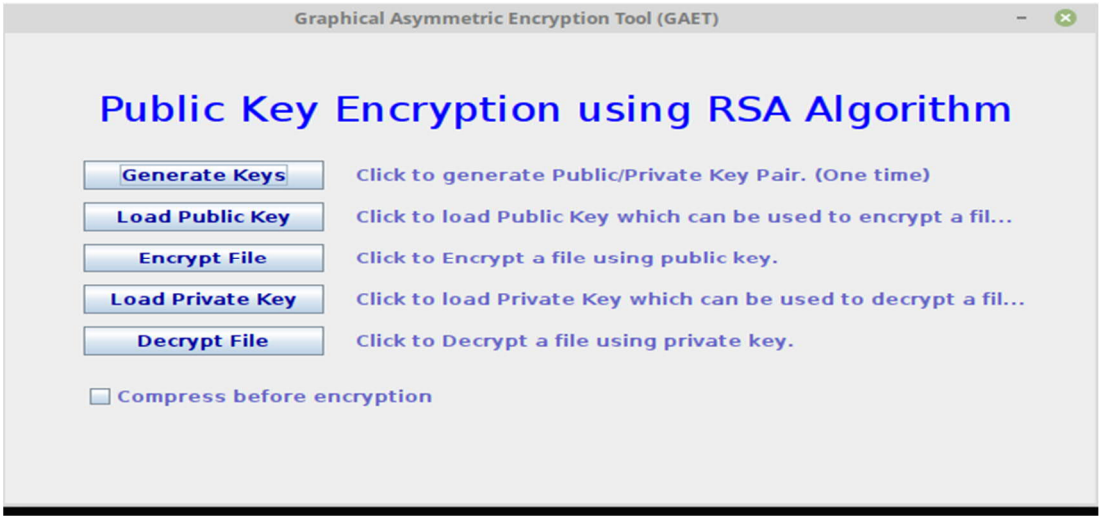

Рисунок 1.5 – Інтерфейс програмної системи GEAT.

Дана програмна система також дозволяє зашифрувати фото, протее робота алгоритму реалізована на класичних підходах і виконання операцій забирає багато часу. Ще одним із недоліків данної програмної системи, це неможливість повноційнної роботи на інших платформах, що робить не зовсім гнучкою. Також дослідивши внутрішню реалізацію, виявлено, що максимальний розмір ключа обмежена до 1024 біт.

Ще одним із хороших аналогів і також є вільним у доступі є програмна система Kleopatra. Розроблена під операційну систему Windows та має максимальне знаяення ключа 4096 біт.

Дана програмна система розвивається і підтримується добровольцями і регулярно оновлюється, тож на сьогодні містить широкий спектр організації захисту даних, файлів та дозволяє організувати зручну покроковість генерування ключів, прив’язку до багатьох компонентів і навіть до облікового запису Windows.

Програма містить ряд інструментыв, які будуть інсталюватися у пакеті. Додаткові програми дозволяють організувати сховище згенерованих ключів. Також є можливість побачити термін дії ключа та від слідкувати її додаткову інформацію. На рисунку 1.6 зображено головне вікно програмної системи.

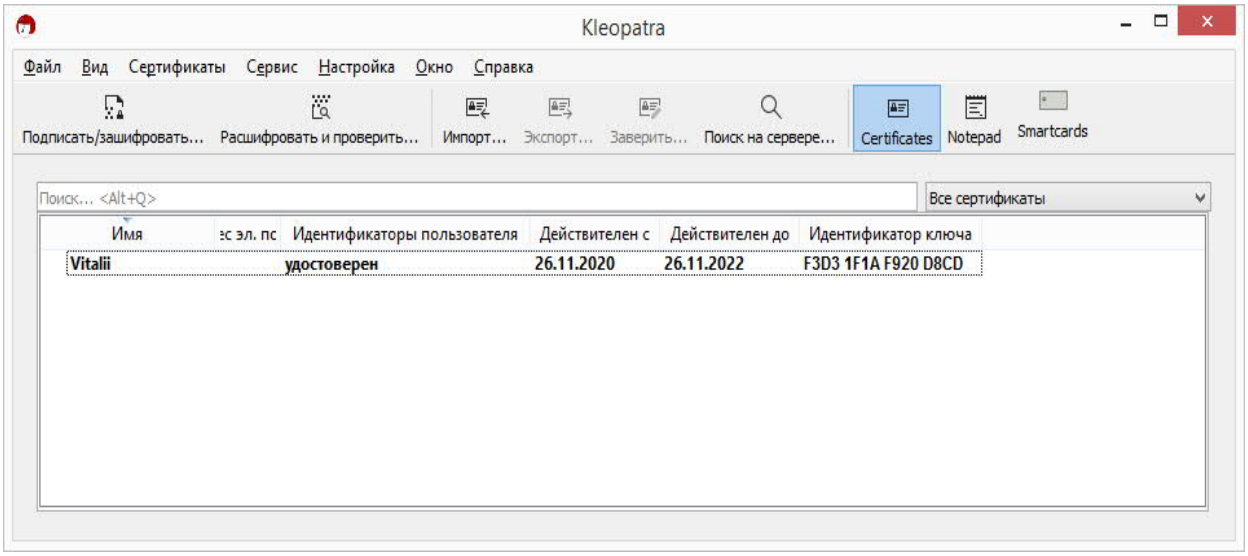

Рисунок 1.6 – Головне вікно програмної системи Kleopatra

Одним із недоліків програмної системи є наявність прив'язки її до платформи, обов'язковим наявність додаткових пакетних программ, щоб здійснювати повноцінну роботу. Наприклад при встановленні даної програми, довелося погоджуватися на встановлення ряду додаткових програм, щоб забезпечити дієздатність. Проте наявність широкого спектру функцій забезпечує користувача всіма необхідними гнучкими інструментами.

Програмна система криптографічного захисту інформації на інформаційних пристроях "Криптософт Storage" призначена для забезпечення конфіденційності та цілісності інформації за допомгою здійснення її криптографічного перетворення тобто виконання оброблювальних функцій шифрування та розшифрування, здійснення розмежування доступу до інформації, що зберігається на носіях інформації (жорсткий диск комп'ютера, сервер, хмари, переносний жорсткий диск, флеш-накопичувач) відповідно до ключових даних та сертифіката відкритого ключа. Забезпечує використання у складі автоматизованих системах класів, обробку інформації, можливість використовувати у складі будь-якої інформаційної системи. Реалізовує такі механізми, як контроль якісної цілісності програмного забезпечення. Є продукцією компанією IT Engineering

Даний програмний засіб розповсюджуєтьсяяк планий продукт і для спрои є можливість встановлення демоверсії, проте вона немає багато можливостей, особливо із перевірки внутрішніх налаштувань із можливістю налаштувати ромір розрядності ключа.

Рисунок 1.7 – Програмна система Криптософт Storage

VeraCrypt – доступне багато платформне програмне забезпечення, що використовує шифрування для дисків та файлів. Підтримує розпаралеленя на

багатопроцесорних і багатоядерних системах, також вміє використовувати апаратне прискорення шифрування, яке доступне на процесорах. На рисунку 1.8 зображено роботу програмної системи.

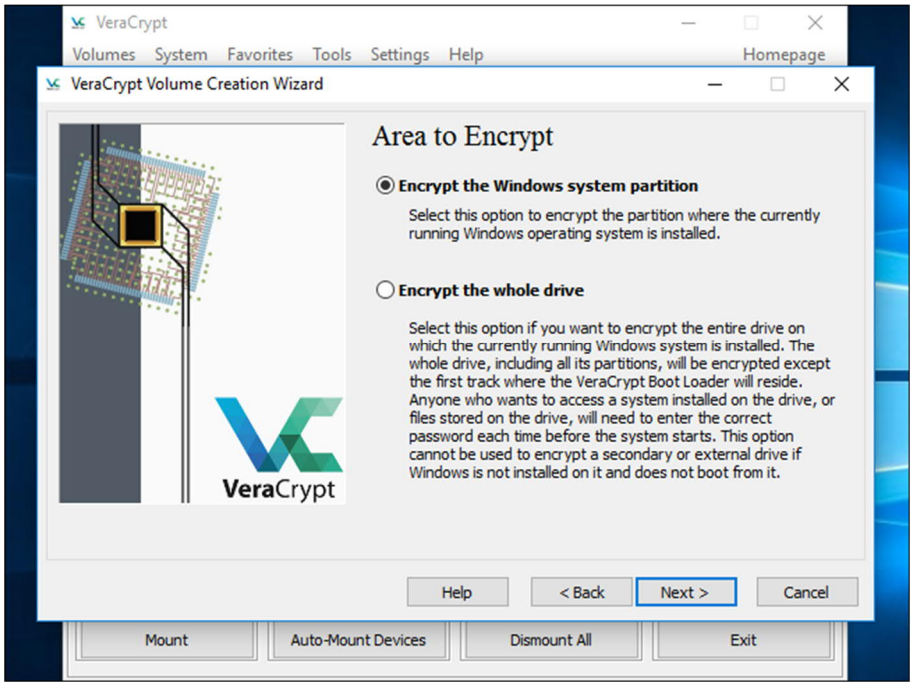

Рисунок 1.8 – Робота програмної системи VeraCrypt.

Програмна сисмета використовує наступні алгоритми шифрування,AES, Perpent та Twofish. Також є наявність п’яти комбінацій даних алгоритмів. Первинний і вторинний ключі заголовку, для режиму XTS, генеруються за допомогою алгоритму PBKDF2 з використанням 512-бітного криптографічного секрету, число ітерацій становить від 327,661 до 655,331, залежно від використанної хеш-функції.

2. МАТЕМАТИЧНІ МОДЕЛІ АЛГОРИТМІВ

2.1. Симетричні криптосистеми

Розглянемо класичну модель К. Шеннона симетричну криптосистему у якій три учасники мають наступні завдання:

– відправник має намір по відкритому каналу передати якесь у захищеному вигляді повідомлення. Для цього він використовуючи ключ *k* шифровує відкритий текст *X* і передає текст у зашифрованому вигляді *Y*;

– отримувач хоче розшифрувати *Y* та прочитати отримане повідомлення *X*. Передбачається, що сам відправник має власне джерело ключа. Згенерований ключ передається на заздалегідь надійному каналу одержувачу;

– злочинець має намір перехопити повідомлення та імітувати помилкові повідомлення або використати їх у своїх корисливих цілях.

Симетричні алгоритмми є досить популярним і найпопулярнійшим є відкритий стандарт для шифрування даних DES, розроблений компанією IBM. Для пояснення роботи алгоритму розглянемо рисунком. 2.1.

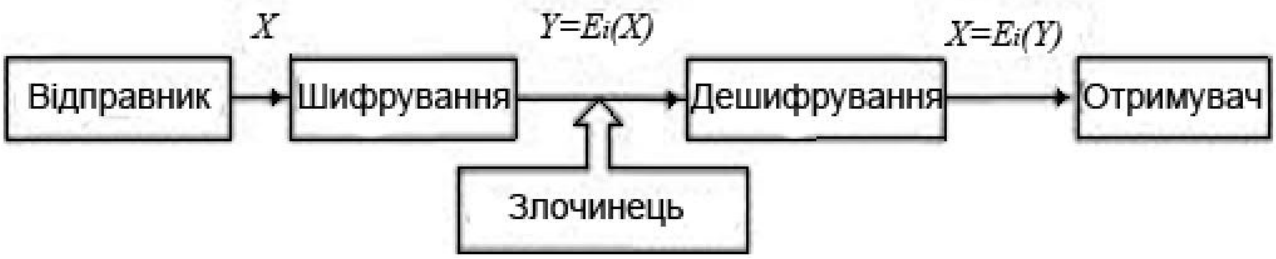

Рисунок. 2.1 – Симетричне шифрування

Перед шифруванням вхідні дані (блок тексту) перетворюють в число шляхом відкритої будь-якої процедури. Простим підходом може бути використання ASCII-кодів послідовнності символів тексту, після якого може бути отримано двійкове число. Розмір блоку повідомлення повинен становити 64 біта. Поділяється блок на дві частини, праву *R* і на ліву *L* частини і надходять на вхід шифрувальної функції для виконання попередньої обробки. Виконання обробки

поляга в тому, що права частина обраховується як логічна сума по модулю 2 (операція додавання по модулю 2) лівої і правої частин вихідного блоку., а на місце лівої частини результуючого блоку *L* поміщається права частина *R* вихідного блоку. Роботу алгоритму зображено на рисунку 2.2.

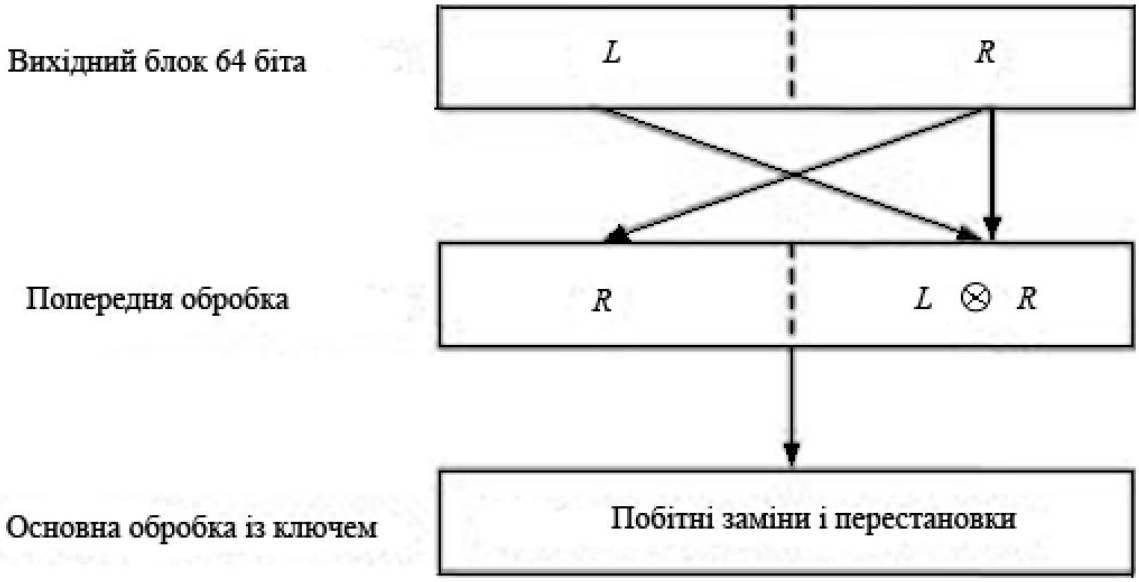

Рисунок. 2.2. – Ілюстрація алгоритму DES

Основну обробку виконує ключ даного алгоритму у вигляді двійковій послідовності довжиною 64 біта, з яких вибираються 8 використовуються для контролю ключа,а 56 біт випадковим чином. За допомогою цієї випадкової функці двійкова послідовность за певною схемою виконується побітно заміна і перестановка [21].

