Міністерство освіти і науки України

Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
завідувач кафедри кібербезпеки
та захисту інформації
_____Н.В. Лукова-Чуйко
«...» червня 2021р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

дипломної роботи
бакалавра
(назва освітнього рівня)

галузь знань 12 Інформаційні технології
(шифр і назва галузі знань)

спеціальність 125 Кібербезпека
(код і назва спеціальності)

освітня програма Кібербезпека
(назва освітньої програми)

на тему: «Дослідження асиметричних алгоритмів шифурвання на прикладі системи Рабіна»

Виконавець: студент IV курсу, групи КБ-42

_____ **Шайна Олексій Максимович**
(підпис) (прізвище ім'я по-батькові)

	Прізвище, ініціали	**Підпис**
Керівник	Ігніска В.І.	

Нормоконтроль	Зюбіна Р.В.	

Київ 2021

Міністерство освіти і науки України

Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
завідувач кафедри кібербезпеки
та захисту інформації
_____ Н.В. Лукова-Чуйко
«10» жовтня 2020 р.

ЗАВДАННЯ
на виконання дипломної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньої програми)

Студенту **КБ-42** (група) **Шайна Олексію Максимовичу** (прізвище ім'я по-батькові)

Тема дипломної роботи Дослідження асиметричних алгоритмів шифурвання на прикладі системи Рабіна

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема дипломної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №2 від 08.10.2020 р.

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Методи криптографічних перетворень.

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Аналіз методів криптографічного захисту інформації. Дослідження асиметричних криптографічних алгоритмів. Моделювання асиметричної криптографічної системи. Проведення криптографічної оцінки обраного алгоритму. Порівняння змодельованої системи з подібними системами. Практична експлуатація та оцінка змодельованої системи.

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність створення системи передачі секретних повідомлень по відкритим каналам зв'язку.

5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання:12 жовтня 2020 року

Завдання видав		В.І Ігніска
	(підпис)	(ініціали, прізвище)
Завдання прийняла до виконання		О.М Шайна
	(підпис)	(ініціали, прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів робіт	Строки виконання робіт (початок-кінець)	Відмітка про виконання
1	Провести аналіз та класифікацію криптографічних систем	07.11.2020 – 03.12.2020	*виконано*
2	Дослідити асиметричні алгоритми передачі секретних повідомлень	04.12.2020 – 27.12.2020	*виконано*
3	Змоделювати криптографічну систему для передачі секретних повідомлень по відкритим каналам зв'язку	28.01.2021 – 12.02.2021	*виконано*
4	Порівняти здмодельовану алгоритмічну систему з іншими відомими асиметричними алгоритмами	13.02.2021 – 23.02.2021	*виконано*
5	Провести криптографічну оцінку алгоритму Рабіна	24.02.2021 – 24.03.2021	*виконано*
6	Провести дослідну експлуатацію обраної криптографічної системи	25.03.2021 – 11.05.2021	*виконано*
7	Оформлення презентації	12.05.2021 – 15.05.2021	*виконано*
8	Оформлення пояснювальної записки	16.05.2021 – 08.06.2021	*виконано*
9	Підготовка до захисту дипломної роботи	09.06.2021 – 21.06.2021	*виконано*

Завдання видав		В.І Ігніска
	(підпис)	(ініціали, прізвище)
Завдання прийняв до виконання		О.М Шайна
	(підпис)	(ініціали, прізвище)

Термін подання дипломної роботи до ЕК 08 червня 2021 року

РЕФЕРАТ

Дипломна робота складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел, додатків, має 59 сторінок основного тексту, та 3 формул. Список використаних джерел містить 91 найменування і займає 7 сторінок.

Метою данної роботи є моделювання сеансу обміну секретними повідомленням, використовуючи асиметричний алгоритм шифрування Рабіна.

У роботі проведено дослідження асиметричних алгоритмів шифрування та проаналізовані інші методи перетворення вихідного тексту до зашифрованого. Побудовано асиметричний алгоритм шифрування за системою Рабіна, а також розроблена власна система для обміну секретних повідомлень з використанням шифрування.

Розроблену програмну реалізацію системи передачі інформації через мережу, з використанням алгоритму Рабіна, можна використовувати для обміну конфіденційними даними між користувачами.

Розроблена система придназначена для користувачів, що хочуть забезпечити надійний захист своїх даних, які передаються по відкритим каналам зв’язку.

Ключові слова: криптосистема, криптостійкість, криптоалгоритм, самосинхронізованність, шифрування, розшифрування, експлуатація, канали зв’язку, ініціалізація, розподілені системи, python, мережа, комунікація.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

AES	–	Advanced encryption standard
DSA	–	Digital signature algorithm
HTTP	–	Hyper text transport protocol
RSA	–	Rivest, Shamir, Adleman
ДСТУ	–	Державний стандарт України
ЕЦП	–	Електронно цифровий підпис
СВК	–	Сертифікат відкритого ключа

ЗМІСТ

ВСТУП

Актуальність данної роботи визначається тим, що користувачі обмінюються персональними даними по відкритим каналам зв'язку. Тому захист від розкриття конфіденційних даних під час передачі повинен бути реалізований у кожній системі.

Інформаційні технології стали невід'ємною частиною життя людини. Кожен день користувачі всесвітньої павутини обмінюються мільйонами мегабайтів інформації через мережу. Ми використовуємо мобільні додатки для спілкування одне з одним, створюємо особисті кабінети на різних веб-додатках, довіряємо свої персональні данні: номера мобільних телефонів та кредитних карток, паспортні данні. Використовуючи мережу кожного дня, з'являється питання, чи можна довіряти конфіденційні данні, передаючи їх веб-додаткам та іншим користувачам всесвітньої павутини. З різким зростанням інформаційного простору, зросла й кількість різноманітних видів атак на мережу та веб-додатки. Одним з таких є атака типу Man-in-the-middle attack. Загроза такого типу спричиняє перехоплення мережевого трафіку з метою розкриття конфіденційних даних. Одним з варіантів захисту від таких атак — шифрування.

Держави з всього світу сформували певні вимоги до стандартів, які описують алгоритми шифрування, що можуть використовуватись для захисту конфіденційних даних від розкриття. Вибір для конкретних систем повинен бути заснований на глибокому аналізі слабких і сильних сторін тих або інших методів шифрування. Обґрунтований вибір тієї чи іншої системи захисту взагалі ж повинен спиратися на якісь критерії ефективності.

Існує багато різноманітних видів реалізації захищених криптографічних систем. Асиметричні методи шифрування підходять для захисту персональних під час їх передачі мережею. Одним з таких є саме алгоритм Рабіна, який використовує факторизацію простих чисел, як спосіб захиститись від атак, які спрямовані на перехоплення інформації. Шифрування в криптосистемі Рабіна потребує тільки одного множення, що може бути зроблено швидко, що підтверджує той факт, що

даний алгоритм не потребує значного споживання ресурсів системи, на відмінну від всім відомого RSA, в якому шифрування виконується за рахунок експаненціального ступеня. Це вигідно, коли ресурси обмежені: наприклад при використанні карт з інтегральною схемою, що містить мікропроцесор з обмеженою пам'яттю, і при необхідності задіяти центральний процесор на короткий час. Складність криптографічної системи Рабіна - така ж, як і у процедури розкладання на множники великих чисел на два простих множника. Іншими словами, криптографічна система так само безпечна, як і всім відома RSA, проте менш витратна.

Аналіз останніх досліджень та літератури. Вчені, які зробили вклад у дослідження асиметричних алгоритмів: Н.Гісін, В.Діффі, М.Хеллман. Серед вітчизняних науковців, які досліджували асиметричні криптосистеми: Горбенко.І.В., Ященко.В.В., Бессалов А.В.

Тому *метою роботи* є моделювання сеансу обміну секретними повідомленням, використовуючи асиметричний алгоритм шифрування Рабіна.

Об'єктом дослідження в данній роботі є процес передачі секретного повідомлення по відкритому каналу зв'язку

Предметом дослідження в данній роботі - це система передачі секретних повідомлень по відкритим каналам зв'язку.

Методи дослідження дипломної роботи:

- моделювання;
- вивчення та узагальнення вітчизняної та зарубіжної літератури;
- структурний аналіз;
- порівняння.

РОЗДІЛ 1
БЕЗПЕКА КОНФІДЕНЦІЙНИХ ДАНИХ ТА ЗАСТОСУВАННЯ КРИПТОГРАФІЇ

1.1 Використання перших шифрів

Історія появи перших шифрів бере свій початок ще за довго до створення всесвітньої павутини, інформаційних технологій та сучасних алгоритмів безпеки персональних даних. Взагалі ж поняття конфіденційність вперше було застосовано у Єгипті у 3000 році до нашої ери. Єгиптяни використовували власні ієрогліфи для обміну даними з одне одним. Навіть зараз, із сучасними технологіями, неможливо розшифрувати їх шифри. Цей період часу називають епохою полі алфавітних шифрів. Принцип роботи такого таких криптосистем полягає у заміні літер або символів вихідного тексту на символи або знаки з іншого алфавіту. Такий метод заміни нагадує метод підстановки, проте дещо складніший. Прикладом алгоритму підстановки є шифр Цезаря. В ньому всі літери вихідного тексту циклічно замінюються за спеціальним ключем, змінюючи порядковий номер літери на певне число позицій.

Більш швидкими темпами криптографія починає свій розвиток під час епохи Відродження. Перша європейська книга, яка описувала 7 видів приховування тексту, носить назву “Послання монаха Роджера Бекона”. Європейці створили алгоритм, в якому кожна літера вихідного тексту шифрується за допомогою власного алгоритму. В період того часу, змогли навіть розшифрувати архіви Людовіка IV, які були скриті, з використанням алгоритму Росіньолей. Продовжують свій розвиток і полі алфавітні шифри. Був розроблений відомий алгоритм з використанням багато алфавітної заміни – шифр Віженера, який неможливо було зламати протягом наступних трьох століть. У середніх віках збільшилась дипломатична активність, що призвело до розширення інформаційного простору між країнами. Обмін повідомленнями почав вимагати ускладнення розшифрування секретних даних,

спричинивши розвиток криптографічних методів для шифрування переданої інформації. Технологія безпеки даних базувалась більш складними алгоритмами на основі знань, отриманих в ході зусиль при розшифруванні класичних шифрів і винаходу нових [1].

Сучасна криптографія бере свій початок під час першої та другої світових війн, коли використання криптографічних алгоритмів стало інструментом ведення боротьби між країнами. Розпочалося змагання серед держав з усього світу за інформаційний простір, в якому вигравав той, хто використовував більш складні та ефективні алгоритми для засекречування конфіденційних даних. Створення першої шифрувальної машини Енігма, спричинила собою резонанс поява шифрувальних та дешифрувальних пристроїв, різної якості та вартості. Багато сучасних криптоаналітиків вважають, що цей шифр був ненадійним, однак в свій час його надійність не викликала жодних сумнівів [2].

Сьогодні ж криптографічні методи використовують різні математичні моделі для створення криптостійких алгоритмів. Складність розшифрування сучасних шифрів гарантує, що перегляд повідомлень буде доступний лише отримувачу та відправнику. Такі алгоритми виключають варіанти злому простим перебором, що підвищує їх рівень криптостійкості. Комбінуючи математичні та обчислювальні методи, сучасні криптосистеми можуть створювати зашифровані повідомлення з різним рівнем секретності. Використання одних и тих же алгоритмів призводить до зниження ефективності шифрування. Цей факт підтверджує необхідність розробки нових методів шифрування конфіденційних даних, з більш складними алгоритмами, а також ефективних з точки зору швидкості дешифрування/шифрування. Спираючись на досвід минулих років, з певністю можна сказати, що кожна існуюча криптосистема рано чи пізно буде зламана, тому безпека потребує використання нових методів та алгоритмів шифрування [3].

1.2 Застосування криптографічних методів та засобів у сучасному світі

Штампування часу – це техніка, яка використовує модель шифрування сліпого підпису, дозволяючи відправнику отримати повідомлення, не розкриваючи особливостей інформації третій стороні. Такий метод зазначає, що об'єкт або зв'язок з таким об'єктом існував, або був доставлений у визначений час. Для перевірки актуальності документа та його справжності, перевірка часом являється визначним чинником. Відмічаючи дату створення, ця техніка унеможливлює підробку об'єкта, доводячи той факт, що одержувач отримав конкретний документ, а не підробку. Даний метод робить можливим перехід з паперових, затверджених мокрою печаттю документів, на електронно-юридичні документи.

Автентифікація – це процес, який дозволяє різним організаціям захищати персональні данні та власні мережеві додатки за допомогою перевірки доступу до захищених ресурсів, тобто надається доступ лише автентифікованим користувачам. Після процесу автентифікації виконується авторизація, для того щоб визначити чи може користувач мати доступ до певних об'єктів. В залежності від привілеїв користувачів надається різний рівень доступності до інформації, яка знаходиться у межах інформаційного простору організації. Взагалі данні методи взаємозамінні, хоча дуже часто використовуються разом [4].

Електронні гроші – це те поняття, яке знаходиться на стадії розвинення. Існують як апаратні види, так і цифрові. Взагалі електронна готівка використовує комбіновані методи криптографічного захисту. Для того, щоб операції, які здійснюються в електронному вигляді над грошима від одного користувача до іншого, не були сфальсифіковані, використовуються методи штампування часу, автентифікації та контролю доступу. Всі платежі фіксуються системою та вносяться до спеціальної бази розрахунків. Існує декілька систем, які охоплюють цей спектр застосувань, від транзакцій, що імітують звичайні паперові операції, вартістю від декількох доларів і вище, до різних схем мікро платежів, які проводять операції з надзвичайно низькою вартістю, до сум, які нестимуть накладні витрати на шифрування та кліринг банку. Існують також деякі гібридні підходи, коли платежі можуть бути анонімними стосовно продавця, але не банку; або анонімними для всіх, але простежуваними (послідовність покупок може бути пов'язана, але не пов'язана

безпосередньо з ідентифікацією витратника). Шифрування використовується для захисту даних, які передаються під час транзакцій, таких як номери карток, персональні данні власника та суми транзакцій [5].

