Галицький фаховий коледж імені В’ячеслава Чорновола

відділення комп’ютерних та видавничих технологій

циклова комісія інформатики та комп’ютерних дисциплін

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач відділенням
комп’ютерних та видавничих
технологій
Чубей О.О. /_____/
підпис
«___» _____2022 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проєкту
освітньо-кваліфікаційного рівня «молодший спеціаліст»
зі спеціальності 122 «комп’ютерні науки»
на тему : «Програмний засіб шифрування файлів алгоритмом RSA»

Студент групи КН-41 Гінзула В.Я. _____
(підпис)

Керівник проєкту Івасьєв С.В. _____
(підпис)

Консультанти:

з техніко-економічного обґрунтування Меленчук Л.І _____
(підпис)

нормоконтролер Гавришків Н.Г. _____
(підпис)

Тернопіль – 2022

Галицький фаховий коледж імені В’ячеслава Чорновола

відділення комп’ютерних та видавничих технологій

циклова комісія інформатики та комп`ютерних дисциплін

ЗАТВЕРДЖУЮ
Завідувач відділенням
комп’ютерних та видавничих
технологій
Чубей О.О. /_____/
підпис
«___»_____ 2021 р.

ЗАВДАННЯ

на дипломне проєктування
на здобуття освітньо-кваліфікаційного рівня «молодший спеціаліст»
студенту Гінзули Володимира Ярославовича
(прізвище, ім’я та по-батькові студента)

1. Тема проєкту _____
_____
_____
_____
затверджена наказом по коледжу від “____” _____2021 р., № _____
2. Термін здачі студентом завершеного проєкту “_____” _____2022 р
3. Вихідні дані до проєкту

4.Перелік питань, які повинні бути розроблені в проєкті: _____
а)основна частина_____
_____
_____
_____
б)техніко-економічного обґрунтування_____
_____
_____
_____
_____
5.Перелік графічного матеріалу_____
_____
6.Консультанти проєкту: _____
_____

Розділ	Консультанти	Підпис, дата	
		Завдання видано	Завдання прийнято
з техніко-економічного обґрунтування	_____ (вчена ступень, звання П.І.Б. _____ консультанта)		

КАЛЕНДАРНИЙ ПЛАН
дипломного проєктування

№ п/п	Найменування етапу	Терміни	
		початку	завершення
1.	Вибір теми, ознайомлення з вимогами до дипломного проєктування.	29.09.21 р.	01.10.21 р.
2.	Огляд типових рішень та написання відповідного розділу ПЗ	7.10.21 р.	13.10.22 р.
3.	Дослідження технологій реалізації та написання відповідного розділу ПЗ	19.10.22 р.	25.10.22 р.
4.	Розробка функціональних вимог до проєкту та робота над структурою програмного продукту. Написання відповідного розділу ПЗ	15.11.22 р.	22.11.22 р.
5.	Встановлення на налаштування середовища реалізації та написання відповідного розділу ПЗ	02.12.22 р.	16.012.22 р.
6.	проєктування програмного засобу (функціоналу, інтерфейсу, бази даних продукту) та написаннявідповідного розділу ПЗ	16.01.22 р.	17.02.22 р.
7.	Реалізація та налаштування програмного засобу та написання відповідного розділу ПЗ	17.03.22 р.	30.03.22 р.
8.	Доопрацювання модулів	05.04.22 р.	15.04.22 р.
9.	Тестування на налагодження програмного продукту та написання відповідного розділу ПЗ	18.04.22 р.	21.04.22 р.
10.	Опрацювання економічного розділу дипломного проєкту та оформлення спеціального розділу	20.05.22 р.	25.05.22 р.
11.	Робота над оформленням пояснювальної записки	03.06.22 р.	12.06.22 р.
12.	Попередній захист дипломного проєкту, доопрацювання	15.06.22 р.	
13.	Підготовка до захисту дипломного проєкту	18.06.22 р.	22.06.22 р.
14.	Захист дипломного проєкту	25.06.22 р.	