Широко застосовуваним є Алгоритм DES. Алгоритм використовується в різних продуктах безпеки інформаційних систем і технологіях. Щоб підвищеити криптостійкість алгоритму DES застосовують триразове шифрування із використанням двох різних ключів. При цьому вважається, що довжина ключа збільшується від 56 до 112 біт. Цей алгоритм із підвищеною криптостійкістю називається “потрійним DES”. Підчас обчислення потребуватиме в трийчі більше часу у порівняні із звичайний DES.

Підчас обміну секретними даними за принципом “кожен з кожним” в системі з певною кількістю користувачів виникне потреба у великій кількості ключів. Ключі повинні бути згенеровані та надійним чином розподілені. Проблемою постає якраз самий ключ для симетричних алгоритмів . Цю проблему вирішують асиметричні криптосистми, засновані на використанні відкритих ключів[22].

2.2. Асиметричні криптосистеми

Асиметричні криптосистеми застосовують у функціях шифрування та дешифрування одного ітого самого ключа. Такій схемі притаманні наступні недоліки:

– принциповою мусить бути надійність каналу для передачі ключа другому учаснику секретних переговорів. Тобто, ключ повинен передаватися по секретному каналу;

– до служби генераціювання ключів ставляться підвищені вимоги, обумовлені тим, що для j користувачів при схемі взаємодії "кожен з кожним" потрібно $j(j\text{-}1)/2$ ключів, отже, залежність кількості ключів від кількості користувачів є квадратичною.

Щоб вирішити перераховані вище проблем симетричного шифрування призначені системи, використовують змішання із асиметричним шифруванням або шифруванням з відкритим ключем, що використовують властивості функцій з секретом, розроблених Діффі і Хеллманом. Асиметричні криптосистеми мають унікальність у вихідному відкритому ключі для зашифрування та розшифрування Ідею такого алгоритму стало застосовування односторонньої функції $f(x)$. Що дозволяє ускладнювати процес взлому. Схему роботи асиметричного крипто алгоритму зображено на рисунку 2.3.

Рисунок 2.3 – Схема асиметричних криптосистем

Ці системи характеризуються наявність у кожного користувача двох ключів: відкритого і закритого. Відкритий ключ може відкрито передаватися всім учасникам секретних переговорів. Таких підхід вирішує дві проблеми:

– відсутня потреба в секретній доставці;

– квадратична залежність кількості ключів від кількості користувачів відсутня – для *j* користувачів потрібно два *j* ключів.

Перший шифр, розроблений на принципах асиметричного шифрування, є шифр RSA. Він названий так за першими літерами прізвищ його винахідників: Леонарда Елдемана Аді Шаміра і Рона Райвеста – засновників фірми RSA Data Secutity. RSA – як найпопулярніший з асиметричних шифрів і найвідоміший [23].

Математичне обгрунтування RSA таке – пошук дільників дуже великого натурального числа, яке є добутком двох простих чисел, що є дуже трудомісткою процедурою. Така процедура зазвичай є укскладеною обчислюванням і є часозатратною. Відповідно, за допомогою відкритого ключа дуже складно обчислити відповідний йому таємний ключ. На рисунку 2.4 зображено схему криптоалгоритму RSA.

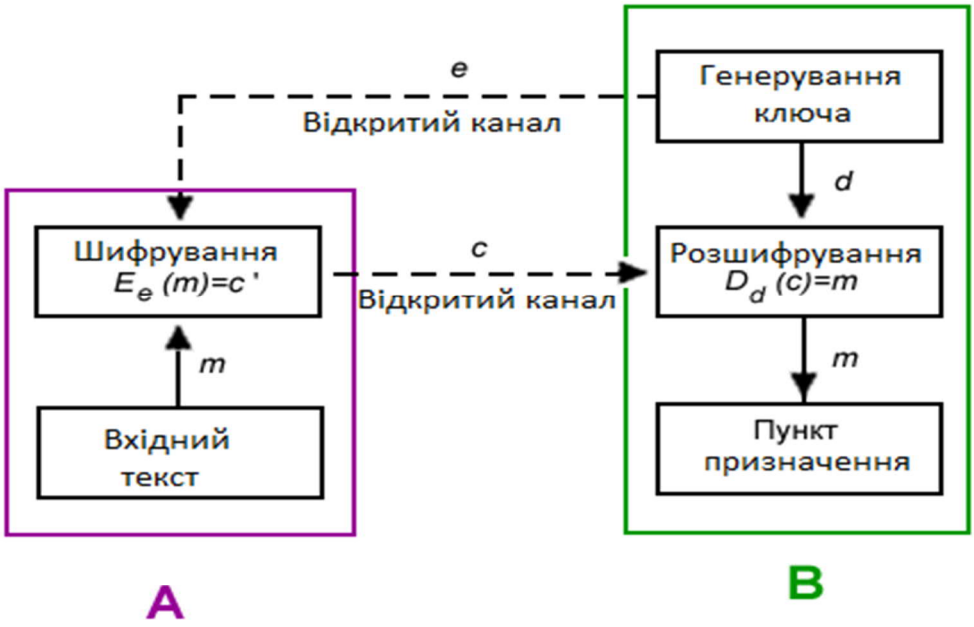

Рисунок 2.4 – Криптоалгоритм RSA

Шифр RSA глибоко вивчений і правдиво визнаний стійким при наявності достатньої довжини ключів. Застосування даного шифру можна побачити прктично в кожні системі шифрування даних. Навіть, для забезпечення стійкості не вистачає 512 біт проте якщо обрати 1024 біти, тоді вважається прийнятним варіантом. З розвитком потужностей процесорів при даній довжині ключа RSA втрачає стійкість до атаки повним перебором, проте, це дозволяє застосувати більш довгі ключі, що в свою чергу підвищить стійкість шифру[24].

Алгоритм шифрування працює таким чином, що включає в себе генерування ключів, шифрування та дешифрування.

Для того, аби згенерувати пару ключів, виконуються такі дії:

– вибирати два великі прості числа p та q;

– обчислити їх добуток $n=p\cdot q$;

– використати функцію Ейлера $\varphi(n)=(p-1)(q-1)$;

– обираємо ціле число e таке, щоб $1<e<\varphi(n)$ та e - взаємно просте з $\varphi(n)$, отже НСД(e, $\varphi(n)$)=1;

– на основі розширеного алгоритму Евкліда знаходиться число d таке, що $ed \pmod{\varphi(n)}=1$ або $d=e^{-1} \pmod{\varphi(n)}$.

Модулем називається число n, а числа e і d - відкритою та приватною експонентами відповідно. Відкритою частиною ключа є пара чисел (n,e) , а пара (n, d) – секретною. Числа p і q після генерації ключів можуть бути знищені, однак вжодному разі не повинні бути відкриті[25].

Щоб шифрування повідомлення відбулося, припустимо, що перший користувач хоче відправити другому повідомлення A. Для початку він повинен використати доповнювальну схему перетворення, що означатиме як узгоджений протокол перетворення, або таблиці перетворення, перетворює A в ціле число a так, щоб $0 \leq a \leq n$. Опісля він обчислює зашифрований тест A', використовуючи відкритий ключ другого користувача e, за допомогою рівняння:

$$A' = a^e \bmod n. \tag{2.1}$$

Це може бути зроблено досить швидко, навіть у випадку, коли текст перевищує 500-бітну розрядність.

Розшифрування виконається наступним чином:

$$a = (A')^d \bmod n. \tag{2.2}$$

Відповідно при виконанні даної операції відновлюєтсья вихідне повідомлення:

$$(A')^d \equiv (a^e)^d \equiv a^{ed} \pmod{n}, \tag{2.3}$$

З умови

$$ed \equiv 1 \pmod{\varphi(n)}, \tag{2.4}$$

випливає твердження, що $ed \equiv k_0 \varphi(n) + 1$ для деякого цілого k_0, отже:

$$a^{ed} \equiv a^{k0\varphi(n)+1} \pmod{n}. \quad (2.5)$$

Згідно з теоремою Ейлера:

$$a^{\varphi(n)} \equiv 1 \pmod{n}, \quad (2.6)$$

тому

$$a^{k0\varphi(n)+1} \equiv a \pmod{n}, (A')^d \equiv a \pmod{n}. \quad (2.7)$$

Здійснення роботи шифрування нам необхідно знати таку пару чисел *e, n* та *d, n*. для дешифрування. Перша пара – відкритий ключ, друга – закритий. Маючи відкритий ключ, можна обчислити значення закритого ключа. Необхідною проміжною операцією цього перетворення є знаходження множників *p* і *q*, для чого потрібно розкласти *n* на множники – ця на виконання цієї процедури витрачається дуже багато часу. Саме з величезною обчислювальною складністю пов'язана криптостійкість RSA.

Рабін розробив криптосистему, яка стала першою асиметричною криптосистемою, яка базується на складності знаходження квадратичного лишку. Вона, як і будь-яка асиметрична криптосистема, використовує відкритий і закритий ключі. Відкритий ключ використовується для шифрування повідомлень і може бути опублікований для загального огляду. Закритий ключ необхідний для розшифрування і повинен бути відомий тільки одержувачу зашифрованих повідомлень. Великою перевага даниї криптосистеми втому, що випадковий текст може бути повністю відновелно повністю від зашифрованного тексту тільки при умові, якщо дишифрування здатне до ефективної факторизації відкритого ключа *n*[26].

Схема є доказово стійкою до атаки на основі підібраного відкритого зашифрованого тексту в рамках підходу “все або нічого”, тоді і тільки тоі коли

при розкладанні цілого числа на прості множники є важкою. Схема шифрування згідно криптосистеми Рабіна наведена на рисунку 2.5.

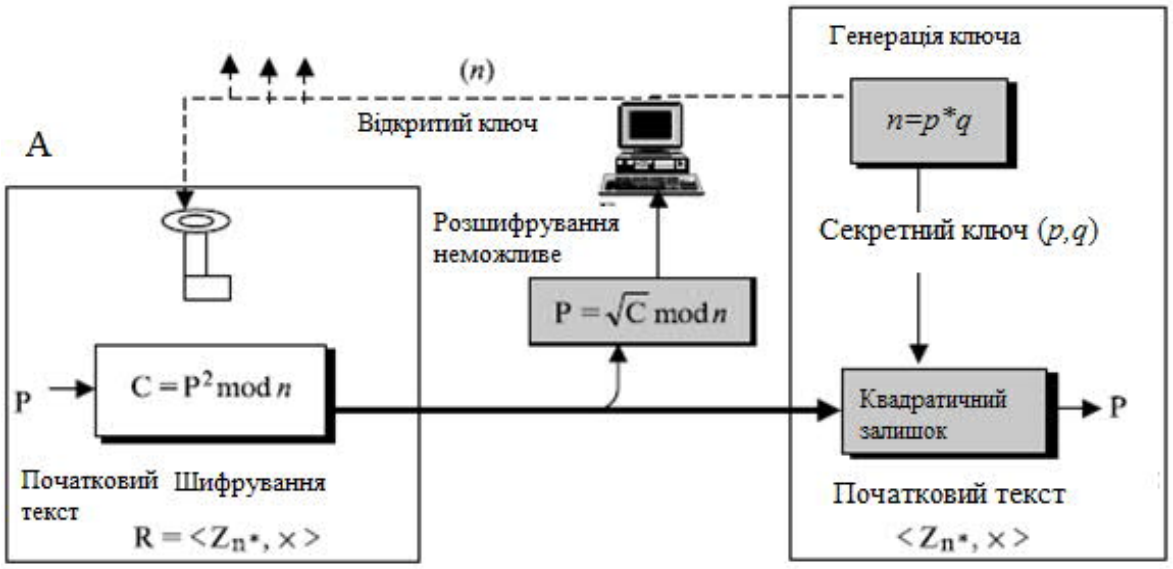

Рисунок 2.5 – Схема шифрування згідно криптосистеми Рабіна

Для здійснення генерації ключів обираються два випадкових числа p та q із врахуванням таких вимог:

– числа повинні бути великими;

– числа повинні бути простими;

– повинна виконуватися умова: $p \equiv q \equiv 3 \bmod 4$.

Виконання цих вимог сильно прискорює процедуру знаходження коренів за модулем p і q. Далі обчислюється число $n=p \cdot q$, тоді число n – відкритий ключ, числа p і q – закритий.

Початкове повідомлення A (текст) шифрується з допомогою відкритого ключа – числа n – за такою формулою:

$$A' = A^2 \bmod n. \qquad (2.8)$$

Використання операції піднесення до квадрату за модулем забезпечує покращення швидкості шифрування за допомгою системи Рабіна, упорівнянні із

швидкість шифрування за методом RSA, навіть якщо в останньому випадку значення експоненти буде вибрання невелике.

При дешифруванні криптограми A' для зручності вводяться додаткові допоміжні величини c_1 і c_2:

$$c_1 = {}^{\circ}A' \mod {}^{\circ}p,\ c_2 = {}^{\circ}A' \mod {}^{\circ}q, \tag{2.9}$$

Для знаходження A необхідно знайти квадратичні лишки c_1, c_2 відповідно за модулями p і q:

$$x^2 \equiv c_1 (\mathrm{mod}\ p),\ y^2 \equiv c_2 (\mathrm{mod}\ q) \tag{2.10}$$

В результаті можна виділити чотири системи порівнянь:

$$\begin{cases} A_1 \equiv x(\mathrm{mod}\ p); \\ A_1 \equiv y(\mathrm{mod}\ q); \end{cases} \begin{cases} A_2 \equiv x(\mathrm{mod}\ p); \\ A_2 \equiv -y(\mathrm{mod}\ q); \end{cases} \begin{cases} A_3 \equiv -x(\mathrm{mod}\ p); \\ A_3 \equiv y(\mathrm{mod}\ q); \end{cases} \begin{cases} A_4 \equiv -x(\mathrm{mod}\ p); \\ A_4 \equiv -y(\mathrm{mod}\ q); \end{cases} \tag{2.11}$$

Одне із рішень (2.11), що ґрунтується на застосовуванні китайської теореми про залишки і буде шуканим повідомленням A.