Захист передачі листів електронною поштою також здійснюється за допомогою алгоритмів шифрування. Цей спосіб дозволяє зберегти таємницю листування від третіх сторін та перешкоджає отриманню персональних даних про учасників. Перехопленні електронні листи неможливо прочитати без спеціальних закритих ключів, які генеруються на стороні клієнтів, та не передаються по каналах зв'язку. Даний тип шифрування називається асиметричним. Кожен хто має електронну адресу, має пару ключів, які пов'язані з цією адресою. Відкриті ключі, прив'язані до електронної адреси та до імені користувача і зберігаються на спеціальному сервері, вони використовуються для шифрування повідомлень, а закриті ключі зберігаються на боці клієнта. За допомогою закритих ключів виконується процес дешифрування, тому передавати їх по каналах зв'язку або третім сторонам, порушує політику безпеки листування [6].

1.2.1 Шифрування в месенджерах та мобільному зв'язку

Месенджер – додаток, який використовується для обміну повідомленнями між користувачами. На відміну від електронної пошти, такі додатки працюють значно швидше за рахунок програмних інтерфейсів sock, які працюють на третьому рівні стеку TCP/IP. Існує велика кількість месенджерів, якими користуються клієнти з усього світу, наприклад WhatsApp, Telegram, Viber тощо.

Шифрування чатів у додатку Telegram використовує протокол MTProto, для наскрізного шифрування, тобто таємних чатів. У даній системі безпеки є ряд переваг, наприклад відкриті та закриті ключі, які зберігається лише на стороні клієнтів. Така архітектура побудови нагадує пірінгову систему передачі даних, де данні розподіляються лише на боці користувачів. Ця модель підвищує рівень анонімності під час обміну інформації, та зберігає таємницю листування ліпше ніж стандартна архітектура обміну повідомленнями по мережі.

WhatsApp використовує протокол Signal, який включає в собі комплексний підхід захисту за рахунок чергування асиметричних та симетричних алгоритмів. Цілісність даних забезпечується за допомогою симетричних систем, тоді як асиметричні виконують роль автентифікації, авторизації та перевірки справжності. Як вже було зазначено, комбінування різних процесів безпеки, допомагають досягти більшого рівня безпеки персональних даних, ніж використання якогось одного [7].

Система безпеки Viber використовує протокол SSL/TLS через 443 порт для шифрування запитів до серверів. Даний метод запобігає прослуховуванню розмови між користувачами, створюючи сесію зв'язку між клієнтами. Коли ваш додаток виконує запит до сервера, він з'єднується з ним за допомогою протоколу HTTPS, який використовує асиметричні алгоритми. Історія листування, яка зберігається у базі даних месенджера, знаходиться у зашифрованому вигляді за допомогою симетричних алгоритмів, які забезпечують конфіденційність при збереженні даних на інформаційних носіях [8-10].

Доступ до з'єднання за допомогою SIM-карти виконується за рахунок автентифікації користувача. Унікальний серійний номер IMEI генерується оператором зв'язку та надається клієнту для проходження успішної ідентифікації особистості. Тобто для того щоб підтримувати розмову, необхідно пройти процес підтвердження доступу за допомогою асиметричних алгоритмів безпеки.

Висновки за розділом 1

У першому розділі була проаналізована історія розвитку методів шифрування. На основі проведених досліджень можна зробити висновок, що безпека конфіденційних даних була важливою ще у 3000 році до нашої ери. З появи перших шифрів можна зазначити час, коли вперше з'явилося поняття інформаційна безпека. Криптографія стала більш популярною в середні віки, оскільки технології шифрування стали все більш складними на основі знань, отриманих в ході зусиль по розшифруванню класичних шифрів і винаходу нових шифрів. Труднощі розшифрування шифрів, які були підготовлені вручну до 20-го століття, різко зросла

з появою шифрувальних машин на початку 20-го століття. Актуальність безпеки даних, які передаються, набрала обертів під час другої світової війни. На зараз шифрування використовується у месенджерах, соціальних мережах, електронній пошті, телефонному зв’язку тощо. Також набрали популярності електронні гроші, в яких анонімізація трафіку транзакцій дуже важливе з точки зори конфіденційності. Це підтверджує те, що у сучасному світі криптографія та шифрування даних грає визначну роль у безпеці даних, що передаються мережею На формулювання поняття захисту робить вплив велика кількість різнопланових чинників, основними з яких виступають: ефективність схвалюваних рішень, концепції побудови і використання захищених інформаційних систем, технічне забезпечення інформаційних систем, характеристики інформаційних систем і їх компонентів, потенційні можливості зловмисної дії на інформацію, її отримання і використання, наявність методів і засобів захисту інформації.

РОЗДІЛ 2
АНАЛІЗ МЕТОДІВ ТА СТАНДАРТІВ ШИФРУВАННЯ

2.1 Класичні техніки шифрування

Сучасні криптосистеми будуються за таким принципом: криптостійкість повідомлення визначається секретністю приватних ключів та довжиною відкритого ключа. Такий підхід до реалізації називається принципом Кіргофа. Навіть якщо алгоритм шифрування відомий, від цього не страждає його стійкість до злому. Алгоритми спроектовані таким чином, аби унеможливити злом шифру, окрім шляху повного перебору в межах ключового простору, тобто здійснити перебір всіх можливих значень закритого ключа. Тому треба зазначити, що саме довжина ключа та секретність його розташування визначає стійкість алгоритму до злому. Існує безліч способів для засекречування конфіденційних даних, проте потрібно спиратися на два визначних чинника [11-12]:

1. Засекречування властивостей носіїв, на яких зберігається інформація
2. Секретність розташування приватних ключів

На даний момент існує чотири класи перетворень даних(Рис.2.1).

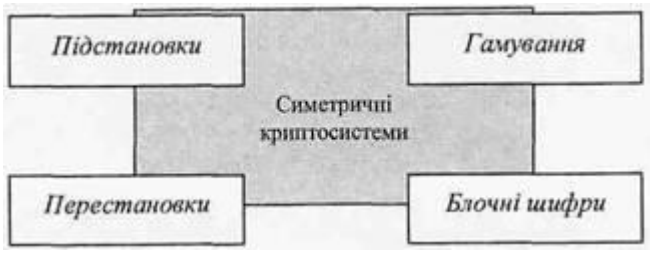

Рисунок 2.1 – Методи криптографічних перетворень

Метод підстановки виконується за рахунок заміщення літер або символів вихідного тексту на літери або символи з того ж алфавіту за певним правилом зсуву. Для підвищення стійкості такого алгоритму, його використовують в парі з методом перестановки.

Метод перестановки полягає у перестановці символів або літер вихідного тексту місцями за певним складним правилом. У більшій кількості випадків використовується у комплексних системах шифрування з іншими класами перетворень

Блочні шифри можна зустріти більш часто, вони формуються за допомогою блоків в межах від 64 до 256 біт. Цей симетричний вид шифрування затверджений міжнародними стандартами безпеки інформаційних даних. Такий клас перетворення використовує вектор ініціалізації, який додається до відкритого тексту для збільшення ключового простору, що унеможливлює перебір грубої сили [13].

Серед широко застосованих шифрів використовують гамування. Цей принцип шифрування полягає в генерації гами шифру з використанням алгоритму псевдовипадкових чисел. Дешифрування в такому випадку виконується за допомогою ще одної генерації гами, яка накладається на вже створений текст шифру. Проте даний метод стає ненадійним, якщо зловмисник дізнається частину вихідного тексту та шифрограму.

2.2.1 Симетричні алгоритми шифрування інформації

Симетричні алгоритми шифрування добре підходять для засекречування інформації на носіях, не передаючи приватні ключи по відкритими каналам зв’язку. В такій системі ключі шифрування та дешифрування збігаються, тому дані алгоритми не підходять для обміну повідомленнями через мережу. Їх умовно можна поділити на потокові та блочні шифри. Прикладом симетричної системи є шифр AES, в якому довжина блоку дорівнює 128 біт. Потокові шифри оброблюють кожен окремий біт або символ вихідного тексту. Їх можна класифікувати на одноразові педи та само синхронізуючі. Потік ключів в таких системах знаходиться весь час в одному стані. Якщо змінити хоча б один символ шифр тексту, це не буде впливати на дешифрування решти тексту. Потоки ключів в само синхронізуючих шифрах, генеруються з певної кількості попередніх символів шифр тексту, тому цей вид потокових шифрів краще обробляє символи, які додаються або видаляються з

вихідного повідомлення. Після такої обробки тексту, шифр буде синхронізуватися знову, тому такі алгоритми не мають поширених помилок. Якщо данні змінюються, кількість неправильних варіантів розшифрування буде обмежена вказаною кількістю попередніх символів, відновлюючи процес дешифрування знову.

Сучасні архіватори використовують такі симетричні алгоритми як шифр DES, IDEA, BlowFish, AES або ZIP-шифрування. Потрібно зазначити, що шифр DES та BlowFish не надійні, тому їх використання зустрічається доволі рідко. Більш того, в найближчому майбутньому залишиться лише надійний шифр AES-128. Навіть якщо з'явиться така обчислювальна потужність, яка зможе перевірити 256 ключів за секунду, то для повного перебору ключового простору знадобиться близько 149 трильйонів років.

В сучасних техніках шифрування зустрічаються криптосистеми, побудовані за принципом еліптичних кривих, які генерують швидші та ефективніші ключі за допомогою математичних еліптичних кривих. Наприклад електронна валюта біткоін використовує такі алгоритми завдяки своїй архітектурі. Шифрування даних в такій системі ECC, найчастіше використовується для засекречування інтернет-трафіку [14-18].

2.2.2 Асиметричні алгоритми шифрування інформації

Клас алгоритмів, в яких використовується така особливість як шифрування за допомогою одного ключа, а дешифрування за допомогою іншого називається асиметричним алгоритмом шифрування інформації. Перший, відкритий ключ публікується для використання користувачами в системах криптографічного захисту та застосовується для шифрування повідомлень. В більшості випадків, цей публічний ключ розташовується на публічному клієнтському сервері, який пов'язується з електронною або доменною адресою власника та має певну тривалість існування. Розсекретити інформацію за допомогою такого ключа неможливо. Для розшифрування повідомлень клієнт має приватний ключ, який знаходиться у секретному місці, доступ до якого має лише він. Головною перевагою

такого алгоритму є те, що система дозволяє обмінюватись даними через мережу без попередньої домовленості про використання ключів. В таких алгоритмах, необхідність публікувати приватний ключ відпадає, тому криптосистема не може бути уразлива до атак перехоплення мережевого трафіку з метою розкриття секретних даних. Прикладом таких систем є алгоритм Рабіна, RSA, Diffie-Hellman та DSA. Такі алгоритми, можуть використовуватись для захищення власної інформації без передачі по каналах зв'язку, проте для цього краще підходять симетричні алгоритми. Системи з відкритим ключем ідеально підходять для автентифікації та авторизації. Використання таких криптосистем в сучасних архіваторах дуже повільне та неефективне в порівнянні з симетричними алгоритмами [19-20].

Криптоалгоритми Рабіна та RSA ґрунтуються на обчислювальній складності факторизації добутку двох великих, цілих та простих чисел, які стали першими придатними одночасно для шифрування та електронно-цифрового підпису. За допомогою таких систем створюється захищене з'єднання за протоколом HTTPS, формуючи потік даних у блоки шифр тексту. Вони використовують такі властивості як:

1. Якщо відомо x, то f(x) обчислити відносно просто.
2. Якщо відомо y = f(x), то для обчислення x немає простого (ефективного) шляху.

Через такі властивості практично не уможливлюється злом шифр тексту перебором ключового простору за розумний час, використовуючи сучасні обчислювальні засоб.

В криптосистемі RSA для шифрування інформації використовується операція зведення в ступінь по модулю великого простого числа, а для дешифрування використовується функція Ейлера для розкладання числа на прості множники. Тоді як в алгоритмі Рабіна перетворення вихідного тексту здійснюється за допомогою взяття числа по модулю до простого числа, а для розшифрування використовується китайська теорема про останки. Ключі в таких алгоритмах утворюють між собою узгоджену пару, в тому сенсі, що вони є взаємно зворотними. На відміну від алгоритму RSA, шифр Рабіна не має настільки широкого практичного застосування,

через свою виняткову особливість у розшифруванні. Проте цей недолік можна усунути за допомогою додавання спеціальних заголовків до мережевих пакетів, які будуть містити позначки для вибору одного з 4 результатів перетворення [21]. Підводячи підсумок до криптографічних алгоритмів з відкритим ключем слід зазначити, що їх доцільно використовувати за такими напрямками:

1. Перш за все це саме засіб автентифікації та авторизації користувачів. Найбільш розповсюдженні криптоалгоритми на сьогоднішній день це:
    1) Алгоритм Diffie-Hellman
    2) Алгоритм Рабіна
    3) Алгоритм RSA
2. Засіб захисту персональних даних, що передаються по мережі або зберігаються на носіях.
3. Засіб для розподілу ключів. Асиметричні алгоритми вимагають значних обчислювальних затрат, тому створювати розподілені системи, які вимагають невеликого інформаційного простору, вони можуть без перешкод [22].