7. Дата видачі завдання ”____” _____ 2021р.
Керівник _____ _____
Завдання прийняв до виконання _____/

Реферат

Дипломний проєкт. Тема: «Програмний засіб шифрування файлів алгоритмом RSA». 65сторінок, 35 рисунки, 5 таблиці, 8 джерел, 2 додатки, 2 блок-схема.

Об’єктом дослідження є шифрування файлів та захист інформації. Розглянуто область діяльності та сфери застосування таких програм. Мета проєкту – збереження особистих даних користувача в безпеці від не бажаних осіб. Розробка програмної логіки та створення зручного інтерфейсу користувача. Під час розробки було розглянуто, асиметричні та симетричні криптосистеми. Проведено дослідження алгоритму RSA як стандарту асиметричного шифрування. Завдання проєкту є розробка програми шифрування для збеження ціної інформації в шифрованих файлах.Результат проєкту – є програма для шифруквання файлів алгоритмом RSA.

Для розробки програми було обрано середовище розробки RAD studio 11, середовище спеціалізується на створенні програмних засобів для компютерів на базі операційної системи Windows. RAD studio 11 підтримує мови програмування: C, C++. Реалізація програми буде виконуватись на мові програмування C++, де зрозумілий синтаксис, велика кількість програмної документації та середовище для розробки програмного інтерфейсу.

Призначення програмного засобу: шифрування та дешифрування фалів агоритмом RSA, іпорт та експорт закритого та відкритого ключа, та екпортування шифрованих фйлів, легкий інтерфейс для користування програмою.

ШИФРУВАННЯ ФАЙЛІВ, АСИМЕТРИЧНЕ, СИМЕТРИЧНЕ ШИФРУВАННЯ, RSA, RAD STUDIO 11, C++.

Abstract

Diploma project. Topic: "Software encryption software with RSA algorithm". 65 pages, 35 figures, 5 tables, sources, 2 appendices, 2 block diagrams.

The object of research is file dissemination and information protection. The scope and scope of such programs are considered. The purpose of the project is to keep the user's personal data safe from unwanted people. Development of software logic and creation of a user-friendly interface. Asymmetric and symmetric keys were considered during the development. The research of RSA algorithm as a standard of asymmetric encryption is carried out. The task of the project is to develop an encryption program to avoid the value of information in encrypted files. The result of the project is a program for encrypting files with the RSA algorithm. Easy-to-use interface and well-designed logical part of the program.

The RAD studio 11 development environment was chosen for the development of the program, this program specializes in creating software for computers based on the Windows operating system. RAD studio 11 supports the following programming languages: C, C ++. The program will be implemented in the C ++ programming language, there is a clear syntax, a large amount of software documentation and an environment for developing the software interface.

The function of the program is encryption and decryption of files by RSA algorithm, import and export of private and public keys, and export of encrypted files, easy interface to use the program.

FILE ENCRYPTION, ASYMMETRIC, SYMMETRIC ENCRYPTION, RSA, RAD STUDIO 11, C ++.

ЗМІСТ

ВСТУП

У наш час захист інформаційних технологій є дуже потрібним. Для того щоб зберегти особисті данні в цілісності, потрібно забезпечити захист інформації. Інформація, процеси, що живлять її мережеву структуру та інформаційні системи – це величезний актив, в якого є своя цінність і він повинен бути добре захищений. Після появи комп'ютерних технологій в будь – якій сфері людської діяльності, обсяг інформації істотно збільшився. І зараз для того щоб скопіювати данні на флеш накопичувач, що містить данні виробництва чи список працівників не займе багато часу. А з урахуванням стану розвитку сучасних інформаційних технологій рівень захисту інформаційних потоків є безумовно актуальною задачею.

Обмін даними, які є для загального користування та які є приватними та спільне використання інформаційних технологій ускладнює процес доступу до інформації. Використання систем розділеної обробки даних послаблює централізований контроль.

На сьогоднішній день для систем захисту інформації широко використовуються асиметричні криптосистеми (RSA, Ель – Гамаля, Криптосистема Рабіна), що базуються на обчислювальних складностях процесів факторизації, дискретного логарифмування, знаходженні точки на еліптичній кривій.