Розшифрування тексту, окрім правильного, призводить ще до трьох хибних результатів, що є однією із головних проблем у незручному використанні криптосистеми Рабіна. Такі проблеми є одним із факторів, які стають перешкодоюь, тому для того вби вона знайшла широке практичне використання, слід дослідити математичну модель, де виникають проблеми. Якщо вихідний текст являє собою текстове повідомлення, тоді не є складним визначити правильний текст. Проте повідомлення може бути потоком випадкових бітів (наприклад, для генерування ключів чи цифрового підпису), то тоді визначення потрібного тексту стає реальною проблемою[27]. Одним із способів вирішити

цей недолік є додавання до повідомлення перед шифруванням відомої якоїсь або.заголовка.

Однак у класичній криптосистемі Рабіна блок відкритого тексту обмежується величиною відкритого ключа ($A<n$). Тому для досить довгих повідомлень потрібно кожен блок шифрувати окремо. Приблизно такою ж обчислювальною складністю, як і факторизація, володіє операція дискретного логарифмування у скінченному полі, на якій ґрунтується асиметрична криптосистема Ель-Гамаля. Вона включає в себе алгоритм шифрування та алгоритм цифрового підпису.

У 1985 році Тахером Ель-Гамалем була запропонована схема яка і являється одним із вдосконалених варіантів алгоритму Діффі-Хеллмана, які застосовуються для шифрування та забезпечення аутентифікації для криптосистеми.

Алгоритм виконується в такій послідовності:

1) генеруємо просте число p;
2) вибираємо ціле число g – що є первісним коренем p;
3) вибираємо випадкове ціле число a щоб відповідало $1<a<p-1$;
4) обчислюємо $h=g^a \bmod p$;
5) відкритим ключем будуть три числа (p, g, h) і приватним ключем – число a.

Повідомлення A, яке повинно бути менше від числа p, шифрується таким чином:

1) вибирається сесійний ключ – випадкове ціле число r таке, що $1<r<p-1$;
2) вираховуються числа $c_1=g^r \bmod p$ і $c_2=h^rA \bmod p$.

Пара чисел (c_1, c_2) є шифротекстом.

Одним з недоліків криптосистеми Ель-Гамаля є те, що довжина шифротексту вдвічі більша від вихідного повідомлення A.

Знаючи закритий ключ , вихідне повідомлення можна обчислити із шифртексту (c_1, c_2) за такою формулою:

$$A = c_2 (c_1^{-1})^a \bmod p \qquad (2.12)$$

При цьому дуже легко перевірити, що $(c_1^{-1})^a \bmod p = g^{-ra} \bmod p$ і тому $c_2(c_1^{-1})^a \bmod p = (h^r A)g^{-ra} \bmod p = (g^{ra}A)g^{-ra} \bmod p = A \bmod p$.

Для практичних обчислень більше підходить наступна формула:

$$A = c_2 (c_1^{-1})^a \bmod p = c_2 (c_1)^{p-1-a} \bmod p \qquad (2.13)$$

Оскільки в схему Ель-Гамаля вводиться випадкова величина r, то шифр Ель-Гамаля можна назвати шифром багатозначної заміни. Через випадковість вибору числа r така схема ще називається схемою імовірнісного шифрування. Імовірнісний спосіб шифрування є перевагою для криптоалгоритму Ель-Гамаля, тому що у таких схемах спостерігається більша стійкість у порівнянні зі схемами із певним процесом шифрування. Недоліком схеми шифрування Ель-Гамаля є подвоєння довжини зашифрованого тексту у порівнянні із початковим текстом[28]. Для схеми імовірнісного шифрування саме повідомлення а ключ не визначать, однозначний, шифротекст. В схемі Ель-Гамаля необхідно використовувати різні значення випадкової величини щоб шифрування різних повідомлень . Якщо використовувати однакові , то для відповідних шифротекстів (c_1, c_2) та (c_1', c_2') виконується співвідношення $c_2(c_2')^{-1} = A(A')^{-1}$. З цього виразу можна легко обчислити A', якщо відомо . На цей час криптосистеми із відкритим ключем вважаються найбільш перспективними. Адже можливість реалізації і широка можливість вдосконалювати обчислювальну складність є доступною і запропонованою у багатьох роботах, тож потрбіно здійснювати пошук найефективніших рішень, виконувати їх комбінува

Схема шифрування на основі криптоалгоритму Ель-Гамаля представлена на рисунку 2.6.

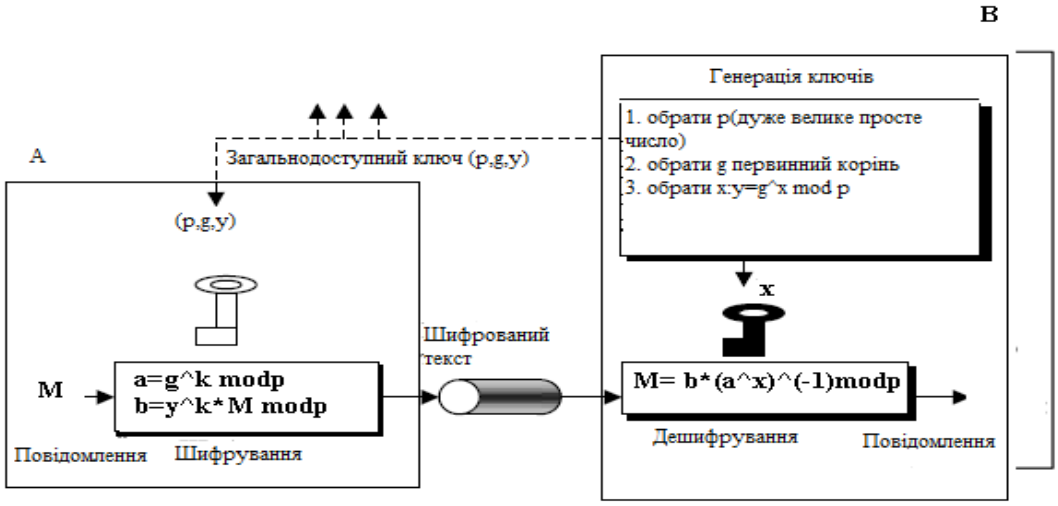

Рисунок 2.6 – Схема шифрування на основі криптоалгоритму Ель-Гамаля

Однак усі операції у проаналізованих криптоалгоритмах відбуваються в двійковій або десятковій системі числення, у яких відсутня можливість розпаралелення процесу обчислень.

2.3. Алгоритм шифрування Криптолайт

У всіх асиметричних алгоритмах шифрування інформаційних потоків найважливішими операціями є модулярне множення та модулярне експоненціювання багаторозрядних чисел. Для підвищення ефективності їх обчислення багатьма авторами запропоновано різні підходи та алгоритми, які розроблені, в основному, з використанням десяткової системи числення, що значно збільшує час виконання обчислення програмною системою.

Одним з підходів щодо підвищення швидкодії знаходження результату модулярного множення є метод Карабуци, але він практично не використовується через складність його реалізації. В описано алгоритм Шенхаге – Штрассена з використанням швидкого перетворення Фур’є, який потребує

$O(n_0 \log(n_0) \log(\log(n_0))$ бітових операцій. Також проблематикою реалізації скаладли обмежені можливості технологій часу[29].

В циклі робіт запропоновано матричні та векторні методи виконання основних арифметичних операцій для знаходження залишку, множення та піднесення до степеня на основі розмежованої двійкової системи числення. Зокрема, для пошуку залишку $a \bmod p$ число a необхідно записати в двійковій системі числення $a=a_{n_0-1}2^{n_0-1}+ \dots +a_i2^i+ \dots +a_1 2+a_0$, де n_0 – розрядність числа a. Тоді:

$$a \bmod p = (\sum_{i=0}^{n_0-1} (a_i 2^i \bmod p)) \bmod p = (\sum_{i=0}^{n_0-1} (a_i a_{1i})) \bmod p \qquad (2.14)$$

де a_i=0 або 1, $a_{1i}=2^i \bmod p$.

З (2.14) випливає, що шуканий залишок дорівнюватиме сумі тих степенів двійки (або a_{1i}), для яких відповідно a_i=1, що проілюстровано в таблиці 2.1.

Таблиця 2.1 – Таблиця знаходження залишку $a \bmod p$.

i	n_0-1	n_0-2		2	1	0
a_i	a_{n0-1}	a_{n0-2}	…	a_2	a_1	a_0
$2^i \bmod p$	$2^{n_0-1} \bmod p$	$2^{n_0-2} \bmod p$		$2^2 \bmod p$	$2^1 \bmod p$	$2^0 \bmod p$
a_{1i}	$a_{1\,n_0-1}$	$a_{1\,n_0-2}$	…	a_{12}	a_{11}	a_{10}

Слід зазначити також, що два послідовні значення a_{1i} та $a_{1\,i+1}$ пов’язані таким рекурентним співвідношенням:

$$a_{1i+1} = \{2 \cdot a_{1i}, 2 \cdot a_{1i}\, p \qquad (2.15)$$

Отже, щоб знайти залишок за модулем в (2.15) не обов’язково виконувати обчислювально витратну операцію ділення з остачею, а можна обмежитися тільки відніманням. Ще, множення на 2 дуже просто реалізується за допомогою

дописування нуля в кінці двійкового запису числа. В таблиці 2.2 наведено приклад пошуку 171 mod 31.

Таблиця 2.2 – Пошук залишку 171 mod 31

i	7	6	5	4	3	2	1	0
a_i	1	0	1	0	1	0	1	1
$2^i \bmod 31$	$2^7 \bmod 31$	$2^6 \bmod 31$	$2^5 \bmod 31$	$2^4 \bmod 31$	$2^3 \bmod 31$	$2^2 \bmod 31$	$2^1 \bmod 31$	$2^0 \bmod 31$
a_{1i}	4	2	1	16	8	4	2	1

Отже, 171 mod 31=(4+1+8+2+1) mod 31=16. Зазначимо, що результат отриманий є на основі додавання мало розрядних залишків степенів двійки за відповідним модулем.

Одним із напрямів підвищення швидко дії асиметричних криптосистем це є пошук нових підходів до виконання арифметичних операцій модулярного множення та експоненціювання. У запропонованому варівнті, організація розпаралеленого паралельного виконання модулярного експоненціювання і встановлено, що паралелізм при цьому є найбільш ефективною формою.

Практична реалізація такої можливості здійснюється за допомогою організованого прискореного обчислення модулярної експоненти на процесорних потоках, що надаються апартним забезпеченням. Доведено теоретично і експерементально, що розроблена організація забезпечує практично n-кратне прискорення виконання операції модулярного експоненціювання для розрядностей 2048 і 4096, де n – це кількість ядер у процесорі.

Застосування системи залишкових класів в методі Монтгомері є ефективним способом організування збільшення швидкодії модулярного множення, проте його часова складність все ще залишається високою при опрацюванні багаторозрядних чисел.

Навіть беручи до уваги існуючі ефективні алгоритми обробки інформаційних потоків у СЗК, що реалізуються на основі програмно-апаратних спецпроцесорів, які забезпечують захист від помилок і певний рівень захисту інформації від несанкціонованого втручання, потрібно ще додатково

досліджувати, аналіз і ефективне застосування нових алгоритмів та методів при застосуванні різних форм СЗК в асиметричних криптосистемах[30].

Для множення двох n – розрядних чисел $a = \sum_{i=0}^{n-1} a_i \cdot 2^i$ та $b = \sum_{j=0}^{n-1} b_j$, де a_i, b_j=0, 1, n–розрядність модуля p за допомогою векторно-модульного методу, потрібно побудувати два вектор-рядки, в першому з яких записуються елементи $c = 2^0 \bmod p$, $c = 2 \cdot c_{i-1} \bmod p$ другий - з a_i, як показано в таблиці 2.3

Таблиця 2.3 – Представлення вектор-рядків медулярного множення

c		c	…	c	c
$a_{n\text{-}1}$	…	a_j	…	a_1	a_0

Результатом модулярного множення двох n – розрядних чисел знаходиться згідно формули:

$$a \cdot b \bmod p = (\sum_{i=0}^{n-1} a_i \cdot c_i) \bmod p \qquad (2.16)$$

Розроблений метод характеризується меншою часовою та апаратною складністю порівняно з матрично-модульним.

Сучасні системи захисту інформації інтенсивно розвиваються та вдосконалюються на основі нових теоретичних положень опрацювання інформаційних потоків та програмно-апаратних засобів реалізації асиметричних криптоалгоритмів. Можна вважати що сучасний етап розвитку технологій дозволяє застосувати широкий ряд інструментів для покращення обробки, наприклад взяти до уваги розпаралелення обчислення у потоках, або використання кращих бібліотек які мають восі вже готові рішення для здвйснення пришвидшеного обчислення.

При цьому, на сучасному етапі розвитку інформаційних технологій виникає ряд проблем та науково-технічних завдань пов’язаних із підвищенням

інформаційної стійкості комп'ютерних систем, зменшення обчислювальної складності асиметричних алгоритмів шифрування та дешифрування.

Слід зазначити, що багатьма авторами розроблені методи різної складності для вирішення даного класу задач. Такі розроблені рішення дозволяють застосувати їх у розробках нових програмно апаратних системах. Одним з найбільш ефективним щодо швидкодії є метод з використанням розмежованої системи числення Радемахера–Крестенсона, з часовою складністю, як на формулі 2.17.

$$O(n) = \begin{cases} n \log_2 n, \textit{якщо} \quad n < 64 \\ \frac{n^2}{2} \log_2 n \cdot, n \geq 64 \end{cases} . \qquad (2.17)$$

В цьому випадку часова складність зменшується з n^3 у двійковій системі числення до $\frac{n^2}{2}\log_2 n$.

Враховуючи та операючись на матеріальні та інформаційні дослідженнях постає проблема у пошуку ефективного підходу реалізації алгоритму із кращою швидкодійною вдосконаленою формою шифрування та дешифрування даних [31].