Незважаючи на досить велике різноманіття криптографічних систем, найпоширеніші з них це алгоритм Рабіна та RSA, які базуються на факторизації числа. Існує досить багато думок про те, яку саме з них обрати для захисту передачі даних. З цієї причини були затверджені певні критерії вибору – стандарти, що формулюють вимоги до криптографічних алгоритмів, тому при виборі тої чи іншої системи потрібно спиратись саме на них

2.2.3 Використання хеш функцій задля передачі відкритого ключа

Перетворення по визначеному алгоритму вихідного масиву даних з довільною довжиною у вихідний бітовий рядок фіксованої довжини називається – хешування. Хешування використовується для забезпечення захисту від нав’язування хибних даних зловмисником під час перехоплення повідомлення, яке передається відкритими каналами зв’язку. Даний процес застосовується для пошуку дублікатів в масивах набору даних для перевірки контрольних сум, з метою знаходження

помилок унікальності та цілісності під час передачі мережею. Хеш-функція бере на себе відповідальність за безпеку зберігання паролів. В цьому випадку визначну роль грають колізії, які є оцінкою ефективності алгоритму хеш-функцій. Хеш-алгоритм повинен мати певні властивості для того, щоб бути ефективним: стійкість до зіткнень, опір попереднього зображення та другий опір попереднього зображення [23-25].

1. Властивість, яка описує стійкість до зіткнень, означає, що важко знайти два різних значення, які утворять один і той же хеш. Така властивість має ще інше значення – хеш-функція без зіткнень. Тобто потрібно звести до мінімуму знаходження таких значень z та y, в яких буде виконуватись x(z) = z(y). Проте, виходячи з того що хеш-функція перетворює вхідні данні до фіксованої довжини, такі зіткнення можливі, але повинні траплятися крайнє рідко. Така властивість ускладнює пошук двох змінних, які будуть мати однакове хеш-значення, після використання хеш-функції. Крім того, якщо хеш-функція стійка до зіткнень, вона є другою стійкою до попереднього зображення.

2. Опір попереднього зображення – це властивість, яка означає, що хеш важко обчислити у зворотному напрямку. Тобто якщо алгоритм хешування *x* створив хеш *y,* то створити вихідне значення z, яке є хешем до y, майже неможливо.

3. Властивість другого опіру попереднього зображення, означає, що важко знайти два однакових хеш-значення. Навіть якщо зловмисник знає хеш та його значення, він однаково не зможе створити таке саме хеш-значення. Ця властивість запобігає утворенню колізій.

Слід зазначити, що існує багато хеш-алгоритмів, які використовуються в різних системах безпеки. До 2001 року хеш-функція MD5 мала досить широке застосування в операційних системах з архітектурою UNIX. Проте через велику кількість колізій на зміну такій системі прийшли алгоритми SHA-256 та SHA-512. У порівняння з MD5 дані хеш-функції мають набагато менше колізій та утворюють масив з більшої довжини вихідного значення.

Для захисту від фальсифікації інформації, хеш-функції проводять хешування за допомогою криптостійкої функції, яка в свою чергу відома лише відправнику та

одержувачу інформації. Особливість використання таких алгоритмів полягає в тому, що зловмисник не зможе відновити повідомлення із перехопленого повідомлення. При цьому, отримане повідомлення разом з хешем використовується також для ідентифікації відправника одержувачем[25-28].

2.2.4 Адміністрування ключами

Системи адміністрування ключами – це розподілена система, яка складається з ряду функцій та процедур, направлені на додавання нових користувачів до системи захисту, заміну, знищення, користування, створення, архівування та розподіл ключів шифрування. Дана система забезпечує конфіденційність обміну інформацією, ідентифікацію та цілісність даних [29].

Важливим фактором в адмініструванні є режимна робота системи, тобто правильно затверджені політики безпеки, які формують вимоги до збереження ключів шифрування. Політика безпеки в системі адміністрування визначується в контексті політики безпеки організації. Вона визначає ті загрози, якім система повинна прямо або опосередковано протидіяти. Формуються правила, які описують відповідальність та звітність всіх суб’єктів, виконуючих керування ключами шифрування. Система розподіляє ключи за рівнями: головний ключ, ключі для шифрування даних та ключі для шифрування ключів.

Всі ключі шифрування повинні мати власний термін дії, використовуючи функції життєвого циклу. Після завершення часу, відведеного на експлуатацію того чи іншого ключа, виконується процес оновлення або видалення із системи адміністрування. Такий підхід до безпеки включає в себе обмеження об’єму часу, яке може бути витрачене на криптоаналіз.

Неважливо наскільки криптостійкі алгоритми використовуються для захисту даних, якщо системи розподілення ключів складаються з тривіальних вирішень. Оприлюднення особливостей побудови архітектури ставить під загрозу безпеку всієї організації. Виходячи з цього слід зазначити, що розподілена система

адміністрування ключами повинна мати комплексний підхід до безпеки зберігання ключів.

2.3 Стандарти, що формулюють вимоги до криптографічних алгоритмів

Для формування вимог до криптографічних алгоритмів існує велика кількість різних інститутів. Головний державний орган формування вимог в Сполученних Штатах – FIPS 199. В Україні головний державний орган, що формує данні стандарти – державна служба спеціального зв'зяку [30-32].

2.3.1 Стандарт FIPS 199

FIPS 199 – це американський інститут стандартів та технологій, який формує безпеку федеральних інформаційних та інформаційних систем. В ньому затверджено, що мінімальний рівень безпеки конфіденційних даних, вимагає щонайменше 112 біт для захисту незначної інформації, 128 біт для інформації з помірним впливом та 192 біта для секретної інформації. Певний рівень безпеки буде визначати алгоритм та розміру ключа, який буде використовуватися. В більшості випадків потрібно використовувати комплексні криптографічні методи для захисту інформації. В ліпшому випадку криптосистеми можуть запропонувати один і той же рівень безпеки персональних даних, але це не завжди може стосуватися продуктивності, доступності та міркувань сумісності. Використання спільних алгоритмів різної міцності, забезпечує середній рівень криптографічного захист. Наприклад, RSA з 2048-бітними ключами може підтримувати рівень безпеки 112 біт, але часто використовується з SHA-256, який може підтримувати рівень безпеки 128 біт. Коли комбінація використовується для генерації цифрового підпису, підпис може забезпечувати лише рівень безпеки 112 біт - менший рівень, запропонований двома алгоритмами..

2.3.2 Стандарти державної служби спеціального зв'язку України

Установа державної служби може лише рекомендувати ті чи інші вимоги до безпеки даних, які не мають високого рівня секретності або не містять в собі державної таємниці. Згідно з наказом №657 до державної служби спеціального зв'язку, набуває чинності наступне: у засобах криптографічного захисту державних інформаційних ресурсів або інформації, вимога щодо захисту якої встановлена законом, реалізуються криптоалгоритми та криптопротоколи, які є національними стандартами в обсязі функцій безпеки а також стандарти, що визначають вимоги до блокових шифрів (ДСТУ 7624:2014, ДСТУ ISO/IEC 18033-3:2015), до потокових шифрів (ДСТУ 8845:2019, ДСТУ ISO/IEC 18033-4:2015), до асиметричних алгоритмів та методів (ДСТУ 4145-2002, ДСТУ ISO/IEC 14888-3:2019), до хеш-функцій (ДСТУ 7564:2014), та до управління ключами (ДСТУ ISO/IEC 11770-2:2015). В даних стандартах описуються вимоги до безпеки інформації з різним рівнем секретності, який затверджує власник інформації.

Висновки за розділом 2

У другому розділі було розглянуте наступне:

1. Методи та стандарти шифрування.

2. Визначено, що існує велика кількість методів та підходів до шифрування персональних даних.

3. криптостійкість кожного з них — це перевірка міцності шифру до злому.

4. Всі сучасні криптосхеми побудовані за принципом Кірхгофа, тобто секретність зашифрованих повідомлень визначається секретністю ключа. Це означає, що, навіть якщо криптоаналітик знає алгоритм шифрування, він однаково не зможе розшифрувати повідомлення, якщо не має відповідного ключа.

5. Всі методи шифрування поділяються на дві групи: симетричні та асиметричні.

6. Існують певні стандарти для затвердження того чи іншого алгоритму, саме вони визначають вимоги до алгоритмів шифрування,

7. Використовуючи алгоритм шифрування, потрібно забезпечити необхідний рівень захисту для інформації, що захищається. А необхідний рівень секретності визначається власником інформації.

8. Одні з надійніших криптосистем для передачі секретних повідомлень є асиметричні алгоритми шифрування, а одна з найбільш поширеніших є система Рабіна.

РОЗДІЛ 3
ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ВПРОВАДЖЕННЯ ОБРАНОГО КРИПТОГРАФІЧНОГО АЛГОРИТМУ

Метод шифрування за алгоритмом Рабина є основоположником методів шифрування, які базуються на факторизації простих чисел. Криптосистема Рабіна - це криптосистема з відкритим ключем, захист якої, як і криптосистеми RSA, пов'язана з труднощами цілочисельної факторизації. Ця криптосистема була запропонована в 1979 році Майклом Рабіном як варіант криптосистеми RSA, для якого факторизація за модулем n має майже таку ж обчислювальну складність, як отримання перетворення дешифрування в результаті перетворення шифрування. Перевага криптосистеми Рабіна полягає в тому, що було доведено, що дешифрування криптосистеми є настільки ж складним, як і цілочисельна факторизація, що на даний момент невідомо, що стосується проблеми. Недоліком цього є те, що кожен результат рабінального ритму може генеруватися за допомогою чотирьох можливих входів. Отже, кожен висновок, який є блоком тексту шифру, представлений у процедурі дешифрування до чотирьох можливих входів, які представляють блок вихідного відкритого тексту [33-35].

3.1 Шифрування

Як у будь-якій асиметричній системі, алгоритм Рабіна вимагає відкритого ключа, за рахунок якого буде виконуватись шифрування та закритого ключа, за допомогою якого буде здійснюватися дешифрування. Проте генерація закритого ключа включає в себе ряд перевірок, тому розробка алгоритму шифрування складається з наступних кроків:

- генерація двох простих чисел
- тест Рабіна Мілера
- генерація відкритого ключа

- представлення тексту для шифрування у вигляді десяткового числа

- перетворення вихідного повідомлення до шифр тексту

Оригінал тексту m шифрується за допомогою відкритого ключа - числа n за такою формулою: c = (m) ² mod n. За допомогою множення швидкість шифрування по модулю системи Рабіна більше, ніж швидкість методу шифрування RSA, навіть якщо в останньому випадку вибрати невеликий ступінь. Через те, що метод RSA вимагає експоненціального ступеня для шифрування повідомлення, швидкість значно менша ніж за алгоритмом Рабіна, проте криптостійкість не змінюється [35]. Приклад: нехай зашифрований текст m дорівнює 20, закриті ключі p та q дорівнюють 7 та 11 відповідно. З цього отримуємо відкритий ключ n, який дорівнює 7 * 11 = 77. Після цього отримаємо зашифроване повідомлення *c*:

$$c = (m)^2 \bmod n = 400 \bmod 77 = 400 \bmod 77 = 15 \quad (3.1)$$

Практично реалізовується таким чином: шифрування даних виконується за рахунок того, що повідомлення, яке передається на вхід функції, представляється у вигляді десяткового числа, функцією *bytes_to_long*, бібліотеки *Crypto*, яка є у довільному доступі. На рисунку 3.1 наведена програмна реалізація функції encrypt, яка приймає на вхід повідомлення та представляє його у зашифрованому вигляді.

```
def encrypt(self, message):
    """
    Encryption
    """
    RabinCipher.p, RabinCipher.q = RabinCipher._get_primes(self)
    RabinCipher.n = RabinCipher._open_key(self, RabinCipher.p, RabinCipher.q)
    RabinCipher.digit_message = bytes_to_long(message.encode('utf-8'))
    RabinCipher.result = pow(RabinCipher.digit_message, 2) % RabinCipher.n
    return RabinCipher.result
```

Рисунок 3.1 — Функція шифрування повідомлення

Все що потрібно для шифрування це звести у квадрат повідомлення, представлене у десятковому вигляді, та взяти його остачу від ділення по модулю до відкритого ключа n. Всі перетворення виконуються на рівні об’єктів класу RabinCrypt, для використання наслідування змінних в інших частинах програми, де це потрібно.

3.1.1 Генерація ключів

Кожна криптосистема з відкритим ключем має власний відкритий ключ, який передається по відкритому каналу, та закритий ключ, який повинен бути відомий лише відправнику та одержувачу. Відкритий ключ використовується для шифрування повідомлення, в той час як закриті ключи використовуються лише для дешифрування даних та генерації публічного ключа [38-40].

Генерація ключів полягає у наступному: вибираються два випадкових простих числа p та q за спеціальним алгоритмом для находження високо розрядних простих чисел. В залежності від інформації яку ми передаємо, довжина таких ключів повинна бути не менша ніж 128 біт. Повинна виконуватися умова: $p \equiv q \equiv 3 \bmod 4$. Виконання цих вимог сильно прискорює процедуру вилучення коренів по модулю p і q. Кількість n (number n) - відкритий ключ, цифри p і q - закритий. Обчислює кількість n = p (н =) q. Наприклад: нехай p = 7 і q = 11. тоді n = p * q = 7 * 11 = 77. кількість n = 77 (Н = 77) - відкритий ключ, і цифри p = 7 і q = 11 (М = 11) - закритий. одержувач повідомляє відправнику номер 77. відправник шифрує повідомлення, використовуючи номер 77, і відправляє його одержувачу. Одержувач розшифровує повідомлення, використовуючи цифри 7 і 11. Наведені числа у прикладі є поганим прикладом закритих ключів, через те що число 77 легко розкладається на прості множники, тому для генерації задовільних умові факторизації простих чисел, я використовую власний алгоритм [41-43].