Метою даного проєкту є створення програмного забезпечення яке спроможне захистити інформацію та унеможливити доступ до неї не бажаним особам методом шифрування цих файлів. Ця програма буде шифрувати та розшифровувати файли для тих людей які мають на це право доступу (секретний ключ).

Поставленим завдання дипломного проєктування є створення програмного засобу для шифрування та розшифрування файлів алгоритмом RSA.

Метою роботи є дослідження сучасної асиметричної криптосистеми та розробка програмного засобу для шифрування і дешифрування файлів.

Об’єктом дослідження є процеси шифрування та дешифрування алгоритмом RSA.

Предметом дослідження є існуючі засоби розробки програмних засобів шифрування та дешифрування алгоритмом RSA.

Реалізація поставленої мети передбачає вирішення таких завдань:

– Аналіз предметної області.

– Дослідити інфраструктуру відкритих та закритих ключів для асиметричних криптосистем.

– Проаналізувати симетричні алгоритми.

– Проаналізувати алгоритм асиметричного шифрування RSA.

– Виконати проєктуваня та реалізацію програмного забезпечення.

– Виконати техніко – економічного обґрунтування роботи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАННЯ СИСТЕМА ШИФРУВАННЯ ФАЙЛІВ АЛГОРИТМОМ RSA

1.1 Постановка завдання

Проблематикою створення цього проєкту є захист та цілісність інформації яка є дуже важлива у наш час. Створення цієї системи збільшить шанси на захист персональних даних, цінних паперів, розробок тощо. Перевагою буде легкість у використанні та достатньо надійний захист інформації.

Шифрування буде відбуватись в декілька натискань, «Завантажити файл», «зашифрувати файл», «Зберегти зашифрований файл», «отримати ключ для розшифрування файлу» дуже просто, програма буде легка у використанні. Перевага обраного способу шифрування в тому є його маштабованість, у ключах допускається будь який їхній розмір: 768 – бітний, 1024- бітний, 2048 – бітиний , 4096 – бітний і так далі[1].

Алгоритм реалізовано на простому математичному підході, тож його реалізація в структурі відкритих ключів є набагато легшою. Адаптивність і безпека зробили RSA найбільше використовуваним алгоритмом шифрування для різних програмних застосунків , включаючи сертифікати, крипто валюти та шифрування електронних скриньок.

Недоліком є те що ви можете якимось способом втрати ключ чи сам зашифрований файл який ви потім не зможете розшифрувати. Ще одим недоліком який є в цій системі це то що в нього відносно не висока швидкість роботи.

Ціллю цієї роботи є створення застосунку для загального користування для забезпечення повного захисту інформації. Забезпечення перетворення інформації яка зрозуміла для звичного читання в ту яку не в змозі прочитати без програмного забезпечення і обернене відновлення інформації у одержувача, створюючи не можливим для читання тих осіб які перехопили

чи прослухують канал зв'язку, без секретного ключа. Сфера застосування шифрування збільшилась і включає в себе не лише засекречену передачу інформації , а і різні методи перевірки цілісності повідомлень, розпізнавання відправника або отримувача, цифрового підпису, інтерактивне підтвердження та технології захищеного обміну даними.

Поява комп'ютерів та електроніки зробила можливою появу складних шифрів. Перевагою комп'ютера стало те що він дозволяв шифрувати будь – які дані, які можуть бути представленні на комп'ютері у двійковому виді. Більшість комп'ютерних шифрів можна прослідкувати в роботі бінарних бітів. Та після появи комп'ютерів шифрування стало набагато безпечнішим. Сучасні шрифти залишились по переду криптоаналізу, використання цих шифрів є ефективним механізмом, яке не вимагає високої затрати ресурсів, але взлам цих ресурсів потребує більших зусиль.

Шифрування робить набагато більше ніж просто захист даних від сторонніх очей. Воно також може бути використане для доказу цілісності та достовірності інформації, використовуючи цифрові підписи. Шифр – це важлива частина керування числовими правами та захист від створення копій. Його можна використовувати для стирання даних. Так як видалену інформацію в деяких випадках можна отримати за допомогою пристрої відновлення даних, якщо на початку зашифрувати дані та викинути ключ, єдине що можна відновити це шифротекст а не початкові данні.