Нехай маємо два числа із розрядністю n_0: $a=a_{n_0\text{-}1}2^{n_0\text{-}1}+\ldots+a_i2^i+\ldots+a_12+a_0$ та $b=b_{n_0\text{-}1}2^{n_0\text{-}1}+\ldots+b_i2^j+\ldots+b_12+b_0$, де a_i, b_j=0 або 1. Для знаходження результату їх множення за модулем p потрібно побудувати матрицю, представлену в таблиці 2.4, де $c_{ij}=2^{i+j} \bmod p$. Тоді добуток чисел a та b за модулем p отримувався згідно такої формули:

$$a \cdot b \bmod p = (\sum_{i=0}^{n_0-1}(\sum_{j=1}^{n_0-1} a_i b_j)c_{ij}) \bmod p = (\sum_{i=1}^{n_0-1}(\sum_{j=1}^{n_0-1} a_i b_j)2^{i+j} \bmod p) \bmod p \quad (2.18)$$

Це означає, що шуканий результат отримується у вигляді суми за модулем p тих c_{ij}, для яких відповідні a_i та b_j дорівнюють 1.

Таблиця 2.4 – Матриця для модулярного множення

	b_{n_0-1}	…	b_j	…	b_1	b_0
a_{n_0-1}	$c_{n_0-1\ n_0-1}$	…	$c_{n_0-1\ j}$	…	$c_{n_0-1\ 1}$	$c_{n_0-1\ 0}$
…	…	…	…	…	…	…
a_i	$c_{i\ n_0-1}$	…	c_{ij}	…	c_{i1}	c_{i0}
…	…	…	…	…	…	…
a_1	$c_{1\ n_0-1}$	…	c_{1j}	…	c_{11}	c_{10}
a_0	$c_{0\ n_0-1}$	…	c_{0j}	…	c_{01}	c_{00}

Слід зазначити, що $c_{ij} = c_{ji}$ і аналогічно до попереднього випадку кожне наступне значення c_{ij} визначається за допомогою рекурентних співідношень:

$$a_{ij+1} = \{2 \cdot c_{ij}, 2 \cdot c_{ij} < p; a_{i+1\,j} = \{2 \cdot c_{ij}, 2 \cdot c_{ij} < p \qquad (2.19)$$

Враховуючи, що $25_{10}=11001_2$ та $21_{10}=10101_2$ і побудувавши відповідну матрицю, можна побачити, що 25·21 mod 29=(24+12+16+6+3+4+16+8+1) mod 29= 90 mod 29=3.

Таблиця 2.5 – Матриця для модулярного множення 25*21 mod 29

25 / 21	1	1	0	0	1
1	24	12	6	3	16
0	12	6	3	16	8
1	6	3	16	8	4
0	3	16	8	4	2
1	16	8	4	2	1

Даний метод дозволяє замінити операцію множення, яка має квадратичну обчислювальну складність $O1(n_0) = n_0^2$, матрично-модульною операцією сумування, яка характеризується лінійно-логарифмічною складністю $O2(n_0)=n_0 \log n_0$

Для удосконалення даного методу можна побудувати матрицю, представлену в таблиці 2.6, де $a_{1j}=2^j \bmod p$.

Таблиця 2.6 – Матриця для удосконаленого матричного методу модулярного множення

	$a_{1\,2n_0-1}$	…	$a_{1\,n_0+1}$	$a_{1\,n_0}$	…	$a_{1\,i}$	…	a_{11}	a_{10}
b_{n_0-1}	a_{n_0}	…	a_1	a_0	…	…	…	0	0
…	…	…	…	…	…	…	…	0	0
b_j	…	…	…	…	…	…	…	0	0
…	0	…	…	…	…	…	…	0	0
b_1	0	…	a_{n_0}	a_{n_0-1}	…	a_{i-1}	…	a_0	0
b_0	0	…	0	a_{n_0}	…	a_i	…	a_1	a_0

Множення відбувається таким чином:

$$a \cdot b \bmod p = a_0 b_0 a_{10} + (a_0 b_1 + a_1 b_0) a_{11} + (a_0 b_2 + a_1 b_1 + a_2 b_0) a_{12} + \ldots + \\ + (a_0 b_{n_0-1} + a_1 b_{n_0-2} + \ldots + a_{n_0-1} b_0) \cdot a_{1\cdot 2 n_0-1} \quad (2.20)$$

Введемо позначення:

$$A_l = (a_0 b_{n_0-1} + a_1 b_{n_0-2} + \ldots + a_{n_0-1} b_0) a_{1l} \quad (2.21)$$

Тоді

$$a \cdot b \bmod p = (\sum_{l}^{2n_0-1} A_l) \bmod p \quad (2.22)$$

З врахуванням (2.3) співвідношення (2.4) набуде такого вигляду:

$$a \cdot b \bmod p = (\sum_{l}^{2n_0-1} A_l \mid n_0 - 1 b_j) a_{1l} \bmod p \quad (2.23)$$

де $i+j=n_0-1$.

Причому, якщо b_j=1 та деякі a_s і a_q=1, то можна спростити обчислення за допомогою наступного співвідношення:

$$(a_s+a_q)a_{1i}= a_{1\,i+1}. \qquad (2.24)$$

В таблиці 2.7 наведено приклад вдосконаленого матричного методу для пошуку значення виразу 21·25 mod 29. Знову числа 21 і 25 представляються в двійковій системі числення і на основі таблиці 2.5 будується відповідна матриця. З урахуванням співвідношень (2.23) та (2.24) одержуємо

$$(24+12+6+3+16+16+8+4+1)\text{mod}29=(19+8+4+1)\ \text{mod}29=3. \qquad (2.25)$$

Таблиця 2.7 – Приклад множення 21·25 mod 29

	19	24	12	6	3	16	8	4	2	1
1	0	1	0	1	0	1	0	0	0	0
1	0	0	1	0	1	0	1	0	0	0
0	0	0	0	1	0	1	0	1	0	0
0	0	0	0	0	1	0	1	0	1	0
1	0	0	0	0	0	1	0	1	0	1

Таким чином, отримано удосконалений новий метод заміни операції множення, яка має квадратичну обчислювальну складність $O1\cdot(n_0)=n_0^2$, матричномодульною операцією сумування з лінійно-логарифмічною складністю $O3=\frac{1}{2}n_0\log_2 n_0$

Отже, при використанні виразу (2.25) обчислювальна складність виконання операції модулярного множення суттєво зменшується, що дозволяє ефективно використовувати запропонований метод в алгоритмах захисту інформаційних потоків, побудованих на асиметричних криптосистемах.

Для зменшення об’єму пам’яті, в якій мають зберігатися проміжні результати матричних обчислень, можна використати векторно-модульний метод модулярного множення чисел $a\cdot=\sum_{i=0}^{n_0-1}a_i\cdot 2^i$ та $b\cdot=\sum_{j=0}^{n_0-1}b_i\cdot 2^j$. В цьому випадку

будується два вектор-рядки (c_i та a_i), перший з яких складається з елементів $c = 2 \cdot c_{i-1} \bmod p, c = 2^0 \cdot b \bmod p$ другий - з a_i (таблиця 2.8).

Таблиця 2.8– Представлення вектор-рядків модульного множення

i	n_0-1	…	2	1	0
c	c	…	c	c	c
a_i	a_{n0}-1	…	a_2	a_1	a_0

Слід зазначити, що кожне наступне значення c_i обчислюється за рекурентною формулою, аналогічною:

$$c_{i+1} = \{2 \cdot c_i, 2 \cdot c_i < p \tag{2.26}$$

Результат модулярного множення двох чисел отримується згідно такої формули:

$$a \cdot b \bmod p = (\sum_{i=0}^{n_0-1} a_i \cdot c_i) \bmod p \tag{2.27}$$

тобто відбувається сумування тих c_i, для яких відповідні a_i дорівнюють 1.

В таблиці 2.6 наведено приклад векторно-модульного методу модулярного множення для пошуку значення виразу 21·25 mod 29. Число 21 представляється в двійковій системі числення: $21_{10}=10101_2$ і на основі таблиці 2.9 будується відповідна матриця.

Таблиця 2.9 - Приклад множення 21·25 mod 29 векторно-модульним методом

i	4	3	2	1	0
a_i	1	0	1	0	1
c_i	2*26mod29=23	2*13mod29=26	2*21mod29=13	2*25mod29=21	2^0*25mod29=25

Отже, 21·25 mod 29=(23+13+25) mod 29=3.

Розроблений метод характеризується меншою часовою та апаратною складністю порівняно з матрично-модульним за рахунок зменшення кількості операцій додавання з $2 \cdot \log_2 n_2$ до $\log_2 n_0$ тобто в два рази. При вирішенні задач криптографії збільшення швидкодії в два рази є суттєвим і значно розширює функціональні можливості апаратного забезпечення, а також спрощує реалізацію відповідних спецпроцесорів.

Для модулярного експоненіювання $a^x \bmod p$ (вважається що $x \le \varphi(p)$, $\varphi(p)$ – значеня функції Ейлера від модуля p) потрібно використати проміжну матрицю, представлену в таблиці 2.10. Її розмірність дорівнює розрядності n_0 модуля p.

В стовбцях матриці записані величини $A_i = a^{2^i} \bmod p$ у двійковій системі числення, тобто a_{ij}=0, 1. Тоді будь–який степінь числа a записується за степенями двійки і шуканий результат можна отримати, перемноживши значення у стовбцях, для яких відповідні x_i у розкладі $x = \sum_{i=0}^{n_0-1} x_i \cdot 2^i$ дорівнюють одиниці, за допомогою такого виразу:

$$a^x \bmod p = \left(\prod_{i=0}^{n_0-1} a^{x_i 2^i}\right) \bmod p = \prod_{i=0}^{n_0-1} \left(\left(\sum_{j=0}^{n_0-1} a_{ij} 2^j\right)^{x_i}\right) \bmod p \qquad (2.18)$$

де a_{ij} – біти двійкового запису числа $a^{2i} \bmod p = \sum_{j=0}^{n_0-1} a_{ij} 2^j$

Таблиця 2.10 – Матриця піднесення до степеня

x_{n_0-1}	…	x_i	…	x_1	x_0
$a_{n_0-1\, n_0-1}$	…	$a_{i\, n_0-1}$	…	$a_{1\, n_0-1}$	$a_{0\, n_0-1}$
…	…	…	…	…	…
$a_{n_0-1\, j}$	…	$a_{i\, j}$	…	$a_{1\, j}$	$a_{0\, j}$
…	…	…	…	…	…
$a_{n_0-1\, 1}$	…	$a_{i\, 1}$	…	$a_{1\, 1}$	$a_{0\, 1}$
$a_{n0-1\, 0}$	…	$a_{i\, 0}$	…	$a_{1\, 0}$	$a_{0\, 0}$
$a^{2^{n_0-1}} \bmod p$	…	$a^{2^i} \bmod p$	…	$a^{2^1} \bmod p$	$a^{2^0} \bmod p$

Основними перевагами даного методу - це здійснення операцій над залишками, а не із великими числами, які дозволяють пришвидшити алгоритм модулярного експоненціювання і зменшить обчислювальну скоалдість. Слід зазначити, що для заповнення матриці доцільно скористатися таким рекурентним співвідношенням:

$$a^{2^{i+1}} \bmod p = (a^{2^i} \bmod p)^2 \bmod p \qquad (2.20)$$

В таблиці 2.11 наведено приклад пошуку значення $23^{19} \bmod 29$ на основі матричного методу модулярного експоненціювання.

Таблиця 2.11 – Приклад пошуку значення $23^{19} \bmod 29$ на основі матричного методу модулярного експоненціювання

1	0	0	1	1
19				
0	1	1	0	1
0	0	0	0	0
1	1	1	1	1
1	1	0	1	1
1	1	0	1	1
7	23	20	7	23
$23^{2^4} \bmod 29$	$23^{2^3} \bmod 29$	$23^{2^2} \bmod 29$	$23^{2^1} \bmod 29$	$23^{2^0} \bmod 29$

Отже, $23^{19} \bmod 29=(7\cdot7\cdot23) \bmod 29=25$. Операцію множення можна виконувати підходами, описаними в п. 2.2.

Для зменшення матриці, представленої у таблиці 2.12, доцільно використати векторно-модульний метод модулярного експоненціювання, у якому не визначаються біти двійкового запису чисел A_i, яке записується в десятковому вигляді. В таблиці 2.12 наведено приклад використання даного методу для пошуку $23^{19} \bmod 29$.

Таблиця 2.12 – Матриця векторно-модульного методу модулярного експоненціювання

i	n_0-1	…	2	1	0
x_i	x_{n0-1}	…	x_2	x_1	x_0
$a^{2^i} \mathrm{mod} p$	$a^{2^{n_0-1}} \mathrm{mod} p$	…	$a^{2^2} \mathrm{mod} p$	$a^{2^1} \mathrm{mod} p$	$a^{2^0} \mathrm{mod} p$
A_i	A_{n0-1}	…	A_2	A_1	A_0

Таблиця 2.13 – Приклад пошуку 23^{19} mod 29 векторно-модульним підходом

i	4	3	2	1	0
x_i	1	0	0	1	1
$a^{2^i} \mathrm{mod} p$	$a^{2^4} \mathrm{mod} 29$	$23^{2^3} \mathrm{mod} 29$	$23^{2^2} \mathrm{mod} 29$	$23^{2^1} \mathrm{mod} 29$	$23^{2^0} \mathrm{mod} 29$
A_i	7	23	20	7	23

Звідси слідує – 23^{19} mod 29= (7·7·23) mod 29=25.

Розрахунки демонструють, що запропонований алгоритм піднесення до степеня двійкового числа будь-якої розрядності за модулем p дозволяє зменшувати складність з $O(n_0^3)$, або $O1\left(n_0^2 \log_2 n_0\right)$ (Монтгомері метод) до $O2\left(\frac{n_0^2 \log_2 n_0}{4}\right)$, тобто ефективність зростає в 4 рази.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ КРИПТОЛАЙТ

3.1. Опис структури та інтерфесу

Програмна система Криптолайт є однію із напростіших у користуванні програмних системи. Доступний і зрозумілий інтерфейс дасть можливість користувачу який є обізнаним із системами і правилами криптосистем здатний швидко опанувати дану програму.

Програма реалізована на мові програмування Java, очікування, що до роботи на різних платформах справджується, всього лиш потрібно мати наявним встановлена Java Virtual Machine. Приклад інтерфейсу програмної системи наведено на рисунку 3.1.