Практична реалізація виглядає наступним чином: генерація двох простих чисел включає в себе ряд перевірок, які повинні виконуватись для того, щоб

довести, що згенероване число дійсно просте. Перш за все я склав перелік з перших сімдесяти простих чисел (Рис. 3.2), припускаючи, що просте число окрім двійки не може бути парним, та не може ділитись на будь-яке інше просте число, окрім себе.

```
prime_digits = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29,
                31, 37, 41, 43, 47, 53, 59, 61, 67,
                71, 73, 79, 83, 89, 97, 101, 103,
                107, 109, 113, 127, 131, 137, 139,
                149, 151, 157, 163, 167, 173, 179,
                181, 191, 193, 197, 199, 211, 223,
                227, 229, 233, 239, 241, 251, 257,
                263, 269, 271, 277, 281, 283, 293,
                307, 311, 313, 317, 331, 337, 347, 349]
```

Рисунок 3.2 — Перелік простих чисел

Надалі обирається довільне число з діапазону від двох у ступені (n-1) до двійки у ступені n, віднімаючи одиницю (Рис 3.3).

```
def getting_random(n):
    return random.randrange(2 ** (n - 1) + 1, 2 ** n - 1)
```

Рисунок 3.3 — Генерація випадкового числа

На рисунку 3.4 створена функція divide_test_low_primes, приймає на вхід число n, за допомогою якого можна змінювати довжину генерованих чисел та виконує перевірку на те чи можна поділити без остачі згенероване число на всі числа з переліку на рисунку 3.3. Якщо згенероване число ділиться на хоча б одне з переліку на рисунку 3.3, воно не задовольняє умові та відпадає, якщо ні, виконується перевірка довжини обраного кандидата. Мінімальна довжина не повинна бути нижча за 391 у квадраті (останнє число з переліку простих чисел на рисунку 3). Така перевірка забезпечує знаходження криптостійкого простого числа, яка виключає злом шифру методом перебору [44-46]. Вибір числа з діапазону буде

виконуватися до того моменту, поки не буде знайдено кандидата, який задовільняє умові, реалізовано зо допомогою нескінченного циклу while.

```
def divide_test_low_primes(n):
    while True:
        candidate = getting_random(n)
        for division in RabinCipher.prime_digits:
            if candidate % division == 0 and division ** 2 <= candidate:
                break
        else:
            return candidate
```

Рисунок 3.4 — Функція перевірки простих чисел на стійкість

Припускаючи, що згенероване число непарне та не поділяється на будь-яке просте число з переліку, можна перейти до тесту Рабіна Мілера.

3.1.2 Тест Рабіна Міллера

Тест Рабіна Міллера, представлений у вигляді функції test_of_miller_rabin на рисунку 3.5, виконується для перевірки згенерованого числа на те, чи дійсно воно є простим та складеним. Число n просте тільки тоді, коли виконується умова:

$$x^2 = 1 \ (mod\ n)$$

```
def test_of_miller_rabin(mrc):
    maxDivisionsByTwo = 0
    numberOfRabinTrials = 2000
    ec = mrc - 1
    while ec % 2 == 0:
        ec >>= 1
        maxDivisionsByTwo += 1
    assert (2 ** maxDivisionsByTwo * ec == mrc - 1)

    def trialComposite(round_tester):
        if pow(round_tester, ec, mrc) == 1:
            return False
        for i in range(maxDivisionsByTwo):
            if pow(round_tester, 2 ** i * ec, mrc) == mrc - 1:
                return False
        return True

    for i in range(numberOfRabinTrials):
        round_tester = random.randrange(2, mrc)
        if trialComposite(round_tester):
            return False
    return True
```

Рисунок 3.5 — Тестова функція Рабіна Мілера

Проте тест на рисунку 3.5 виконує перевірку не тільки простого числа, а на те чи є воно складеним. На практиці ж виконується:

- дано n, потрібно знайти s, таке що n - 1 = 2Sq для деякого непарного q.
- візьмемо випадкове a ∈ {1, ..., n-1}
- якщо aq = 1, то n проходить тест і ми припиняємо виконання
- для i = 0, ..., s-1 перевірити рівність a (2 ^ i) q = -1
- якщо рівність виконується, то n проходить тест (припиняємо виконання)
- якщо жодне з вищенаведених умов не виконано, то n - складене.

Тобто у ході генерації двох закритих ключів відбувається наступне: спочатку генерується випадкове число з особисто зазначеного діапазону, потім це число проходить перевірку на стійкість та довжину, після цього виконується тест Рабіна Мілера [48].

3.1.3 Генерація відкритого ключа

Два згенерованих приватних ключа p та q помножаються між собою для отримання відкритого ключа. Саме він буде передаватися по каналах мережевого зв'язку між двома клієнтами, або може розташовуватись на публічному сервері [53]. Для цього я створив закриту класову функцію _open_key, яка наведена на рисунку

```
def _open_key(self, p, q):
    """
    Open key - n
    n = p * q
    """
    self.p = p
    self.q = q
    return self.p * self.q
```

3.6.

Рисунок 3.6 — Функція генерації відкритого ключа

На рисунку 3.6 зображена класова функція _open_key, яка приймає два закритих ключа та повертає результат їх множення.

3.2 Дешифрування

Щоб розшифрувати повідомлення потрібні закриті ключі - числа p і q. Процес дешифрування виглядає наступним чином: по-перше, використовуючи розширений алгоритм Евкліда, з рівняння yp (r) · pyq (r) · q = 1 (M = 1), знайти кількість yp (r). Далі, використовуючи китайську теорему про залишки, ми обчислюємо чотири можливі варіанти розшифрування вихідного тексті [55]. Практично реалізовується наступним чином: функція дешифрування decrypt на рисунку 3.7 використовує розгорнутий алгоритм Евкліда та китайську теорему про залишки.

```
def decrypt(self, c):
    """
    Decryption
    """
    if not RabinCipher.p or not RabinCipher.q or not RabinCipher.n:
        raise ValueError
    RabinCipher.c = c
    a, b = RabinCipher._egcd(self, RabinCipher.p, RabinCipher.q)
    m1 = pow(RabinCipher.c, (RabinCipher.p + 3) // 4, RabinCipher.p)
    m2 = pow(RabinCipher.c, (RabinCipher.q + 3) // 4, RabinCipher.q)
    x = (m1 * b * RabinCipher.q + m2 * a * RabinCipher.p) % RabinCipher.n
    y = (m1 * b * RabinCipher.q - m2 * a * RabinCipher.p) % RabinCipher.n
    array = [x, RabinCipher.n - x, y, RabinCipher.n - y]
    long_text = RabinCipher._encryption_choice(self, array)
    result_array = []
    for i in long_text:
        try:
            result_array.append(long_to_bytes(int(i)).decode('utf-8'))
        except:
            continue
    return result_array[0]
```

Рисунок 3.7 — Функція дешифрування даних

Функція decrypt (Рис 3.7) виконує ряд перевірок про наявність необхідних змінних, у разі їх не наявності згенерується помилка. Перш за все для знаходження a та b, що використовуються для знаходження множників задля китайської теореми про залишки, необхідно використати розширений алгоритм Евкліда [58-61].

3.2.1 Розширений алгоритм Евкліда

Для початку потрібно розділити одне ціле число на інше (ненульове) ціле число, ми отримуємо ціле фактор ("відповідь") плюс залишок (як правило, раціональне число).

Наприклад:

13/5 = 2 ("фактор") + 3/5 ("залишок").

Можна перефразувати цей поділ, повністю з точки зору цілих чисел, без посилання на операцію ділення:

13 = 2 (5) + 3.

Звертаючи увагу на те, що цей вираз отримують із наведеного вище, множивши на дільник 5. Можна назвати цей спосіб запису ділення цілих чисел алгоритмом ділення на цілі числа. Якщо a і b є додатними цілими числами, існують цілі числа, унікальні невід’ємні цілі числа q і r, так що

$a = qb + r$, де $0 <= r < b$.

q називається часткою, а r - залишком.

Найбільший спільний дільник цілих чисел a і b, що позначається gcd (a, b), є найбільшим цілим числом, яке ділить (без залишку) як a, так і b. Наприклад:

gcd (15, 5) = 5, gcd (7, 9) = 1, gcd (12, 9) = 3, gcd (81, 57) = 3.

Gcd двох цілих чисел можна знайти шляхом багаторазового застосування алгоритму ділення, це відоме як евклідовий алгоритм [64]. Неодноразово ділити дільник на остачу, поки залишок не дорівнює 0. Gcd - це остання ненульова залишок у цьому алгоритмі. Наступний приклад показує алгоритм.

Знаходження *gcd* від 81 і 57 за Евклідовим алгоритмом:

81 = 1 (57) + 24

57 = 2 (24) + 9

24 = 2 (9) + 6

9 = 1 (6) + 3

6 = 2 (3) + 0.

З цього виходить, що якщо gcd (a, b) = r, то існують цілі числа p і s, так що:

p (a) + s (b) = r.

Змінивши кроки в Евклідовому алгоритмі, можна знайти цілі числа p і s. Зробити цу можливо за наведеним вище прикладом. Починаючи з наступного до останнього рядка, маємо:

3 = 9 -1 (6)

З верхнього рядка виходить:

6 = 24 - 2 (9), отже:

3 = 9 - 1 (24 - 2 (9)) = 3 (9) - 1 (24).

9 = 57 - 2 (24), отже:

3 = 3 (57 - 2 (24)) - 1 (24) = 3 (57) - 7 (24).

24 = 81 - 1 (57), що дає:

$$3 = 3\,(57) - 7\,(81 - 1\,(57)) = 10\,(57) - 7\,(81).$$

Отже, було знайдено:

$$p = -7 \text{ та } s = 10.$$

Процедура, якої дотримуюсь вище, трохи безладна через всі заміни, які нам доводиться робити. Можна зменшити обсяг обчислень, залучених до пошуку p і s, виконавши деякі допоміжні обчислення, рухаючись вперед в алгоритмі Евкліда (і заміни назад не будуть потрібні). Це відоме як розширений евклідовий алгоритм.

Перш ніж представляти цей розширений алгоритм Евкліда, потрібно розглянути спеціальну програму, яка є найбільш поширеним використанням алгоритму. Використовувати форму алгоритму, яка лише вирішує цей особливий випадок, хоча загальний алгоритм не набагато складніший [66].

Можна розглянути налаштування криптосистеми Hill. Припущу що потрібно зробити арифметику за модулем 26, і іноді доводилось знаходити обернену до числа мод 26. Це виявилося складним завданням (і не завжди можливим). Ми спостерігали, що число x мало зворотний модуль 26 (тобто число y так, що xy = 1 mod 26) тоді і лише тоді, коли gcd (x, 26) = 1. Тут немає нічого особливого щодо 26, тому потрібно розглянути загальний випадок знаходження обернених чисел за модулем n. Обернене до x існує тоді і лише тоді, коли gcd (x, n) = 1. З цього виходить, що якщо це правда, існують цілі числа p і s, так що

$$px + sn = 1.$$

Але це говорить про те, що px = 1 + (-s) n, або іншими словами, px (еквівалент) 1 (mod n). Отже, p (зменшений mod n, якщо потрібно) є оберненим x mod n. Розширений алгоритм Евкліда дасть метод для ефективного обчислення p (зауважу, що в цьому додатку не дбаємо про значення для s, тому просто будемо його ігнорувати).

Розширений евклідівський алгоритм для знаходження оберненого до числа mod n, потрібно нумерувати кроки евклідового алгоритму, починаючи з кроку 0. Фактор, отриманий на етапі i, буде позначатися qi. Під час виконання кожного кроку

алгоритму Евкліда ми також потрібно обчислити допоміжне число pi. Для перших двох кроків дається значення цього числа: p0 = 0 і p1 = 1. Для решти кроків рекурсивно обчислюємо pi = pi-2 - pi-1 qi-2 (mod n). Продовжую це обчислення на один крок після останнього кроку алгоритму Евкліда.

Алгоритм починається з "ділення" n на x. Якщо останній ненульовий залишок трапляється на етапі k, то якщо цей залишок дорівнює 1, x має обернене значення і це pk + 2. (Якщо залишок не дорівнює 1, то x не має зворотного.) Наприклад:

Знайду обернену до 15 mod 26.

Крок 0: 26 = 1 (15) + 11 p0 = 0

Крок 1: 15 = 1 (11) + 4 p1 = 1

Крок 2: 11 = 2 (4) + 3 p2 = 0 - 1 (1) mod 26 = 25

Крок 3: 4 = 1 (3) + 1 p3 = 1 - 25 (1) mod 26 = -24 mod 26 = 2

Крок 4: 3 = 3 (1) + 0 p4 = 25 - 2 (2) мод 26 = 21

p5 = 2 - 21 (1) mod 26 = -19 mod 26 = 7

Хочу звернути увагу, що 15 (7) = 105 = 1 + 4 (26) (еквівалент) 1 (мод 26).

Практична реалізація данного алгоритму виконується на рисунку 3.8:

```
def _egcd(self, a, b):
    """
    Extended GCD (Euklid algorithm)
    """
    x, y, u, v = 0, 1, 1, 0
    while a != 0:
        q, r = b // a, b % a
        m, n = x - u * q, y - v * q
        b, a, x, y, u, v = a, r, u, v, m, n
    return x, y
```

Рисунок 3.8 — Розгорнутий алгоритм Евкліда

Після того, як будуть знайденні a та b, за допомогою алгоритму Евкліда, знаходжу необхідні множники m1 та m2. Другим аргументом у функції pow

використовую перевірку відношення як 3 до 4, бо згенеровані числа повинні бути відповідними до цієї умови за алгоритмом Рабіна [68]. Надалі виконується китайська теорема про залишки. Щоб отримати дійсний результат з даної теореми, я припустив, що при оберненому дешифруванні невірні варіанти неможливо буде декодувати, тому зробив перевірку можливих рішень за допомогою оператора try на рисунку 3.9.

```
result_array = []
for i in long_text:
    try:
        result_array.append(long_to_bytes(int(i)).decode('utf-8'))
    except:
        continue
return result_array[0]
```

Рисунок 3.9 — Виключення можливих варіантів відповідей

Цикл на рисунку 3.9 використовує функцію представлення десяткового числа в оберненому символьному вигляді та декодує його за допомогою функції декодування

```
while len(result) != 2:
    while True:
        n = 512
        prime_candidate = divide_test_low_primes(n)
        if not test_of_miller_rabin(prime_candidate):
            continue
        else:
            if (prime_candidate % 4) == 3:
                result.append(prime_candidate)
                break
            else:
                continue
return result
```

Рисунок 3.10 — Криптостійкість Алгоритму

Використовуючи цикл while для проходження вибіркового вибору необхідних кандидатів, виконується перевірка відношення p = q як 3 mod 4 (Рис 3.10).