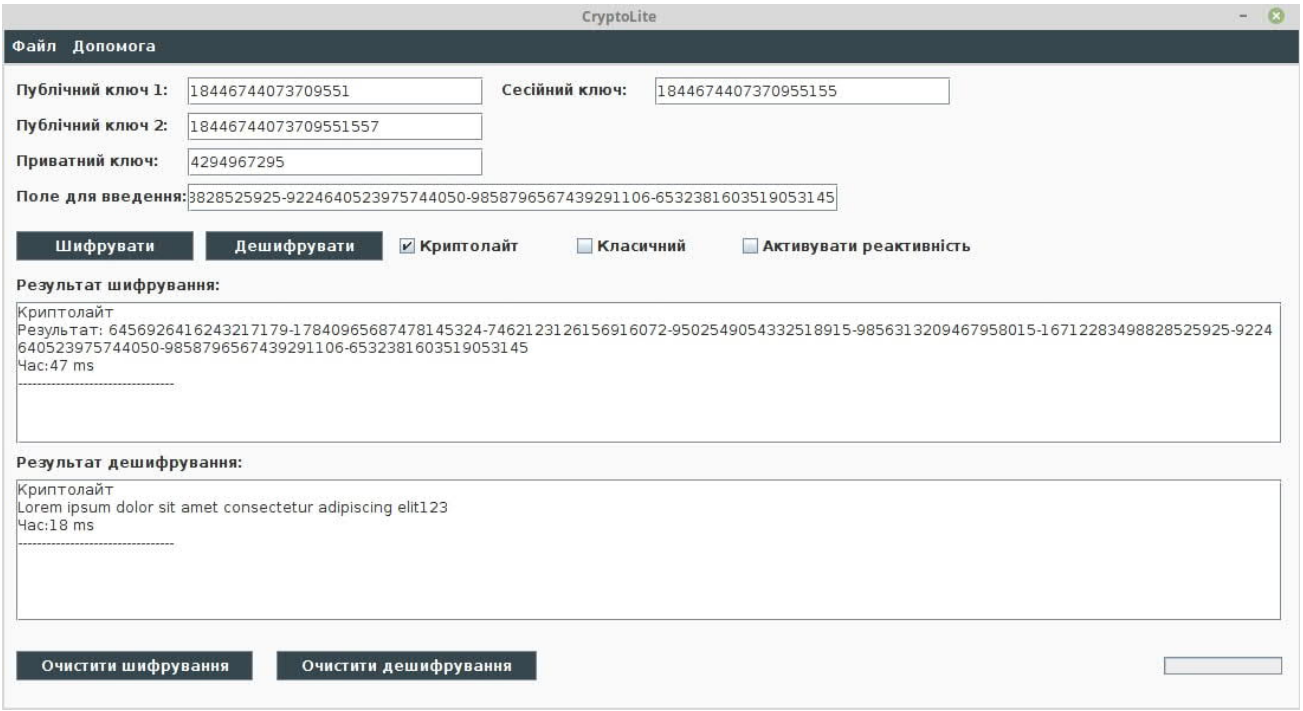

Рисунок 3.1 – Інтерфейс програми

На інтерфейсі є наявні поля введення двох публічних ключів, приватного ключа, та сесійного ключа, та поля введення текстового повідомлення, яке буде зашифровано. Приклад полів для введення зображено на рисунку 3.2

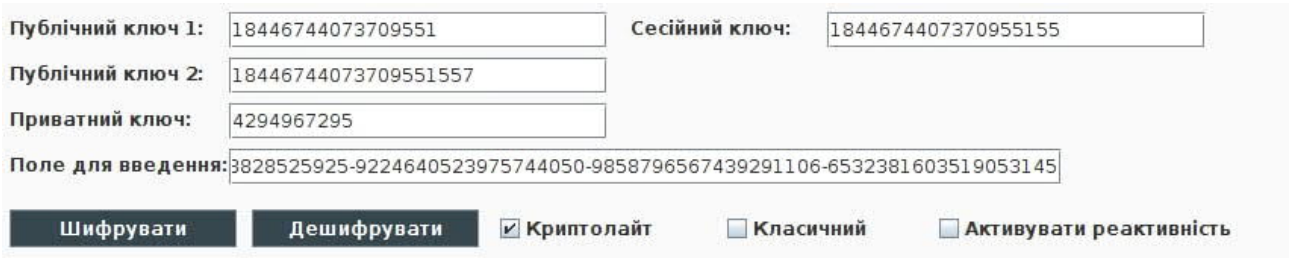

Рисунок 3.2 –Поля введення елементів для організації шифрування

На інтерфейсі також можна помітити наявні кнопки для організації шифрування, нажавши на кнопку “Encrypt”, та дешифрування “Decrypt”. Також є можливість налаштувати запускання трьох способів обробки і організації шифрування. Наприклад користувач може обрати класичну криптосистему для обробки даних, також можна увімкнути вдосконалений алгоритм для асиметричної криптосистеми виконувати обробку. Для покращення і пришвидшення обробки можна увімкнути реактивність обробоки. Реактивність має надати кращу швидкодію обробки і розвантажити обчислювальну складність між потоками. Виконання роботи шифрування та дешифрування можна спостерігати у панелі виведення, як зображено на рисунку 3.3.

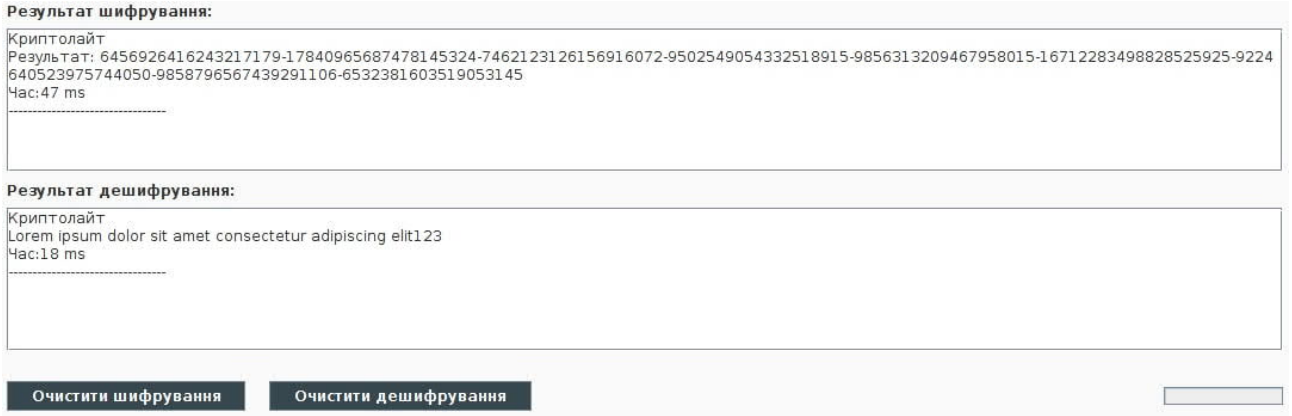

Рисунок 3.3 – Фрагмент інтерфейсу із полями виведення

На рисунку 3.3 зображено також кнопки для очищення полів виведення, щоб користувачу, ну або досліднику, можна було очистити для зручності і продовжувати роботу.

Програмна система здатна експортувати та зберігати дані обчислень у файл із розширенням *excel* внезалежності від платформи.

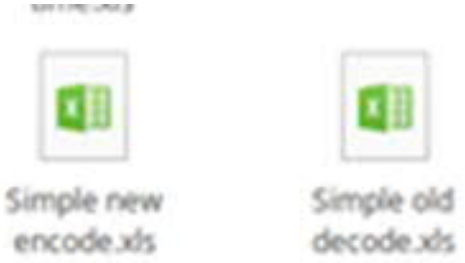

Рисунок 3.3 – Результати обробки даних

3.2. Програмна реалізація алгоритму Криптолайт

Реалізація програмної системи, як вже було сказано, вибір припав на мову програмування Java, так як це є одна із популярних мов програмування, дає можливість реалізації рішень на різні операційні системи[34]. Тож реалізація алгоритму здійснено на основі даної мови програмування із способом архітектурної оптимізації обробки даних. Спершу нам потрібно реалізувати фрагмент, що буде виконувати обчислення модулярного експоненціювання за класичним способом. Для реалізації роботи із великим числами буде застосовано утиліту BigInteger, щоб чіткість виконання операційбула контрольована

Цей спосіб дозволить реалізувати для доступності правильності виконання алгоритму. Такий підхід дозволить переконатися у достовірності виконання обробки правильно.

Для реалізації повної демонстрації проблемних моментів класичного алгоритму рекомендовано глянути на додаток Б, де можна помітити, що реалізація складається із багатьох частинок корекціного коду, для полекшення і правильності обробки обчислення.

Реалізація класичного алгоритму ошифрування дани програмним чином виконоється наступним чином:

```
public static EncryptResult encrypt(BigInteger numberToCode,
BigInteger keyP, BigInteger keyQ, BigInteger keyX, BigInteger keyK)
{
BigInteger keyY = modPowFunction2(keyQ, keyX, keyP);
BigInteger keyA = modPowFunction2(keyQ, keyK, keyP);
BigInteger tempRez2 = modPowFunction2(keyY, keyK, keyP);
BigInteger keyB = tempRez2.multiply(numberToCode).mod(keyP);
return new EncryptResult(keyA, keyB);
}
```

Даний спосіб обробки є складним у реалізації, особливо є навантаження на апаратно-обчислювалну машину, так як в пам’яті повинні зберігатися значення із попередньої ітерації, та щей підняття до степення яке є не спрощеним і викликає досить велике навантаження на процесор і оперативну пам’ять. Тобто є вірогідність вийти за межі виділеної пам’яті і це не дає можливість користувачі задати одну із наймаксимальніших розрядностей ключа.

Так як ціллю є полегшення обчислювальної складності ми ще можемо якось посприяти і можливості внесення користувачем ключа великої розрядності, що нам може дозволити максимальне значення числа відповідно до розрядності системи. Наявність блоків програмного коду для виправлення обчислення створює додаткову часову затрату і навантаження на апаратний засіб, що є досить не порібним, особливо із збереженням результатів кожної ітерації та піднесенням до степення, який є надто великим чисом, що в процесі породжу занадто багато навантажень для процесора та ОЗП.

Рекомендовано знайти можливість спростити дану реалізацію, також класичний метод не дозволяє здійснити ручне розпаралелення обчислення ітерацій, тобто ми не можемо кожну ітерацію обчислити в паралельному потоці, тіки ітеративно запустивши цикл. Складність реалізації зображено на рисунку 3.4.

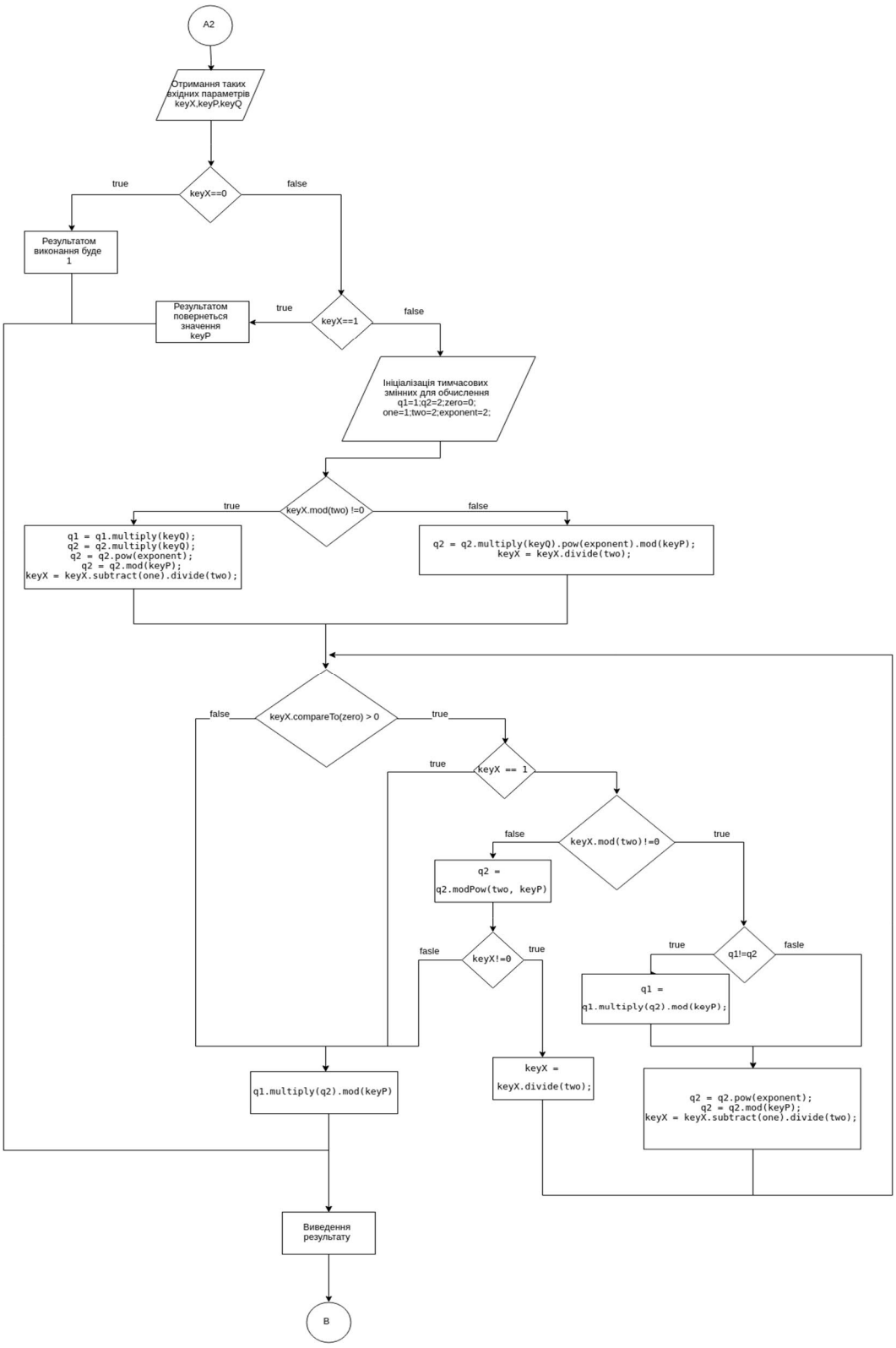

Рисунок 3.4 – UML діаграма класичного алгоритму

Наступним кроком до реалізації буде створення нового способу обробки, а саме векторно-модулярного експоненціювання, як зображено на рисунку 3.6.

```
public static EncryptResult encrypt(BigInteger numberToCode,
BigInteger keyP, BigInteger keyQ, BigInteger keyX, BigInteger
keyK) throws ArithmeticException {
        String decimalKeyP = keyP.toString(2);
        String decimalKeyX = keyX.toString(2);
        String decimalKeyK = keyK.toString(2);
        char[] decimalKeyXRadixArray = radixCharArray(decimalKeyX,
decimalKeyP.length());
        List<BigInteger> iterationOneY =
firstPart(decimalKeyXRadixArray, keyP, keyQ);
        BigInteger keyY = new BigInteger("1");
        for (BigInteger item : iterationOneY
        ) { keyY = keyY.multiply(item); }
        keyY = keyY.mod(keyP);
        char[] decimalKeyKRadixArray = radixCharArray(decimalKeyK,
decimalKeyP.length());
        List<BigInteger> iterationOnB =
firstPart(decimalKeyKRadixArray, keyP, keyY);
        BigInteger keyB = new BigInteger("1");
        for (BigInteger item : iterationOnB) { keyB =
keyB.multiply(item); }
        keyB = keyB.mod(keyP).multiply(numberToCode).mod(keyP);
        List<BigInteger> iterationOneA =
firstPart(decimalKeyKRadixArray, keyP, keyQ);
        BigInteger keyA = new BigInteger("1");
        for (BigInteger item : iterationOneA
        ) { keyA = keyA.multiply(item); }
        keyA = keyA.mod(keyP);
        return new EncryptResult(keyA,keyB);
    }
```

Аналізуючи реалізацію векторно-модулярного способу, ми можемо зрозуміти, відсутні корекційні дані для покращення правильності виконання обчислення. Беручи повну версію роботи всієї функції із Додатку Б можна зрозуміти, що в даній реалізації також відсутнє піднесення до ітеративного піднесення до степення із пониженням його, натомість ми підносимо в кожній ітрації до степеня 2, що дозволяє розвантажити і при цьому, пришвидшити обчислення, та зберігати тільки потрібні значення відповідно до правил пунку. Складність реалізації роботи алгоритму відображено на рисунку 3.5.

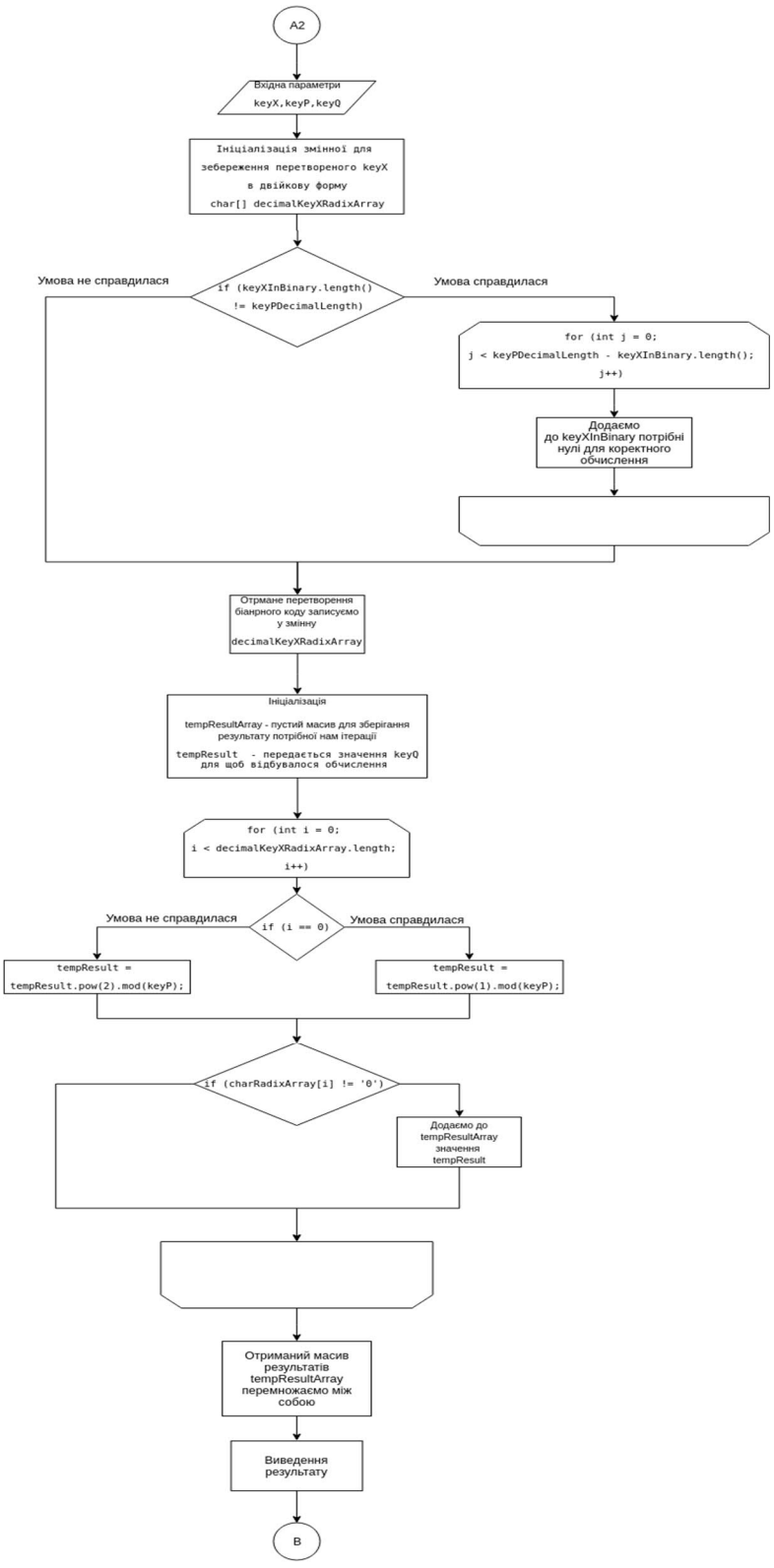

Рисунок 3.5 – UML діаграма запропонованого алгоритму

Щоб програмна система даних частинок обробки дозволила обійти роботу збирача інформаційного мусору та пам’ять після обробок звільнялася вже після обробки даних, застосовано підхід збереження даних першим кроком, а

вивільнення пам’яті і очистка вже невикористаної як другий крок. Тобто ми спочатку зберегли всі результати очислення і вже потім очищаємо від непотрібних змінних.

Розробка і корекція побудови шифротексту та розбивання його на блоки, дозволяє розбити загальне повідомлення на частини які можна зашифрувати та зформувати вихідний шифротекст. Для цього ми робимо залежність від розміру ключа та визначаємо на скільки повинні ми зпростити і поділити саме повідомлення. Отже наступна послідовність організації блоків:

1) перетворюємо повідомлення на набір байтів;
2) визначаємо на скільки ми будемо ділити набір байтів на блоки;
3) розділяємо блоки на конкретні дані;
4) додаємо корективу на останій блок.

Після отримання масиву блоків байт частинок тексту, ми маємо можливість продовжити обробку шифрування, зашифровуючи блоки і це дасть надійність для вихідного шифротектсу бути не зрозумілим.

Перетворення вже попередньо розшифрованих блоків назад у стрічку працює за наступною послідовністю:

1) отримується блок і підкореговує його у відповідності до розміру блоку який задається попередньо;
2) отриманий під корегований блок додаємо до буфера у правильній послідовності;
3) корегуємо останій блок;
4) формуємо набір байтів із буфера у один набір байтів;
5) конвертуємо набір байтів у вихідне повідомлення

Програмна реалізація вище зазначеної логіки розбивання повідомлення на блоки має вигляд наступний:

```
public static String[] splitMessage(String message, int
blocksSize) {
        String messageInBytes = new
BigInteger(message.getBytes()).toString();
        List<String> blocks = new ArrayList<>();
```

```
        int indexStart = 0;
        int indexEnd = blocksSize - BLOCK_SIZE_CORRECTOR;
        int messageLength = messageInBytes.length();
        do {
            if (indexEnd <= messageLength)
{ blocks.add(messageInBytes.substring(indexStart, indexEnd));
            } else
{ blocks.add(messageInBytes.substring(indexStart,
messageLength)); }
            indexStart = indexEnd;
            indexEnd = indexEnd + blocksSize -
BLOCK_SIZE_CORRECTOR;

        } while (indexEnd <= messageLength);
        if (indexStart < messageLength) {
            String lastBlock =
messageInBytes.substring(indexStart, messageLength);
            if
(lastBlock.startsWith(LAST_BLOC_ORIGINAL_REPLACE_HELPER)) {
                lastBlock =
lastBlock.replaceFirst(LAST_BLOC_ORIGINAL_REPLACE_HELPER,
LAST_BLOC_REPLACE_HELPER);
            }
            blocks.add(lastBlock);
        }
        return blocks.toArray(new String[0]);
    }
```

Останім кроком буде реалізація алгоритму шифрування на основі асиметричної криптосистеми та із врахуванням шифрування блоків. Тобто ми повинні отримати на виході зашифроване повідомлення, яке буде складатися із зашифрованих блоків. Приклад програмної реалізації алгоритму криптолайт, який і здійснює основне шифрування:

```
public static String encrypt(String message, BigInteger keyP,
BigInteger keyQ, BigInteger keyX, BigInteger keyK) throws
ArithmeticException {
        List<EncryptResult> encryptResults = new ArrayList<>();
        String[] blocks = BytesBlockSplitter.splitMessage(message,
keyP.toString().length());
        for (String messageInByte : blocks) {
            encryptResults.add(NewCoder.encrypt(new
BigInteger(messageInByte), keyP, keyQ, keyX, keyK));
        }
        StringBuilder encryptMessageHolder = new StringBuilder();
```

```
encryptMessageHolder.append(encryptResults.get(KEY_A_INDEX).getKey
A());
        encryptMessageHolder.append(SEPARATOR);
        for (int i = 0; i < encryptResults.size(); i++) {

encryptMessageHolder.append(encryptResults.get(i).getKeyB());
            if (i < encryptResults.size() - 1) {
                encryptMessageHolder.append(SEPARATOR);
            }
        }
        return encryptMessageHolder.toString();
    }
```

Робота алгоритму реалізовано наступним чином:

1) отримуємо ключі та повідомлення для обробки;

2) використовуємо функцію розбивання повідомлення на блоки і задаємо умову по якому будуть розбиватися блоки, це залежність від одного із публічних ключів, а саме ключ який використовується як модуль;

3) зашифровуємо блоки та додаємо їх зашифровані частини в буфер.

4) формуємо вихідне повідомлення із дільником і коректною постановкою вихідних символів, для можливості правильного розшифрування.

5) виводимо вихідне повідомлення у програмну систему

Розшифрування виконується також із застосуванням даного алгоритму наступними кроками:

1) отримуємо шифротекст, публічний ключ та приватний ключ

2) ділимо шифротекст на блоки

3) рошфировуємо ці блоки і отримуємо частину байтів які є частиною основного повідомлення

4) застосовує конвертацію блоків у повідомлення із заданням розміру блоку у відповідності до публічного ключа, який є модулем для обчислення.

5) отримане розшифроване і зформоване повідомлення виводимо

Отже, врезультаті ми отримуємо алгоритм послідовності шифрування даних із застосуванням математичних і програмних підходів, взято до уваги особливості роботи мови програмування та застосовано гнучку систему

розробки класичного і пропонованого алгоритмів шифрування. Підчас розробки даних алгоритмів спостерігалися проблематики їх корекції і організування блокового поділу повідомлення.

Ефективність роботи програмної системи також залежить від правильно підібраного підходу реалізації, саме тому було обрано шаблон проектування MVP який дає змогу розділити логіку виконання від логіки графічного інтерфейсу. На рисунку 3.6 зображено архітектуру програмної системи.

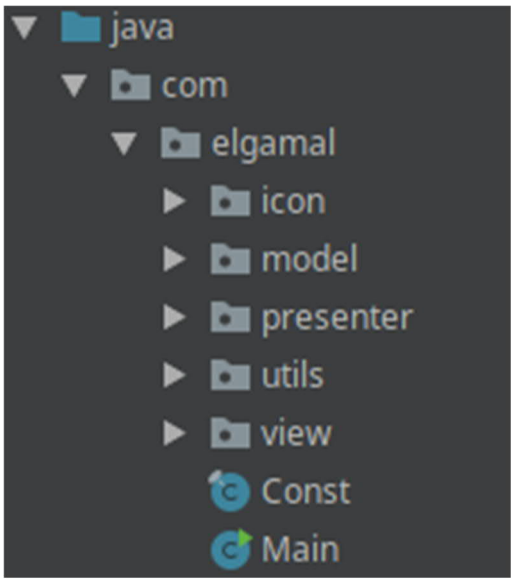

Рисунок 3.6 – Архітектурна структура прогрмної системи

Вдало підібрана архітектура дозволяє розширити можливості виконання програмної реалізації і додаванням нових компонентів та бібліотек для покращення обробки та обчислення. Саме це призвело до бажання наділити програмну систему ще додатковою можливість, а саме розпаралеленою обробкою даних. Ми маємо можливість розпаралелити по потоками обпилення. Так для розпаралелення застосовано бібліотеку RxJava[35].

Дана бібліотека дозволяє організувати блок виконання обробки, який буде розподілено системою самої бібліотеки у потоки. Інструмент має всобі реалізовані функції можливості розподілення обчислювальної складності у відповідності до набору потоків. Широкий сектр можливостей бібліотеки надає

нам інструмент вибору набору потоків де саме виконуватися обчисленнюю Рекомендовано для обчислень використовувати “computation”, який якраз призначений саме для математичних операцій.

Програмна реалізація організації реактивності має наступний вигляд:

```
Completable.fromRunnable(() ->
tempResultCash.add(CryptoLite.encrypt(numberToEncrypt, keyP, keyQ,
keyX, keyK)))
                .subscribeOn(Schedulers.computation())
                .observeOn(Schedulers.single())
                .subscribe(() -> {
mEncodeWithReactiveXResultList.add(new ModelForPrint(
                                keyQ.toString(),
                                keyP.toString(),
                                keyX.toString(),
                                keyK.toString(),
                                numberToEncrypt,
                                String.valueOf(totalTime))) }
```

Реалізація розпаралелення обчислення між потоками, дозволяє реалізувати швидкодію обробки даних, вданому випадку це пришвидшити виконання шифрування данх.