3.2.2 Китайська теорема про залишки

У сучасному формулюванні китайська теорема про залишки звучить так: нехай p = p1 * p2 * ... * pk , де pi — попарно взаємно прості числа. Підставимо у відповідність довільному числу a (0 <= a <= p) кортеж (a1, ... , ak), де ai = a mod pi. Тоді ця відповідність (між числами і кортежами) буде взаємно однозначним [71]. І, більш того, операції, що виконуються над числом a, можна еквівалентно виконувати над відповідними елементами кортежами - шляхом незалежного виконання операцій над кожним компонентом. З цього виходить рівняння на рисунку 3.11.

$$a \Longleftrightarrow \Big(a_1, \ldots, a_k\Big),$$
$$b \Longleftrightarrow \Big(b_1, \ldots, b_k\Big),$$

Рисунок 3.11 — Незалежне виконання операцій

Модулярне рівняння розв’язується на рисунку 3.12:

$$(a+b) \pmod p \Longleftrightarrow \Big((a_1+b_1) \pmod{p_1}, \ldots, (a_k+b_k) \pmod{p_k}\Big),$$
$$(a-b) \pmod p \Longleftrightarrow \Big((a_1-b_1) \pmod{p_1}, \ldots, (a_k-b_k) \pmod{p_k}\Big),$$
$$(a \cdot b) \pmod p \Longleftrightarrow \Big((a_1 \cdot b_1) \pmod{p_1}, \ldots, (a_k \cdot b_k) \pmod{p_k}\Big).$$

Рисунок 3.12 — Модулярне рівняння

У своєму первинному формулюванні ця теорема була доведена китайським математиком Сунь-Цзи приблизно в 100 р н.е. А саме, він показав в окремому випадку еквівалентність рішення системи модулярних рівнянь і рішення одного модулярного рівняння [75].

Програмна реалізація китайської теореми про залишки була наведена на рисунку 8 у функції decrypt. Використовуючи комбінований метод знаходження необхідних множників, за допомогою розширеного алгоритму Евкліда, дана функція повертає 4 дійсних результати, та обирає правильний варіант відповіді для подальшого представлення у читабельному вигляді [76].

3.3 Аналіз алгоритму Рабіна

Для того щоб виконати криптографічну оцінку та аналіз алгоритму Рабіна, необхідно врахувати його ефективність та криптостійкість до злому.

3.3.1 Ефективність

Розшифрування тексту на додачу веде до трьох помилкових результатів - це головний недолік криптосистеми Рабіна і один з факторів, який перешкоджав тому, щоб він знайшов більш широке застосування ніж RSA. Якщо вихідний текст являє собою текст повідомлення, визначення правильного тексту - це не складно вирішити. Один із способів вирішення цієї проблеми є, додавання в повідомлення до шифрування - заголовка, або певної позначки. Алгоритм Рабіна схожий на шифрування RSA, але замість того, щоб будувати повідомлення у десятковому вигляді у ступінь, коли для шифрування використовується операція зведення в квадрат, який благотворно впливає на швидкість виконання алгоритму без шкоди для надійності [77]. Ефективність використання алгоритму Рабіна полягає в його швидкості шифрування, що впливає на продуктивність роботи, та у простоті побудови.

3.3.2 Безпека алгоритму

Велика перевага криптосистеми Рабіна є те, що випадковий текст може бути повністю вилучений з зашифрованого тексту тільки за умови, що перекладач

здатний ефективно згенерувати та передати відкритий ключ n. Криптосистема Рабина є доказовою стійкою до атаки на основі обраного відкритого зашифрованого тексту в рамках підходу "все або нічого", тоді і тільки тоді, коли проблема розкладання цілого числа на прості множники складна [80]. Опір принцип "все або нічого" полягає в тому, що, маючи текст, закодованих в певному алгоритмі, зловмисникові необхідно реконструювати блок вихідного тексту, який, як правило, визначає безпеку параметра криптосистеми. маючи вихідний і зашифрований текст, зловмисник повинен відновити весь блок секретний ключ під час нападу або досягає успіху, або не отримує нічого. Слово "нічого" передбачає, що зловмисник вже не секретна інформація, ні до, ні після невдалої атаки.

Криптосистема Рабіна є абсолютно не уразлива до атак на основі обраного шифр тексту, як правило, зловмисники використовують всі його можливості. Наприклад, додавання надмірності, повторення останніх 64 біт, ви можете зробити тільки корінь. Алгоритм розшифрування в даному випадку дає один корінь, який вже відомий зловмисникові [81-83].

3.4 Особливості програмної реалізації алгоритма Рабіна

Шукаючи існуючу бібліотеку, яка шифрує та дешифрує текст за алгоритмом Рабіна, я зіткнувся з тим, що її немає у довільному доступі, тому для розробки системи безпеки для передачі даних, я вирішив написати її самостійно [84].

3.4.1 Розробка клієнтів

Перед тим як розробляти клієнтів, я вирішив створити для них графічний інтерфейс для спілкування. Для проектування я використав внутрішню бібліотеку Tkinter. Tkinter - крос-платформна подієво-орієнтована графічна бібліотека на основі засобів Tk, написана Стін Лумхольтом і Гвідо ван Россум. Входить в стандартну бібліотеку Python. Tkinter - це вільне програмне забезпечення, яке розповсюджується під Python-ліцензією [85-87].

Інтерфейс для спілкування не відрізняється в обох клієнтах, окрім назви елемента Button, який використовується як клас-наслідник з базовими внутрішніми функціями, та логіки обробки інформації.

Як зазначено на рисунку 3.13, я створюю змінну window, яка й буде розміщати в собі всі об'явленні елементи робочого вікна.

```
window = Tk()
window.title("Bob")
frame1 = Frame(master=window, width=1115, height=400)
frame1.pack()
btn_send = Button(text="Init connection", width=91, height=3, font='Times 18', bg="grey", fg="white", command=init)
input_field = Text(master=frame1, width=50, height=30, font='Times 16', bg="white", fg="black")
output_field = Text(master=frame1, width=50, height=30, font='Times 16', bg="white", fg="black")
input_field.place(x=0, y=0)
output_field.place(x=560, y=0)
btn_send.pack(side=BOTTOM, expand=True)
```

Рисунок 3.13 — Створення графічного інтерфейсу

Клієнт Bob

Для того, щоб створити клієнтів, які можуть спілкуватись одне з одним, за протоколом https, я використав внутрішню бібліотеку socket. Інтерфейс Python – пряма транслітерація системного виклику Unix і інтерфейсу бібліотеки до об'єктно-орієнтованого стилю Python: socket () повертає об'єкт socket, методи якого здійснюють різні системні виклики socket. Типи параметрів трохи вище, ніж в інтерфейсі C: як і при read () і write () операціях з файлами Python, виділення буфера при операціях отримання відбувається автоматично, а довжина буфера неявно відбивається при операціях відправки [88].

Клієнт Bob зініціалізує з'єднання з клієнтом alisa, згенерує відкритий ключ для шифрування данних та передасть їх до клієнта alisa. Клієнт Bob розміщується на localhost за ip адресою 127.0.0.1, порт для взаємодії я обрав 8002. За цим номером не зареєстровані стандартні служби, тому використовую саме його. Як зазначено на рисунку 3.14, для ініціалізації з'єднання я використовую функцію init, яка закріплена за кнопкою 'Init connection' [89].

```
def init():
    cipher = RabinCipher()
    RabinCipher.p, RabinCipher.q = cipher._get_primes()
    open_key = RabinCipher.n = cipher._open_key(RabinCipher.q, RabinCipher.p)
    input_field.insert(1.0, "Generated open key for Alisa: " + str(open_key))
    sock = socket.socket()
    sock.connect(('localhost', 8002))
    sock.send(bytes(str(open_key), encoding='utf-8'))
    data = sock.recv(2048)
    sock.close()
    result = RabinCipher.decrypt(cipher, int(data.decode('utf-8')))
    output_field.insert(1.0, "Message from Alisa: " + str(result))
    return result
```

Рисунок 3.14 — Опис функції init

Клієнт Alisa

Функція на рисунку 3.15 згенерує відкритий ключ для клієнта alisa, та після отримання даних від alisa, припинить виконання програми.Клієнт Alisa, як і клієнт

```
while True:
    data = conn.recv(1024)
    if data:
        from tkinter import *

        def send_message():
            content = input_field.get(1.0, "end-1c")
            cipher = RabinCipher()
            RabinCipher.n = int(data.decode('utf-8'))
            encrypted_text = RabinCipher.encrypt(cipher, content)
            output_field.insert(1.0, "Encrypted message: " + str(encrypted_text))
            conn.send(bytes(str(encrypted_text), encoding='utf-8'))


        window = Tk()
        window.title("Alisa")

        frame1 = Frame(master=window, width=1115, height=400)
        frame1.pack()
        btn_send = Button(text="Send", width=91, height=3, font='Times 18', bg="grey", fg="white", command=send_message)
        input_field = Text(master=frame1, width=50, height=30, font='Times 16', bg="white", fg="black")
        output_field = Text(master=frame1, width=50, height=30, font='Times 16', bg="white", fg="black")
        input_field.place(x=0, y=0)
        output_field.place(x=560, y=0)
        btn_send.pack(side=BOTTOM, expand=True)
        window.mainloop()

    if not data:
        break

conn.close()
```

Рисунок 3.15 — Розробка клієнту Alisa

Bob, розташовується за адресою localhost — 127.0.0.1, та прослуховує порт за номером 8002. Саме за цим портом, клієнт Bob ініціалізує з'єднання з клієнтом Alisa.

Як зазначено на рисунку 3.15, Alisa прослуховує з'єднання, поки не будуть надісланні данні. Після того як клієнт отримає свій відкритий ключ, він зашифрує данні за допомогою функції encrypt, та зможе передати їх клієнту Bob [90].

3.4.2 Тестування та аналіз створеного програмного забезпечення

Для початку клієнт Alisa ініціалізує з'єднання з клієнтом Bob. Клієнт Bob, отримуючи запит за з'єднання, згенерує відкритий ключ для клієнта Alisa та зможе передати його бо відкритому каналу, утримуючи закриті ключі у себе, та не афішуючи їх нікому. На рисунку 3.16 можна побачити інтерфейс програми клієнта Bob.

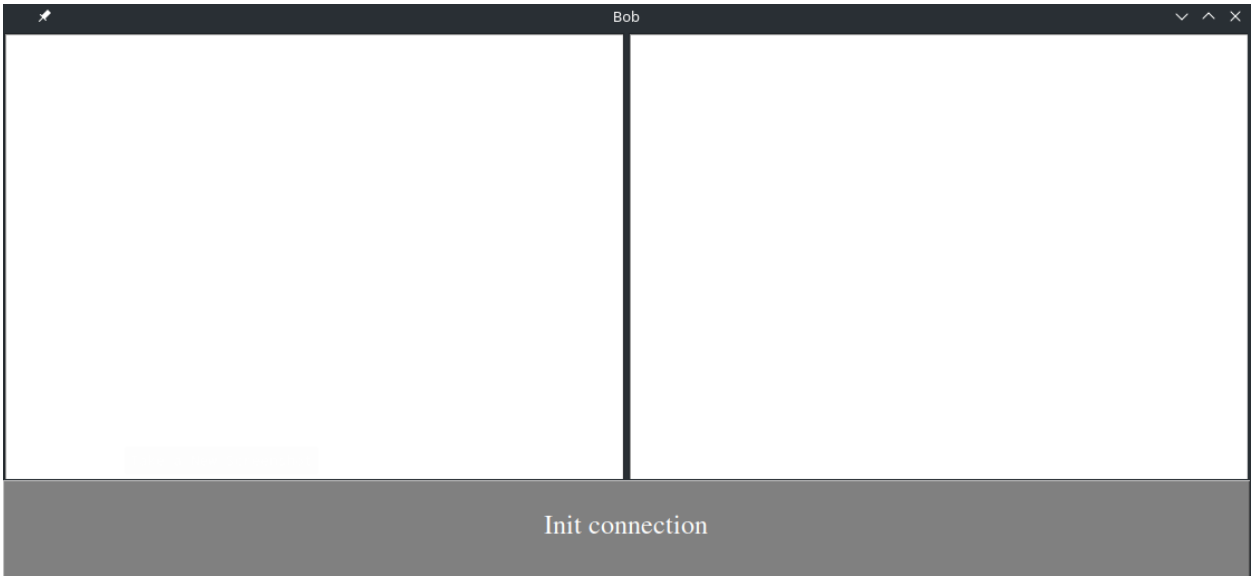

Рисунок 3.16 — Інтерфейс клієнта Bob

З лівої сторони графічного інтерфейсу буде відображений згенерований відкритий ключ (рисунок 3.17) а з правої сторони, розшифроване повідомлення від клієнта Alisa [91].

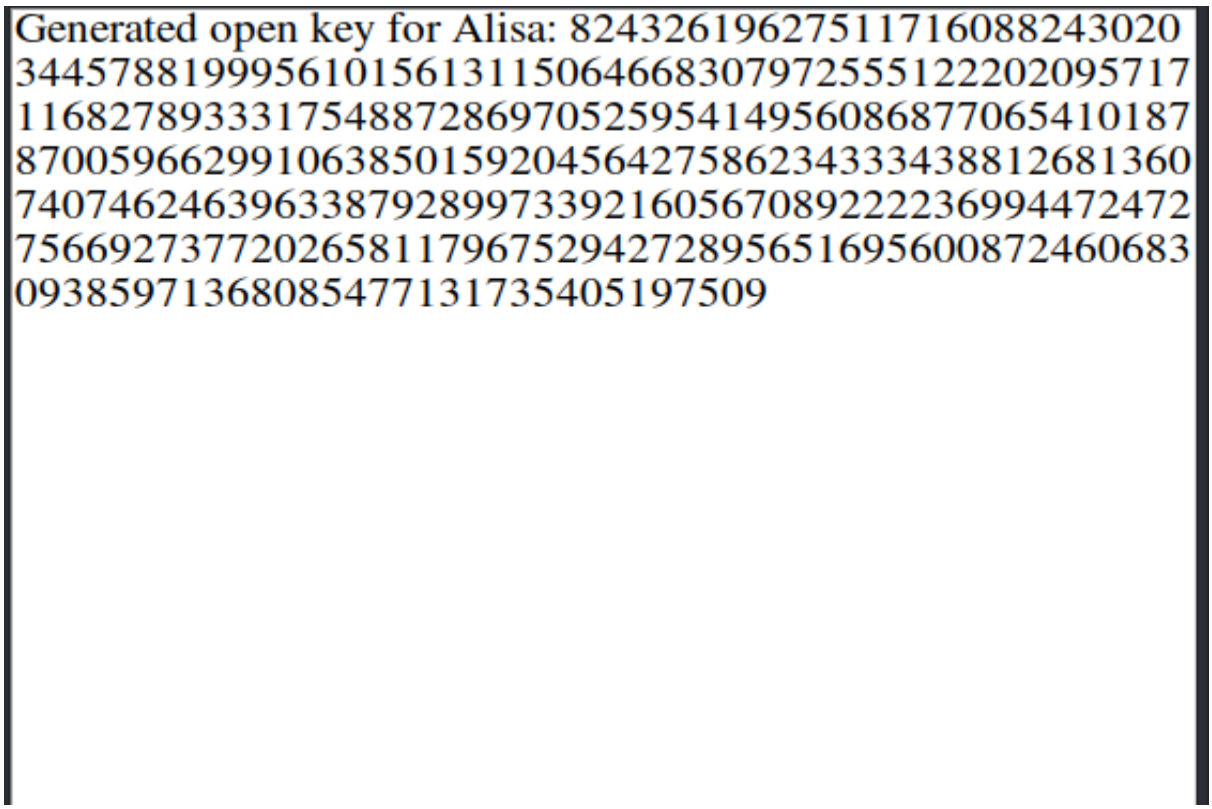

Рисунок 3.17 — Згенерований відкритий ключ

Після того, як клієнт Bob згенерував відкритий ключ та передав його клієнту Alisa, відкриється графічний інтерфейс клієнта Alisa (Рис 3.18).

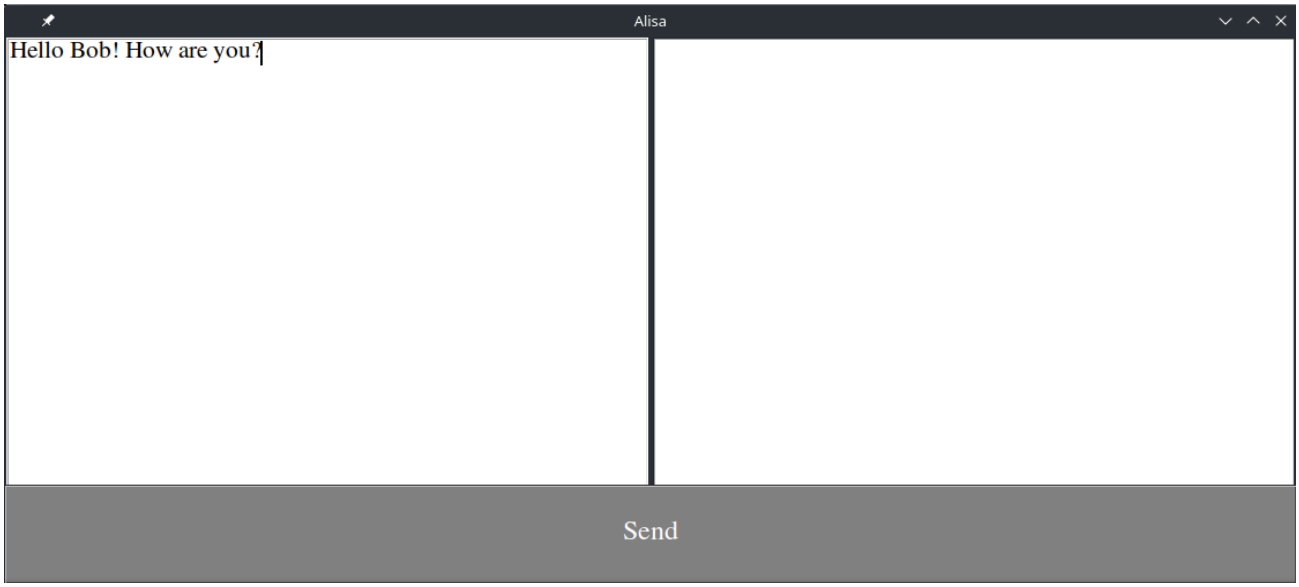

Рисунок 3.18 — Графічний інтерфейс клієнта Alisa

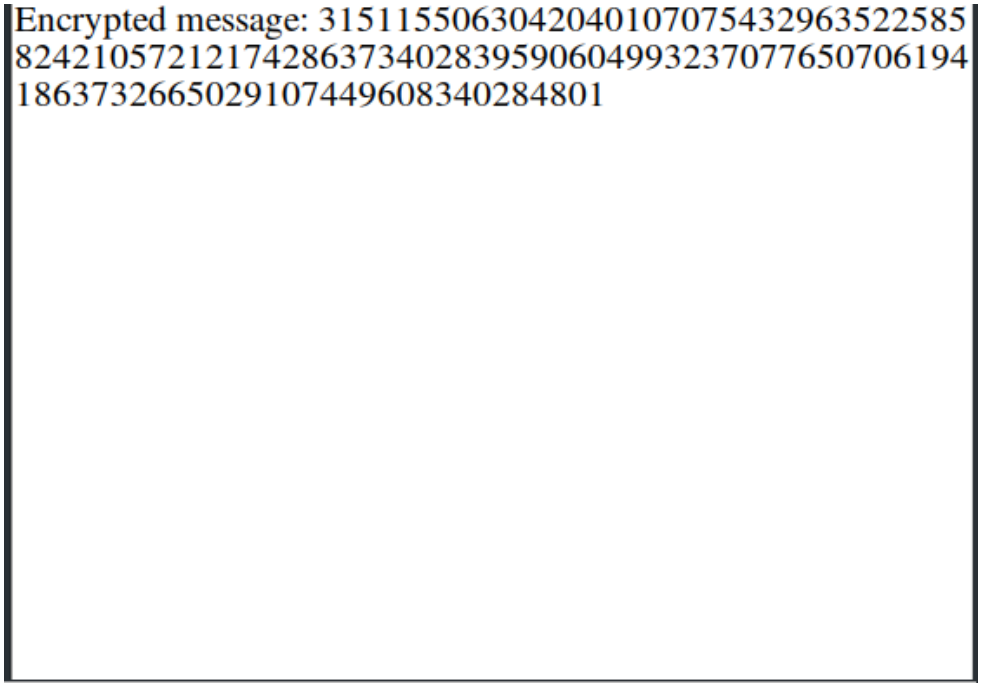

Рисунок 3.19 — Зашифроване повідомлення

З лівої сторони (рисунок 3.18) — текстове поле для введення даних, які потрібно передати клієнту Bob. З правої сторони (Рис 3.19) — зашифрований текст, за допомогою відкритого ключа. Після того, як клієнт Alisa натисне на кнопку Send, зашифроване повідомлення зможе передатися до клієнта Bob (Рис 3.20).

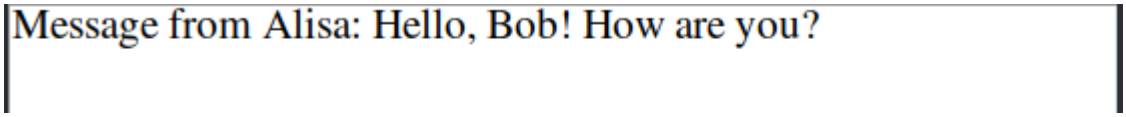

Рисунок 3.20 — Розшифроване повідомлення

На рисунку 3.21 представленні два діалогових вікна. Зверху інтерфейс клієнта Bob, а знизу клієнта Alisa. Простота використання такого графічного інтерфейсу, дозволяє зосередитись на реалізованій системі безпеки, яка включає в собі шифрування криптостійкого алгоритму шифрування Рабіна. Архітектуру такої системи легко можна маштабувати, додавши сервер, в якому будуть зберігатися відкриті ключи для шифрування.

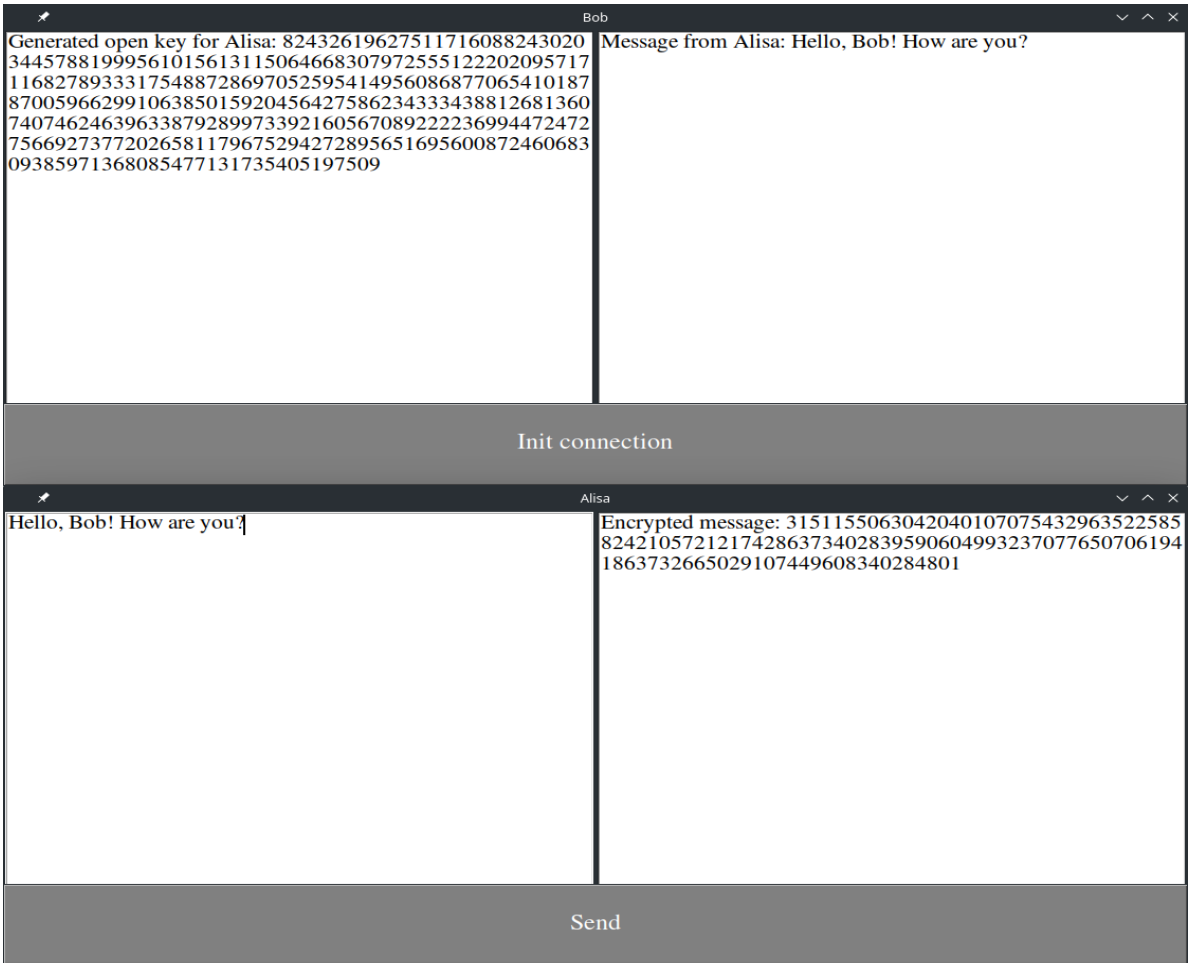

Рисунок 3.21 — Діалогові вікна обох клієнтів

На рисунку 3.21 зображений кінцевий результат виконання змодельованої криптосистеми для передачі секретних повідомлень.

Висновки за розділом 3

У 3 розділі було проведено дослідження асиметричного алгоритму шифрування Рабіна. Проаналізована його ефективність та безпека використання. Була розроблена система для передачі повідомлень мережею. Підсумовуючи все, що було досліджено, можна сказати що криптосистема Рабіна, як будь-яка асиметрична криптографічна система використовує публічний та секретні ключі для шифрування та розшифрування даних. Публічний ключ використовується для шифрування даних. За допомогою такого ключа неможливо розшифрувати повідомлення, тому він може передаватися каналах зв’язку або зберігатися на сервері з публічним

доступом для кожного користувача. Приватний ключ розташовується на боці клієнта або у секретному місці та не повинен передаватися через мережу. Генерація закритий ключів в криптосистемі Рабіна виконується за допомогою тестування кандидатів на те чи є число складеним, задовольняє умові, що формує визначення просто числа та відношення один до одного як 3 mod 4. Під час розшифрування використовується розширений алгоритм Евкліда, для знаходження необхідних дільників до китайської теорему про залишки. Ефективність даного алгоритму полягає в швидкості його шифрування, а безпека у факторизації великого простого числа. Даний метод унеможливлює злом алгоритму, окрім як перебір грубої сили. Для підтвердження отриманих результатів під час дослідження була реалізована власна система для передачі повідомлень через мережу, з використанням графічного інтерфейсу

ВИСНОВКИ

У ході дипломної роботи було розв’язано завдання щодо моделювання криптографічної системи на прикладі асиметричного алгоритму шифрування Рабіна. В ході розв’язання поставлених задач були отримані наступні наукові та практичні результати:

1. Було проведено аналіз та класифікацію існуючих алгоритмів шифрування. Досліджено основні класи перетворень та їх практичне застосування.