Щоб зберегти вихідні дані у програмній системі реалізовано структурне зберігання даних, у відповідності до обраних користувачем функцій здійснення шифрування даних. Нижче наведено приклад програмної реалізації структури зберігання даних у вигляді глобальних змінних, де в програмній системі зберігаються всі вихідні дані та дають можливість виконувати маніпуляцію ними:

```
private final List<ModelForPrint> mSimpleNewEncodeResultList;
private final List<ModelForPrint> mSimpleNewDecodeResultList;
private final List<ModelForPrint> mSimpleOldEncodeResultList;
private final List<ModelForPrint> mSimpleOldDecodeResultList;
private final List<ModelForPrint> mEncodeWithReactiveXResultList;
private final List<ModelForPrint> mDecodeWithReactiveXResultList;
private final List<ModelForPrint>
mNewEncodeWithPowerAndStepsResultList;
private final List<ModelForPrint>
mNewDecodeWithPowerAndPowerResultList;
private final List<ModelForPrint> mSimpleNewEncodePartResultList;
private final List<ModelForPrint> mSimpleOldEncodePartResultList;
```

```
private final List<ModelForPrint>
mEncodeWithReactiveXPartResultList;
private final List<ModelForPrint>
mNewEncodeWithPowerAndStepsPartResultList;
```

Наступним кроком реалізована можливість конвертації структури у вихідний файл із відповідним найменування, що відповідає найменування обраної функції обробки. Так програмну реалізцію фрагмент із зберіганням структури у вихідний файл із выдповідним найменуванням:

```
ExcelUtil.saveEncodeDataAtExcelHSSFile("Simple new encode",
Const.ENCODE,
mSimpleNewEncodeResultList)
```

Також користувачу дана можливість почистити закешовані значення які накопичувалися підчас роботи із програмою, саме тому була реалізована наступна програмна функція для очищення структури із збереженими даними:

```
mSimpleNewDecodeResultList.clear()
```

Повний код реалізації даної роботи структур наведено у Додатку Б.

Щоб комфортно здійснити реалізацію конвертації самої структури у вихідний файл було застосовано бібліотеку Apach POI, аби скориставшись зручним інструментом даної бібліотеки записати вихідні дані у excel файл із відповідним найменуванням.

Бібліотека Apach POI розповсюджується безплатно згідно ліцензії Apach, та є доступною на репозиторіях для користування і роботи із EXCEL.

Приклад програмної реалізації фунції, що дозволяє гнучко описати запис структури у файл. Функція дозволяє гнучко описати можливість запису структури у відповідні колонки із найменуваннями:

```
public static boolean saveEncodeDataAtExcelHSSFile(String
fileName, String sheetName, List<ModelForPrint> result) {
        int resultSize = result.size();
        HSSFWorkbook hssfWorkbook = new HSSFWorkbook();
        HSSFSheet sheet = hssfWorkbook.createSheet(sheetName);
```

```
        HSSFRow titleRow = sheet.createRow(0);
        titleRow.createCell(0).setCellValue(Q);
        titleRow.createCell(1).setCellValue(P);
        titleRow.createCell(2).setCellValue(X);
        titleRow.createCell(3).setCellValue(K);
        titleRow.createCell(4).setCellValue(MESSAGE);
        titleRow.createCell(5).setCellValue(TIME);
        try { for (int i = 0; i < resultSize; i++) {
                int rowIteration = 1 + i;
                HSSFRow row = sheet.createRow(rowIteration);
                ModelForPrint modelForPrint = result.get(i);

row.createCell(0).setCellValue(modelForPrint.getQ());

row.createCell(1).setCellValue(modelForPrint.getP());

row.createCell(2).setCellValue(modelForPrint.getX());

row.createCell(3).setCellValue(modelForPrint.getK());

row.createCell(4).setCellValue(modelForPrint.getMessage());

row.createCell(5).setCellValue(modelForPrint.getTime());}
            hssfWorkbook.write(new FileOutputStream(fileName +
".xls"));
            hssfWorkbook.close();
            return true;
        } catch (IOException e) {e.printStackTrace();return
false;}}
```

3.3. Тестування та порівняння із аналогами

Найкращою перевіркою дієздатності програмної системи та розробленого алгоритму потрібно провести тестування на рівних умовах і вивести аналітичні дані, порівнюватися також буде класичний та пропонований алгоритм.

Беручи до уваги наведених аналогів у пункті 1.3 проаналізуємо програмні системи. Одним із недоліків всіх наведених вище програмних систем є проблема, а саме неможливість працювати на багатьох платформах тай мають багато залежностей від однієї платформи як Windows. Отже, для тестування нам потрібно створити однакові умови тоді запускатимемо програми на ОС Windows.

Оберемо простий текстовий формат для шифрування де він буде аналогічний кожному, задамо ключі шифрування із розрядністю 1024, так як не всі програмні системи підтримують широкий діапазон розрядності ключів. Дляпорівнювання було борано дві програмні системи, одну із найкращих це Kleopatra та найпростішу у використанні, що є розроблена добровольцями GATE.

Результати роботи часових характеристик шифрування даних наведено на рисунку 3.7.

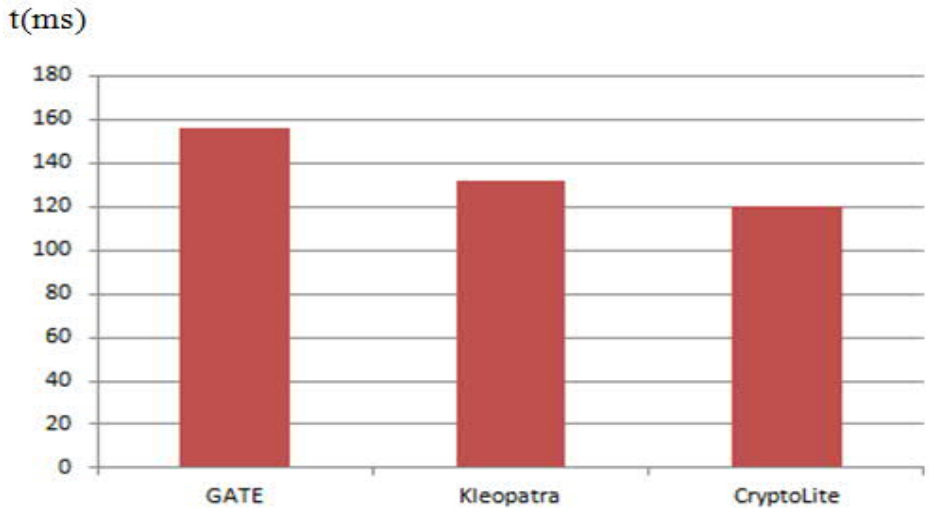

Рисунок 3.7 – часова характеристика роботи програмних систем

Аналізуючи часові вихідні дані, можемо зробити висновок, що швидкодія програмних систем має не велику різницю, проте пропонований алгоритм обробки даних для шифрування дає прискорення, облегшує обчислювальну складність, що має вплив на часові показники.

Наступним кроком перевірки нам потрібно протестувати та порівняти дієздатності класичного та пропонованого алгоритму. Для цього ми повинні задати їм однакові умови, тобто ми обираємо числа із розрядність від 32 до 54.

Виконаємо обчислення із навантаженням, де показник розрядності змінюється за правилом 2^k-1

Вхідні дані:

1) Публічний ключ 1 = 18446744073709551

2) Публічний ключ 2 =18446744073709551557

3) Приватний ключ=2^k-1, k – розрядність числа

4) Повідомлення “Lorem ipsum dolor sit amet consectetur adipiscing elit sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.”

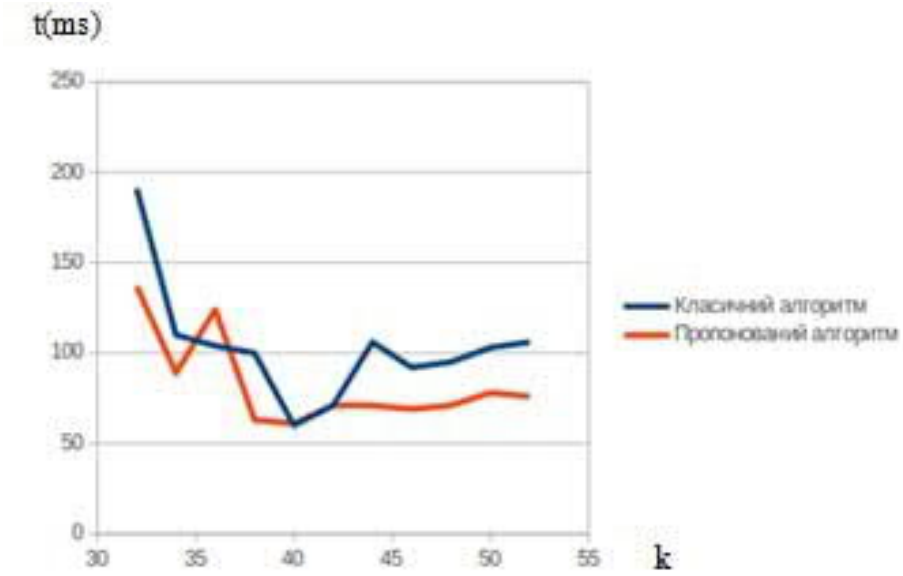

Рисунок 3.8 – Часові характеристики обрахунку класичним та запропонованим алгоритмами, де x=2^k-1

Як можемо спостерігати із часових характеристик вихідних даних, рисунку 3.8, при збільшенні розрядів для приватного ключа, швидкодія пропонованого алгоритму має перевагу над класичним. Хоча при малих розрядах алгоритми не поступаються один одному.

Виконаємо обчислення із навантаженням, де показник розрядності змінюється за правилом 2^k+1.

Вхідні дані:

1) Публічний ключ 1 = 18446744073709551

2) Публічний ключ 2 =18446744073709551557

3) Приватний ключ =2^k+1, k – розрядність числа

4) Повідомлення “ Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Ut enim ad minim veniam quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodoю Lorem ipsum dolor sit amet consectetur adipiscing elit sed do eiusmod tempor incididunt ut labore et dolore magna aliqua consequat.”

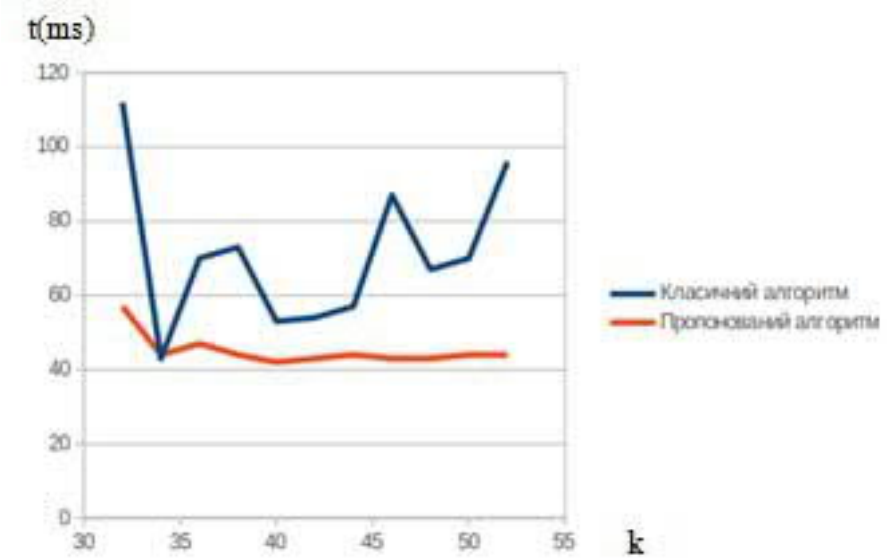

Рисунок 3.9 – Часові характеристики обрахунку класичним та запропонованим алгоритмами, де x=2^k+1

Аналізуючи вихідну часову характеристику із рисунку 3.10 можна зрозуміти, що при збільшенні розрядності для ключа, час затрачений на обробку в класичного метода зростає, а в пропонованого має практично стабільний час внезалежності від розрядності ключа.

Виконаємо обчислення із навантаженням, де показник розрядності змінюється за правилом (k/2) +1.

Вхідні дані:

1) Публічний ключ 1 = 18446744073709551

2) Публічний ключ 2 =18446744073709551557

3) Приватний ключ = (*k*/2) +1, *k* – розрядність числа

4) Повідомлення “ Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Ut enim ad minim veniam quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodoю Lorem ipsum dolor sit amet consectetur adipiscing elit sed do eiusmod tempor incididunt ut labore et dolore magna aliqua consequat.

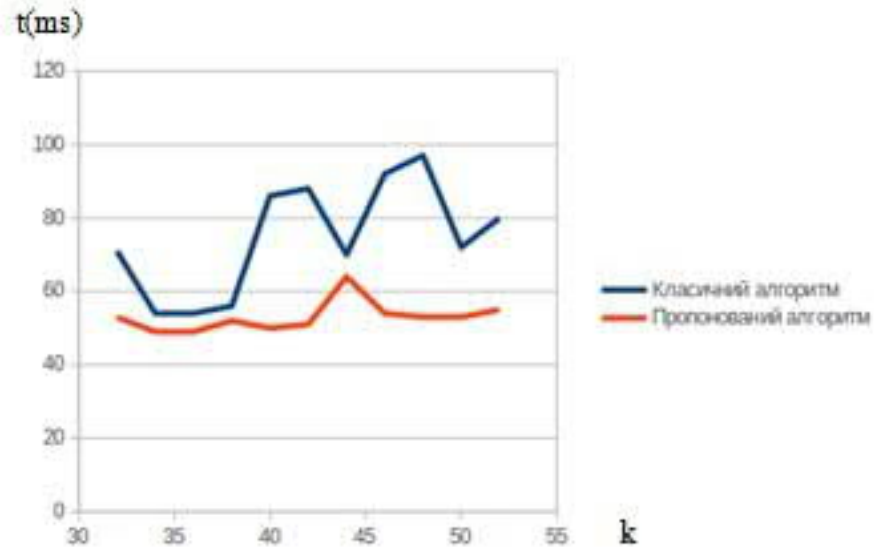

Рисунок 3.9 – Часові характеристики обрахунку класичним та запропонованим алгоритмами, де *x=(k/2)* +1

Вихідні дані із часової діаграми, що зображено на рисунку 3.12. бачимо, що перевага пропонованого алгоритму в часових показниках є досить значною у порівняні із класичним.

Виконаємо обчислення із навантаженням, де показник розрядності змінюється за правилом (*k*/2) +1.