2. Досліджено та проаналізовано найпоширеніші асимтеричні алгоритми та проведено порівняння ефективності використання криптографічних систем цього типу під час передачі по відкритим каналам зв’язку.

3. Було змодельовано криптографічну систему на основі асиметричного алгоритму Рабіна для передачі секретних повідомлень по відкритим каналам зв’язку

4. Було порівняно змодельовану криптографічну систему з іншими відомими асиметричними алгоритми та проведено криптографічну оцінку алгоритму Рабіна

5. Розроблено та обґрунтовано технічну реалізацію системи передачі секретних повідомлень на основі обраного криптографічного алгоритму.

6. Було проведено дослідну експлуатацію асиметричної криптографічної системи Рабіна

Після проведення практичного тестування криптосистеми Рабіна у системі для передачі секретних повідомлень можна зробити висновок, що даний асиметричний алгоритм підходить для захисту конфіденційних даних, які передаються по відкритим каналам зв’язку. Було доведено, що використання алгоритмів шифрування для безпеки персональних даних, під час передачі по мережі, є ефективним. Швидкість шифрування у криптосистемі Рабіна швидша ніж в інших поширених асиметричних алгоритмах. Безпека шифру базуєтсья на факторизації великих чисел, що підтверджує той факт, що системи такого типу не вразливі до атак перебору за розумний час. Змодельовану криптографічну систему можна вдосконалити, за допомогою використання алгоритмів, які спрямовані на

оптимально швидке знаходження великих додатніх простих чисел. Поліпшення ефективності алгоритму Рабіна можна досягти за допомогою додавання спеціальних заголовків, які формують посилання на правильний результат дешифрування, під час передачі потоку бітів через мережу, що збільшить швидкість обробки вихідного тексту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вікіпедія. Криптографія [Електронний ресурс] / Вікіпедія. – 2021. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Cryptography.

2. EconomicTimes. Defensive [Електронний ресурс] / EconomicTimes. – 2021. – Режим доступу до ресурсу: https://economictimes.indiatimes.com/definition /cryptography.

3. Марк Лутц. Python / Марк Лутц. 2019. – 811 с. – (ORailly).

4. Kasperskiy L. Cryptography [Електронний ресурс] / Labarotory Kasperskiy // Kasperskiy. – 2021. – Режим доступу до ресурсу: https://www.kaspersky.com/resource-center/definitions/what-is-cryptography.

5. Mark G. Ciphers [Електронний ресурс] / Golosii Mark // Practical Cryptography. – 2020. – Режим доступу до ресурсу: http://practicalcryptography.com/ciphers/.

6. Europe C. Post-Quantum Cryptography [Електронний ресурс] / Coolisi Europe // Practical Cryptography. – 2017. – Режим доступу до ресурсу: https://www.enisa.europa.eu/publications/post-quantum-cryptography-current-state-and-quantum-mitigation.

7. Tech T. Encryption [Електронний ресурс] / Target Tech // Practical Cryptography – Режим доступу до ресурсу: https://searchsecurity.techtarget.com/definition/encryption.

8. Україна. Стандарти що формують вимоги до криптографічних алгоритмів [Електронний ресурс] / Україна // Служба державного спеціального зв'язку – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0651-08#Text.

9. Україна. KMU [Електронний ресурс] / Україна // Служба державного спеціального зв'язку – Режим доступу до ресурсу: https://www.kmu.gov.ua/news/247952015

10. Encryption Defenition [Електронний ресурс] // Investopedia. – 2019. – Режим доступу до ресурсу: https://www.investopedia.com/terms/e/encryption.asp.

11. Саймон Сингх. Книга шифров. Тайная история шифров и их расшифровки / Саймон Сингх. – New York, 1999. – 416 с. – (Doubleday).

12. Практична криптографія / Ш. Брюс, Ф. Нільс. – 811 с. – (Наталья Селина). Реалізація криптографічних систем на практиці. Моделювання захисту та впровадження шифрування.

13. Дэвид Кан. Взломщики кодов / Дэвид Кан. 1967. – 1164 с. – (Macmillan Publishers).

14. Іжевський Л. В. Управління криптографічною безпекою / Л. В. Іжевський. // Управлінська посада, проектування, прогназування, управління (практична цінність). – 2011.

15. Statistical fraud detection: A review / R. J. Bolton, D. J. Hand. // Statistical Science. – 2002. – №17 (3). – P. 235–249.

16. Fast, Accurate, End-To-End Enterprise Risk Management With ACI Proactive Risk Manager [Electronic resource] // ACI Universal Payments. – 2018. – Access: https://www.aciworldwide.com/-/media/files/collateral/data-sheets/aci-proactive-risk-manager-for-enterprise-risk-fl-us.pdf.

17. Bruce Schneier. Applied Cryptography: Protocols, Algorithms, and Source Code in C / Bruce Schneier. 2017. – 1233 с.

18. Как выбрать стандарт связи для сети IoT [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: https://habr.com/ru/company/commandspot/blog/390825/

19. Іван Е. Таинственные страницы (Занимательная криптография) / Ефішев Іван. – Москва, 2011. – 452 с.

20. The Guardian [Електронний ресурс] – Режим доступу до ресурсу: https://www.theguardian.com/technology/2018/oct/22/what-is-the-internet-13-key-questions-answered

21. SAS Fraud Management [Electronic resource] // SAS. – 2018. – Access: https://www.sas.com/content/dam/SAS/en_us/doc/factsheet/sas-fraud-management-109498.pdf.

22. Гудфеллоу Я., Бенджио И., Курвилль А. Математичний аналіз/ пер. с анг. А. А. Слинкина. – 2-е изд., испр. – М.: ДМК Пресс, 2018. – 652 с.: цв. ил.

23. Архитектура Y messenger [Електронний ресурс] // habr. – 2019. – Режим доступу до ресурсу: https://habr.com/ru/post/507206/.

24. HandsOnCryptography / D. Hand, H. Mannila, P. Smyth. – Cambridge, MA, 2001. – 546 p.

25. Сравнение 3х популярных мессенджеров для безопасного общения. [Електронний ресурс] // Crypto World. – 2018. – Режим доступу до ресурсу: https://cryptoworld.su/sravnenie-3x-populyarnyx-messendzherov-dlya-bezopasnogo-obshheniya/.

26. С глазу на глаз. 7 мессенджеров для приватной переписки [Електронний ресурс] // Club Est Nod. – 2020. – Режим доступу до ресурсу: https://club.esetnod32.ru/articles/analitika/s-glazu-na-glaz/.

27. Как сделать свой мессенджер. Создание мессенджера. Наш подход к разработке архитектуры мессенджера [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://bar812.ru/kak-sdelat-svoi-messendzher-sozdanie-messendzhera-nash-podhod-k-razrabotke.html.

28. Explainer: What is Messenger? [Електронний ресурс] // WebWise. – 2021. – Режим доступу до ресурсу: https://www.webwise.ie/parents/explained-what-is-messenger/.

29. Telegram (software) [Електронний ресурс] // Wikipedia. – 2021. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Telegram_(software)

30. Congdon P. Applied Bayesian Modelling / Congdon., 2003. – 478 p. – (Wiley; 1 edition).

31. Assymetric Cryptography [Електронний ресурс] // Wikipedia. – 2021. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Public-key_cryptography.

32. Duda R. O. Pattern classification 2nd Edition / R. O. Duda, P. E. Hart, D. G. Stork. – NY: Wiley-Interscience, 2001. – 738 p.

33. Public-key cryptography [Електронний ресурс] // CryptographyIo. – 2020. – Режим доступу до ресурсу: https://cryptography.io/en/latest/hazmat/primitives/asymmetric/.

34. Differences between Hash functions, Symmetric & Asymmetric Algorithms [Електронний ресурс] // CryptoMathic. – 2017. – Режим доступу до ресурсу: https://www.cryptomathic.com/news-events/blog/differences-between-hash-functions-symmetric-asymmetric-algorithms.

35. Types of Encryption: 5 Encryption Algorithms & How to Choose the Right One [Електронний ресурс] // Security Boulevard. – 2019. – Режим доступу до ресурсу: https://securityboulevard.com/2020/05/types-of-encryption-5-encryption-algorithms-how-to-choose-the-right-one/.

36. Symmetric Key Encryption - why, where and how it's used in banking [Електронний ресурс] // Peter Smirnoff & Dawn M. Turner. – 2019. – Режим доступу до ресурсу: https://www.cryptomathic.com/news-events/blog/symmetric-key-encryption-why-where-and-how-its-used-in-banking.

37. Symmetric Encryption 101: Definition, How It Works & When It's Used [Електронний ресурс] // TheSslStore. – 2017. – Режим доступу до ресурсу: https://www.thesslstore.com/blog/symmetric-encryption-101-definition-how-it-works-when-its-used/.

38. Symmetric cryptography [Електронний ресурс] // IBM. – 2018. – Режим доступу до ресурсу: https://www.ibm.com/docs/en/ztpf/1.1.0.14?topic=concepts-symmetric-cryptography.

39. Problem Solving with Algorithms and Data Structures using Python [Електронний ресурс] // Runestone. – 2016. – Режим доступу до ресурсу: https://runestone.academy/runestone/books/published/pythonds/index.html.

40. 10 Algorithms To Solve Before your Python Coding Interview [Електронний ресурс] // Medium. – 2021. – Режим доступу до ресурсу: https://towardsdatascience.com/10-algorithms-to-solve-before-your-python-coding-interview-feb74fb9bc27.

41. Data Structures and Algorithms in Python [Електронний ресурс] // Jovian. – 2020. – Режим доступу до ресурсу: https://jovian.ai/learn/data-structures-and-algorithms-in-python.

42. Master Algorithms with Python for Coding Interviews [Електронний ресурс] // Educative. – 2015. – Режим доступу до ресурсу: https://www.educative.io/blog/python-algorithms-coding-interview.

43. Базовые алгоритмы с помощью Python [Електронний ресурс] // Simple code. – 2014. – Режим доступу до ресурсу: https://shwanoff.ru/algorithms-python/.

44. What is hashing mean? [Електронний ресурс] // sqlserver. – 2020. – Режим доступу до ресурсу: https://searchsqlserver.techtarget.com/definition/hashing.

45. Security Assessment of Cryptographic Algorithms [Електронний ресурс] // Research Gate. – 2021. – Режим доступу до ресурсу: https://www.researchgate.net/publication/318924761_Security_Assessment_of_Cryptographic_Algorithms.

46. Криптоаналіз [Електронний ресурс] // Вікіпедія. – 2021. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Криптоаналіз.

47. Ахо А. В. Структуры данных и алгоритмы: Пер. с англ.: Уч. пос. / А. В. Ахо, Д. Хопкрофт, Д. Д. Ульман. – М.: Издательский дом "Вильямс", 2007. – 400 с.

48. Breslow L. A. Simplifying decision trees: a survey / L. A. Breslow, D. W. Aha. // Knowledge Engineering Review. – 1997. – №12(1). – P. 1–40.

49. Архитектура клиент-сервер [Електронний ресурс] – Режим доступу до ресурсу: https://sergeygavaga.gitbooks.io/kurs-lektsii-testirovanie-programnogo-obespecheni/content/lektsiya-6-ch1-arhitektura-klient-server.html.

50. Клиент-серверная архитектура в картинках [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/495698/.

51. Що таке комплексна система захисту інформації (КСЗІ) [Електронний ресурс] // AlterSign. – 2020. – Режим доступу до ресурсу: http://altersign.com.ua/korysna-informacija/pobudova-kszi/shcho-take-kompleksna-systema-zahystu-informaciji-kszi

52. ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ [Електронний ресурс] // Searching form. – 2021. – Режим доступу до ресурсу: https://searchinform.ru/informatsionnaya-bezopasnost/.

53. Пошук простих чисел [Електронний ресурс] // Mech math. – 2020. – Режим доступу до ресурсу: http://mech.math.msu.su/~shvetz/54/inf/perl-problems/chPrimes_sIdeas.xhtml.

54. Розширений алгоритм Евкліда [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: http://e-maxx.ru/algo/export_extended_euclid_algorithm.

55. Калькулятор алгоритму Евкліда [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://planetcalc.ru/3298/.

56. Криптосистема Рабина - Rabin cryptosystem [Електронний ресурс] – Режим доступу до ресурсу: https://ru.xcv.wiki/wiki/Rabin_cryptosystem.

57. Радченко М. 1С: Программирование для начинающих Детям и родителям, менеджерам и руководителям Разработка в системе "1С:Предприятие 8.3" / Максим Радченко. – Москва: ООО "1С-Паблишинг", 2016. – 780 с

58. Miller–Rabin primality test [Електронний ресурс] – Режим доступу до ресурсу:https://en.wikipedia.org/wiki/Miller%E2%80%93Rabin_primality_text=The%20Miller%E2%80%93Rabin%20primality%20test,the%20Solovay%E2%80%93Strassen%20primality%20test...

59. Китайська теорема про залишки [Електронний ресурс] – Режим доступу до ресурсу: http://4ua.co.ua/mathematics/qb3ac69a4d43a89421316d36_0.html.

60. Блочні шифри [Електронний ресурс] – Режим доступу до ресурсу: https://studref.com/403682/informatika/blochnye_shifry.

61. ОСНОВЫ БЛОЧНОГО ШИФРОВАНИЯ [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://www.delphiplus.org/zashchita-informatsii-v-telekommunikatsionnykh-sistemakh/osnovy-blochnogo-shifrovaniya.html.

62. Шифры перестановки [Електронний ресурс] – Режим доступу до ресурсу: http://citforum.ru/security/cryptography/yaschenko/78.html.

63. Основи шифрування [Електронний ресурс] – Режим доступу до ресурсу: https://sites.google.com/site/anisimovkhv/learning/kripto/lecture/tema

64. Шифр Цезаря или как просто зашифровать текст [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/534058/.