Вхідні дані:

1) Публічний ключ 1 = 18446744073709551

2) Публічний ключ 2 =18446744073709551557

3) Приватний ключ = Вага Хемінга

4) Повідомлення “ Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Ut enim ad minim veniam quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodoю Lorem ipsum dolor sit amet consectetur adipiscing elit sed do eiusmod tempor incididunt ut labore et dolore magna aliqua consequat. ”

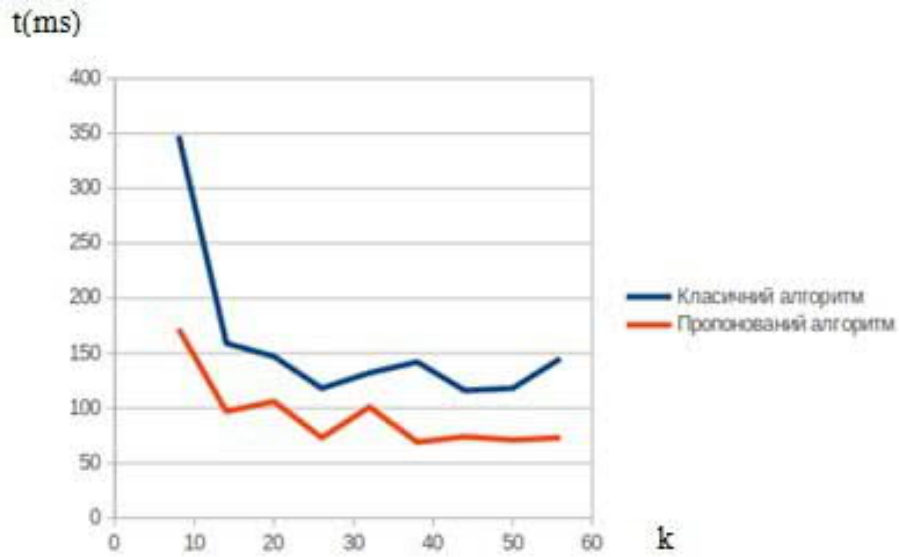

Рисунок 3.10 – Часова характеристика обрахунку класичним та пропоновани алгоритмами.

Аналізуючи вихідні часовіхарактеристики із рисунку 3.12 можна зробити висновок, що пропонований алгоритм має значну перевігу у виконанні перед класичним.

Отже, провівши ряд тестувань, можна виявити, що перевагами програмної системи основаної на пропонованому алгоритмі криптолайт є можливість здійснювати обробку внезалежності до операційної системи, дієздатність роботи пропонованого алгоритму, тки впливає на часові характеристики, хоча є і не точності із проблематикою реалізаціїї поділу повідомлення на блоків. Також проведено дослідження дієздатності класичного алгоритму та пропонованого, а

саме складність їх реалізації, спосіб виведення даних та часові характеристики. Пропонований алгоритм має значну перевагу над класичним при підвищенні розрядності ключа. Тким чином можна зрозуміти, що пропонований алгоритм має пропоновані математичні рішення для покращення обчислювальної здатності.

Ще унікальною річю підчас дослідження було виявлено можливість задвати набагато більшу розрядність ключу. Тобто при дослідженні аналогів, виявлено що максимальна розрядність ключа сягає 4096 бітів, що дає обмеження користувачу у виборі ключів, в той час програмна система основана на алгоритмі криптолайт дозволяє перекласти обмеження кількості вибору ключа на максимальну розрядність числа, що може надати операційна система.

ВИСНОВКИ

При виконання кваліфікаційної роботти отримано наступні результати:

1. Досліджено інформацію та її форми;
2. Проаналізовано засоби шифрування та дешифрування, було здійснено пошук аналогів та їх особливостей;
3. Проаналізовано та досліджено синхронні та асинхронні криптосистеми. Досліджено їхню роботу та математичні моделі;
4. На основі теоретичної бази пропонованого алгоритму дослідження математичних моделей оптимізації обчислювальної складності у асиметричних криптосистемах, було розроблено вдосконалений алгоритм на основі векторно-модулярного експоненціювання.
5. Здійснено реалізацію програмної системи на основі класичного та пропонованого алгоритму із застосуванням програмних оптимізаційних вдосконалень для покращення роботи.
6. Проведено тестування працездатності алгоритму та порівняння із програмними аналогами і класичним алгоритмом. Порівнювальна характеристика зображена у вигляді часових діаграм.
7. Дослідження показало, що пропонований алгоритм є набагато ефективнішим, як у розробці, так і у швидкодійності обчислення, на відмінну класичного, що потребую нагромадженої кількості додаткових перевірок, для корегування обчислення, що також має і вплив на його швидкодію. При порівнюванні із програмами аналогами, вдосконалений алгоритм показав результат як ефективний, проте різниця у швидкої не є досить значною.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Суслін В.В. Алгоритм шифрування даних на основі модулярного експоненціонування / Суслін В.В. Батько Ю.М. // III Науково-практична конференція молодих вчених і студентів «Інтелектуальні комп'ютерні системи та мережі» ЗУНУ. Тернопіль, 2020. 11.с
2. Суслін В.В. Програмна система шифрування даних на основі алгоритму криптолайт/ Суслін В.В. Батько Ю.М. // III Науково-практична конференція молодих вчених і студентів «Інтелектуальні комп'ютерні системи та мережі» ЗУНУ. Тернопіль, 2020. 12.с
3. Інформація [Електронний ресурс] – Режим доступу до ресурсу: https://uk.m.wikipedia.org/wiki/Інформація
4. Сінельнікова Т. Ф. Поняття інформатики [Електронний ресурс] / Т. Ф. Сінельнікова – Режим доступу до ресурсу: https://dl.nure.ua/pluginfile.php/468 /mod_resource/content/3/content/content2.html
5. Романец Ю.В., Тимофеев П.А., Шаньгин В.Ф. Защита информации в компьютерных системах и сетях. М.: Радио и связь, 2001. 376 с.
6. Воробьев И.В. Защита информации в персональных ЭВМ. М.: Мир, 2008. 312 с.
7. Мафтик С. Механизмы защиты в компьютерных сетях. М.: Мир, 2013. 219 с.
8. Ростовцев А.Г., Маховенко Е.Б. Теоретическая криптография. М.: Профессионал, 2005. 480 с.
9. Рябко Б.Я., Фионов С.В. Криптографические методы защиты информации: Учебное пособие для вузов. М.: Телеком, 2005. 229 с.
10. Панасенко С.П. Алгоритмы шифрования. М.: Академический Проект, 2006. 529 с.

11. Ella D. L. Шифрування та Дешифрування [Електронний ресурс] / Deon Lackey Ella. – 2012. – Режим доступу до ресурсу: https://developer.mozilla.org/uk/docs/Archive/Security/Шифрування_та_Дешифрування.

12. Петров В.П., Петров С.В. Информационная безопасность человека и общества. М.: ЭНАС, 2007. 334 с.

13. Бранстед Д.К., Смид М.Э., Стандарт шифрования данных: прошлое и будущее. М.: МИИЭР, 2008. 513 с.

14. Шнаер Б. Практическая криптография. М.: Вильямс, 2007. 424 с.

15. Задірака В.К., Олексюк О.С. Комп'ютерна криптологія. Тернопіль, Київ, 2002. 504 с.

16. Вербіцький О.В. Вступ до криптології. Львів: ВНТЛ, 1998. 247 с.

17. Криптология. М.:, Иванов И.И. Высшая школа, 2004. 230 с.

18. Винокуров А.Ю. Блочные алгоритмы. М.: Монитор, 2005. 132 с.

19. Жигульская Г.М., Голубицкая Е.А. Экономика связи. М.: Радио и связь, 2003. 318 с.

20. Айерленд К. Классическое введение в современную теорию чисел. М.: Мир, 2007. 416 с.

21. Акушский И.Я., Юдицкий Д.М. Машинная арифметика в остаточных классах. М.: Сов. радио, 1968. 440с.

22. Вариченко Л.В. Абстрактные алгебраические системы и цифровая обработка сигналов. К.: Наука думка, 2006. 247 с.

23. Сидельников В.М. Криптография и теория кодирования. М.: ФИЗМАТЛИТ, 2008. 324 с.

24. Амербаев В.М. Теоретические основы машинной арифметики. Алма-Ата: Наука, 2006. 324 с.

25. Ворона В.А., Тихонов В.А. Системы контроля и управления доступом. М.: Горячая линия - Телеком, 2010. 272 с.

26. Деднев М.А., Дыльнов Д.В. Защита информации в банковском деле и электронном бизнесе. М.: КУДИЦ ОБРАЗ, 2004. 321 с.

27. Казарин О.В. Безопасность программного обеспечения компьютерных систем. М.: Высшая школа, 2013. 243 с.

28. Ярочкин В.И. Информационная безопасность. М.: Академический Проект, 2008. 544 с.

29. Барсуков В.С. Безопасность: технологии, средства, услуги. М.: КУДИЦ-ОБРАЗ, 2001. 496 с.

30. Мещеряков Р.В., Белов Е.Б., Лось В.П., Основы информационной безопасности. М.: Горячая линия - Телеком, 2006. 544 с.

31. Краснобаев В.А. Методы повышения надежности специализированных ЭВМ систем и средств связи. Харьков: ХВВКИУ РВ, 2000. 172с.

32. Ковалевский В. Криптографические методы. М.:Компьютер Пресс, 2013. 312 с.

33. Севастьянова Б.А. Совершенные шифры. М.: Гелиос АРВ, 2003. 160 с.

34. Суслін В. В. Програмна підсистема модулярного експоненціювання для асиметричних криптосистем / The software subsystem of the modular exponentiation in asymmetric cryptosystems [Електронний ресурс] / Віталій Вікторович Суслін. – 2019. – Режим доступу до ресурсу: http://dspace.wunu.edu.ua/bitstream/316497/39213/1/Суслін ДП_2019.pdf.

35. Якименко І.З., Касянчук М.М., Кінах Я.І., Власюк І.М., Суслін В.В. Удосконалення реалізації асиметричних криптоалгоритмів на основі системи залишкових класів. М. VI Всеукраїнська школа-семінар молодих вчених і студентів «Сучасні комп’ютерні інформаційні технології» АСІТ, 2018. 79с.

36. Суслін В.В. Алгоритми модулярного експоненціювання для асиметричних криптосистем / Суслін В.В. Момотюк О.В, І.З.Якименко, М.М.Касянчук // Науково-практична конференція молодих вчених і студентів «Інтелектуальні комп’ютерні системи та мережі» ТНЕУ-2019. – Тернопіль, 2019 . 15.с

37. Еккель Б. Філософія Java / Брюс Еккель., 2018. – 1168 с. – (4).

38. Нуркевич Т. Реактивне програмування з прикладами RxJava / Томаш Нуркевич., Бен Крістенсен., 2017. – 358 с.

39. Методичні вказівки щодо проходження переддипломної практики студентами освітнього ступеня «Магістр» спеціальності 123 Комп’ютерна інженерія / Л.О. Дубчак, Г.М.Мельник, Ю.М. Батько – Тернопіль: ЗУНУ, 2020. - 16 с

40. Типові вимоги до оформлення дипломних робіт за освітньо-кваліфікаційними рівнями “спеціаліст” і “магістр”/ За ред. проф. Г.П. Журавля. Тернопіль: ТНЕУ, 2007. 32 с.

41. Рекомендації НМК МОНУ з „Комп’ютерної інженерії” щодо тематики, змісту і оформлення кваліфікаційних робіт випускників ВНЗ України (прийняті 16 травня 2006 року на виїзному засіданні НМК в Таврійському державному університеті ім.. В.І.Вернадського, м. Сімферополь) / Укл. проф. Тарасенко В.П., Мельник А.О., проф. Скатков О.В., проф.

42. Бюлетень ВАК України. 2007. No6.

43. Методичні вказівки до виконання кваліфікаційної кваліфікаційної роботи для студентів спеціальностей 8.091501 „Комп’ютерні системи та мережі” / В.Ф. Ємецью, Я.С.Парамуд,. Львів: Вид-во НУ „Львівська політехніка”, 2003. 28 с.

44. Магістерська атестаційна робота. Положення і методичні вказівки для магістрів інституту телекомунікаційних та комп’ютерних систем / В.М. Локазюк, В.Т. Кондратов, І.В. Троцишин. Хмельницький: ХНУ, 2004. 28 с.

45. Березький О.М., Білоусов І.А., Васильків Н.М., Васильцов І.В. Методичні рекомендації до виконання дипломної роботи з освітньо- кваліфікаційного рівня „Магістр”. Спеціальність 8.091501 – комп’ютерні систми та мережі / Під ред. І.А. Білоусова. Тернопіль: ТАНГ, 2002. 12 с.

46. ДСТУ 3008-95. Державний стандарт України. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення. Введ. 23.02.95. К.: Держстандарт України, 1995. 37 с.

47. Дипломне проектування за напрямами підготовки "Приклада математика", "Комп’ютерна інженерія", "Програмна інженерія" [Текст]: навч.-метод. посіб. / Є.С. Сулема; за заг. ред. І.А. Дички. К.:НТУУ"КПІ", 2011. 224 с.

48. Освітньо – професійна програма (проєкт) «Комп’ютерна інженерія».

Другого (магістерського) рівня вищої освіти. Галузі знань 12 Інформаційні

49. технології спеціальності 123 Комп’ютерна інженерія. Тернопіль 2020. 11 с.

50. Дипломне проектування за напрямами підготовки "Прикладна математика", "Комп’ютерна інженерія", "Програмна інженерія" [Текст]: навч.-метод. посіб. / Є.С. Сулема; за заг. ред. І.А. Дички. – К.:НТУУ"КПІ", 2011. – 224 с.

51. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання / Нац. Стандарт України. – Вид. офіц. – [Уведено вперше ; чинний від 2016-07-01]. – Київ : ДП «УкрНДНЦ», 2016. – 17 с.

52. Загальні рекомендації з підготовки, оформлення, захисту й оцінювання випускних кваліфікаційних робіт здобувачів вищої освіти першого бакалаврського і другого магістерського рівнів / за ред. доц. М.І. Шинкарика. – Тернопіль:ТНЕУ, 2018. – 60

53. Березький О.М., Дубчак Л.О., Мельник Г.М. Методичні рекомендації до виконання кваліфікаційної роботи з освітньогоступеня “Магістр”. Спеціальність: 123 - Комп’ютерна інженерія. Магістерська програма - Комп’ютерна інженерія" / Під ред. О.М. Березького – Тернопіль: ЗУНУ, 2020.– 41 с.

54. Гураль. І.В., Дубчак Л.О. Методичні вказівки до оформлення курсових проектів, звітів про проходження практики, випускних кваліфікаційних робіт для студентів спеціальності «Комп’ютерна інженерія» / Під ред. 17О.М. Березького. Тернопіль: ТНЕУ, 2019. 33 с.