65. Иллюстрация работы RSA на примере [Електронний ресурс] – Режим доступу до ресурсу: http://www.michurin.net/computer-science/rsa.html.

66. Алгоритм шифрування RSA [Електронний ресурс] – Режим доступу до ресурсу: http://enisey.name/umk/pzis/ch18s07.html.

67. Шифр Роджера Уильямса XVII века взломан [Електронний ресурс] – Режим доступу до ресурсу: https://xakep.ru/2012/12/09/59778/.

68. Факторизация методом Ферма [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://foxford.ru/wiki/informatika/faktorizatsiya-metodom-ferma.

69. Шифрування гамуванням [Електронний ресурс] – Режим доступу до ресурсу: https://sites.google.com/site/anisimovkhv/learning/kripto/lecture/tema6

70. ШИФРОВАНИЕ НА ОСНОВЕ МЕТОДОВ ПОДСТАНОВКИ [Електронний ресурс] – Режим доступу до ресурсу: https://studref.com/403677/informatika/ shifrovanie_osnove_metodov_podstanovki.

71. Родригес А. Web-сервисы RESTful: основы / Родригес. // IBM Developer Works. – 2015. – С. 1–10.

72. Як працює HTTPS [Електронний ресурс] – Режим доступу до ресурсу: https://developers.google.com/search/docs/advanced/security/https?hl=ru.

73. Документація щодо використання сокетів [Електронний ресурс] – Режим доступу до ресурсу: https://digitology.tech/docs/python_3/library/socket.html

74. Захист інформаційних ресурсів [Електронний ресурс] – Режим доступу до ресурсу: https://vfranchuk.fi.npu.edu.ua/images/files/statty/32_ZIR_cript.pdf.

75. Tkinter. Бібліотека для створення графічних інтерфейсів у Python [Електронний ресурс] // Wikipedia – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Tkinter.

76. Кодування та декодування даних різного типу [Електронний ресурс] – Режим доступу до ресурсу: http://www.kievoit.ippo.kubg.edu.ua/kievoit/2016/38/index.html.

77. Обзор протокола http [Електронний ресурс] // Developer Zilla. – 2021. – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Web/HTTP/Overview.

78. ЗАХИСТ ПЕРСОНАЛЬНИХ ДАНИХ: ПРАВОВЕ РЕГУЛЮВАННЯ ТА ПРАКТИЧНІ АСПЕКТИ [Електронний ресурс] – Режим доступу до ресурсу: https://rm.coe.int/168059920c.

79. . Використання циклів у Python 3 [Електронний ресурс] – Режим доступу до ресурсу: https://python-scripts.com/loops-for-while.

80. Криптографія. Історія шифрувальної справи [Електронний ресурс] – Режим доступу до ресурсу: https://rostec.ru/news/kriptografiya-istoriya-shifrovalnogo-dela/.

81. Історія криптографії [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/КриптографічнаІсторія.

82. ПЕРЕХОПЛЕННЯ МЕРЕЖЕВОГО ТРАФІКА. МЕРЕЖЕВИЙ СНІФІНГ [Електронний ресурс] // Інформаційно комунікаційні технології. – 2021. – Режим доступу до ресурсу: https://e-journals.npu.edu.ua/index.php/ikt/article/view/65.

83. ОГЛЯД ПРОГРАМНИХ ЗАСОБІВ ДЛЯ АНАЛІЗУ МЕРЕЖЕВОГО ТРАФІКУ [Електронний ресурс] // Левчук А.С. – 2020. – Режим доступу до ресурсу: https://core.ac.uk/download/pdf/233567941.pdf.

84. АНАЛІЗ ТРАФІКУ КОМП'ЮТЕРНОЇ МЕРЕЖІ НА ОСНОВІ ЕКСПЕРИМЕНТАЛЬНИХ ДАНИХ СЕРЕДОВИЩА WIRESHARK [Електронний ресурс] // І. М. Дзюба, О. Ю. Марценкевіч. – 2021. – Режим доступу до ресурсу: http://science.lpnu.ua/sites/default/files/journal-paper/2017/may/2619/814ism2015min-55-62.pdf.

85. Все об атаке "Человек посередине" (Man in the Middle, MitM) [Електронний ресурс] // Олег Іванов. – 2021. – Режим доступу до ресурсу: https://www.anti-malware.ru/analytics/Threats_Analysis/man-in-the-middle-attack.

86. Як захиститись від атаку типу "Людина посередені" [Електронний ресурс] // Tcinet. – 2016. – Режим доступу до ресурсу: https://tcinet.ru/press-centre/glossary/article.php?ELEMENT_ID=5219.

87. КРИПТОГРАФІЯ: ЗАГАЛЬНІ ВИЗНАЧЕННЯ, КЛАСИФІКАЦІЯ.АСИМЕТРИЧНІ ТА СИМЕТРИЧНІ КРИПТОАЛГОРИТМИ, ЇХ

ПОРІВНЯННЯ. [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://dehtyarov09.wordpress.com/2014/03/16/.

88. ОБЩЕЕ ОПИСАНИЕ КРИПТОАЛГОРИТМА AES [Електронний ресурс] – Режим доступу до ресурсу: https://bit.nmu.org.ua/ua/student/metod/cryptology/AES.pdf.

89. Сравнение симметричного и асимметричного шифрований [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://academy.binance.com/ru/articles/symmetric-vs-asymmetric-encryption.

90. Криптоаналіз шифрувальної машина Енігма [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Криптоаналіз_Енігми.

91. ADVANCED ENCRYPTION STANDARD (AES) [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

ДОДАТОК А

Лістинг алгоритму Рабіна

```
import random
from Crypto.Util.number import bytes_to_long, long_to_bytes


class RabinCipher:
    p = 0
    q = 0
    n = p * q
    prime_digits = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29,
                    31, 37, 41, 43, 47, 53, 59, 61, 67,
                    71, 73, 79, 83, 89, 97, 101, 103,
                    107, 109, 113, 127, 131, 137, 139,
                    149, 151, 157, 163, 167, 173, 179,
                    181, 191, 193, 197, 199, 211, 223,
                    227, 229, 233, 239, 241, 251, 257,
                    263, 269, 271, 277, 281, 283, 293,
                    307, 311, 313, 317, 331, 337, 347, 349]

    def _egcd(self, a, b):
        """
        Extended GCD (Euklid algorithm)
        """
        x, y, u, v = 0, 1, 1, 0
        while a != 0:
            q, r = b // a, b % a
            m, n = x - u * q, y - v * q
```

```
        b, a, x, y, u, v = a, r, u, v, m, n
    return x, y

def _encryption_choice(self, array):
    if RabinCipher.digit_message in array:
        return RabinCipher.digit_message
    else:
        raise ValueError

def encrypt(self, message):
    """
    Encryption
    """
    RabinCipher.p, RabinCipher.q = RabinCipher._get_primes(self)
    RabinCipher.n = RabinCipher._open_key(self, RabinCipher.p, RabinCipher.q)
    RabinCipher.digit_message = bytes_to_long(message.encode('utf-8'))
    RabinCipher.result = pow(RabinCipher.digit_message, 2) % RabinCipher.n
    return RabinCipher.result

def decrypt(self, c):
    """
    Decryption
    """
    if not RabinCipher.p or not RabinCipher.q or not RabinCipher.n:
        raise ValueError
    RabinCipher.c = c
    a, b = RabinCipher._egcd(self, RabinCipher.p, RabinCipher.q)
    m1 = pow(RabinCipher.c, (RabinCipher.p + 3) // 4, RabinCipher.p)
    m2 = pow(RabinCipher.c, (RabinCipher.q + 3) // 4, RabinCipher.q)
    x = (m1 * b * RabinCipher.q + m2 * a * RabinCipher.p) % RabinCipher.n
```

```
    y = (m1 * b * RabinCipher.q - m2 * a * RabinCipher.p) % RabinCipher.n
    array = [x, RabinCipher.n - x, y, RabinCipher.n - y]
    long_text = RabinCipher._encryption_choice(self, array)
    bytes_text = long_to_bytes(int(long_text))
    result_text = bytes_text.decode('utf-8')
    return str(result_text)

def _open_key(self, p, q):
    """
    Open key - n
    n = p * q
    """
    self.p = p
    self.q = q
    print(type(self.p * self.q))
    return self.p * self.q

def _get_primes(self):
    """
    Generating two large prime digits
    p is first large prime number
    q is second large prime number
    These digits are used like secret keys
    Digits must be checked by tests
    :return: (p, q)
    """

    def getting_random(n):
        return random.randrange(2 ** (n - 1) + 1, 2 ** n - 1)
```

```
def divide_test_low_primes(n):
    while True:
        candidate = getting_random(n)
        for division in RabinCipher.prime_digits:
            if candidate % division == 0 and division ** 2 <= candidate:
                break
        else:
            return candidate

def test_of_miller_rabin(mrc):
    maxDivisionsByTwo = 0
    numberOfRabinTrials = 2000
    ec = mrc - 1
    while ec % 2 == 0:
        ec >>= 1
        maxDivisionsByTwo += 1
    assert (2 ** maxDivisionsByTwo * ec == mrc - 1)

    def trialComposite(round_tester):
        if pow(round_tester, ec, mrc) == 1:
            return False
        for i in range(maxDivisionsByTwo):
            if pow(round_tester, 2 ** i * ec, mrc) == mrc - 1:
                return False
        return True

    for i in range(numberOfRabinTrials):
        round_tester = random.randrange(2, mrc)
        if trialComposite(round_tester):
            return False
```

```
            return True

        result = []

        while len(result) != 2:
            while True:
                n = 512
                prime_candidate = divide_test_low_primes(n)
                if not test_of_miller_rabin(prime_candidate):
                    continue
                else:
                    if (prime_candidate % 4) == 3:
                        result.append(prime_candidate)
                        break
                    else:
                        continue
        return result


if __name__ == '__main__':
    cipher = RabinCipher()
    user_input = input(str("Please enter the message: "))
    print("Your message: {}".format(user_input))
    new_message = cipher.encrypt(user_input)
    print('Encrypted message: {}'.format(new_message))
    result = cipher.decrypt(new_message)
    print('Decrypted message: {}'.format(result))
```

ДОДАТОК Б

Лістинг клієнтів Bob та Alisa

Клієнт Alisa

```
from main import RabinCipher
import socket

sock = socket.socket()
sock.bind(('', 8002))
sock.listen(1)
conn, addr = sock.accept()

while True:
    data = conn.recv(1024)
    if data:
        from tkinter import *

        def send_message():
            content = input_field.get(1.0, "end-1c")
            cipher = RabinCipher()
            RabinCipher.n = int(data.decode('utf-8'))
            encrypted_text = RabinCipher.encrypt(cipher, content)
            output_field.insert(1.0, "Encrypted message: " + str(encrypted_text))
            conn.send(bytes(str(encrypted_text), encoding='utf-8'))



        window = Tk()
        window.title("Alisa")
```

```
        frame1 = Frame(master=window, width=1115, height=400)
        frame1.pack()
        btn_send = Button(text="Send", width=91, height=3, font='Times 18', bg="grey", fg="white", command=send_message)
        input_field = Text(master=frame1, width=50, height=30, font='Times 16', bg="white", fg="black")
        output_field = Text(master=frame1, width=50, height=30, font='Times 16', bg="white", fg="black")
        input_field.place(x=0, y=0)
        output_field.place(x=560, y=0)
        btn_send.pack(side=BOTTOM, expand=True)
        window.mainloop()

    if not data:
        break

conn.close()
```

Клієнт Bob

```
from rabins_cipher import RabinCipher
import socket
from tkinter import *


def init():
    cipher = RabinCipher()
    RabinCipher.p, RabinCipher.q = cipher._get_primes()
    open_key = RabinCipher.n = cipher._open_key(RabinCipher.q, RabinCipher.p)
```

```
    input_field.insert(1.0, "Generated open key for Alisa: " + str(open_key))
    sock = socket.socket()
    sock.connect(('localhost', 8002))
    sock.send(bytes(str(open_key), encoding='utf-8'))
    data = sock.recv(2048)
    sock.close()
    result = RabinCipher.decrypt(cipher, int(data.decode('utf-8')))
    output_field.insert(1.0, "Message from Alisa: " + str(result))
    return result


window = Tk()
window.title("Bob")
frame1 = Frame(master=window, width=1115, height=400)
frame1.pack()
btn_send = Button(text="Init connection", width=91, height=3, font='Times 18', bg="grey", fg="white", command=init)
input_field = Text(master=frame1, width=50, height=30, font='Times 16', bg="white", fg="black")
output_field = Text(master=frame1, width=50, height=30, font='Times 16', bg="white", fg="black")
input_field.place(x=0, y=0)
output_field.place(x=560, y=0)
btn_send.pack(side=BOTTOM, expand=True)

window.mainloop()
```

ДОДАТОК В

Лістинг програми графічного інтерфейсу

```
from tkinter import *

window = Tk()
window.title("Main")
frame1 = Frame(master=window, width=700, height=400)
frame1.pack()
btn_send = Button(text="Send", width=56, height=3, font='Times 18', bg="grey", fg="white")
input_field = Text(master=frame1, width=50, height=30, font='Times 16', bg="white", fg="black")
output_field = Text(master=frame1, width=50, height=30, font='Times 16', bg="white", fg="black")
input_field.place(x=0, y=0)
output_field.place(x=360, y=0)
btn_send.pack(side=BOTTOM, expand=True)

window.mainloop()
master = Tk()
master.title("Система для передачи сообщений")
w = master.winfo_screenwidth()
h = master.winfo_screenheight()
w = w // 2  # середина экрана
h = h // 2
w = w - 200  # смещение от середины
h = h - 200
master.geometry('2000x1500+{}+{}'.format(w, h))
```

```
body = Text(master, bg="white", font='Times 11', fg='black')
body.pack(expand=True, fill=BOTH, side=LEFT)
body2 = Text(master, bg="white", font='Times 11', fg='black')
body2.pack(expand=True, fill=BOTH, side=LEFT)
master.mainloop()
```