Більша частина підключень, які здійснюється з основними веб застосунками, будуть закодовані TSL , вказаним HTTPS або замком в URL – адресі браузера. Будь які повідомлення в месенджерах також зашифровані, і також можна мати зашифровану папку на вашому мобільному телефоні. Електронні листи захищенні протоколами OpenPGP. VPN також використовує шифрування і все що зберігається у хмарі повинно бути зашифроване. Також можна зашифрувати жорсткий диск на комп'ютері і створити зашифровані голосові дзвінки[2].

Велика кількість систем зв’язку використовують шифрування для захисту наших персональних даних подалі від небажаних осіб. Шифрування є ключовим аспектом забезпечення кошельків крипто валюти, важливою частиною мережі є захист системою Tor і це теж використовується у багатьох технологіях.

1.2 Інфраструктура відкритих ключів

Інфраструктура відкритих ключів (ІВК) базується на асиметричної криптографії (криптографія з відкритими ключами). Асиметрична криптографія, як розділ науки криптографії з'явилася в кінці 70-х років 20-го століття. В даний час в системах захисту інформації широко використовуються, як симетрична, так і асиметрична криптографії.

Процес криптографічного перетворення (шифрування) інформації виглядає наступним чином. Відкритий текст (інформацію, яку потрібно зашифрувати) шифрують за допомогою певного криптографічного алгоритму і ключа шифрування. Зашифрований по надійному криптографічним алгоритмом текст практично неможливо розшифрувати без додаткових даних, які називаються ключем розшифрування.

1.3 Симетричні алгоритми

Криптографія з симетричним, або секретним, ключем використовує однакові ключі для шифрування і розшифрування повідомлень. Ключ цей знають тільки відправник і адресат, він не повинен бути відомий третій особі. Тому головна проблема симетричної криптографії полягає в попередній передачі секретного ключа одним абонентом іншому по надійному каналу. Але для цього потрібно зберігати багато ключів для різних користувачів[3].

Існує величезна різноманітність конкретних реалізацій алгоритмів шифрування симетричними ключами. Найбільшого поширення набув алгоритм DES (Data Encryption Standard), прийнятий національним бюро

стандартів США в 1977 році. У 1991 році аналогічний алгоритм був прийнятий в якості вітчизняного стандарту (ДСТУ 28147-89). Певне поширення набули також алгоритми RC4, RC5, IDEA та ін.

1.4 Використання асиметричних алгоритмів

В алгоритмах цього типу для шифрування і розшифровки інформації використовуються пара ключів: відкритий і закритий, кожен з яких не може бути отриманий з іншого. Відкритий ключ розсилається всім абонентам, закритий тримається в таємниці.

Для того щоб відправити повідомлення абоненту, потрібно при шифруванні використовувати його відкритий ключ, одержувач ж розшифровує повідомлення за допомогою свого закритого секретного ключа. Ніхто, крім одержувача, не може розшифрувати повідомлення, так як ніхто більше не має доступу до цього закритого ключа. Навіть той, хто зашифрував повідомлення за допомогою відкритого ключа, не зможе його розшифрувати. Такий протокол забезпечує приватність без необхідності володіння надійним каналом.

У 80-х роках основним симетричним криптоалгоритмом для внутрішнього застосування в США був DES (Data Encryption Standard). Однак уже в 90-х роках стали виявлятися його основні недоліки. Головним з яких була довжина ключа, складова 56 біт. Такий розмір ставав недостатньо великим зважаючи на постійне зростання продуктивності ЕОМ, так як ключ міг бути розкритий шляхом простого перебору всіх можливих варіантів шифрування. При використанні алгоритму з відкритим ключем не має потреби в секретному каналі для передачі ключа, через те що відкритий ключ не є секретної інформацією.

Різниця ключів – відкритого і закритого – в криптографії з відкритими ключами дозволило створити такі технології:

– електронні цифрові підписи (завдання забезпечення цілісності, авторства, актуальності інформації, автентифікації суб'єкта та інформації, неспростовності);

– розподілена перевірка справжності (завдання ідентифікації, аутентифікації суб'єкта, авторизація доступу суб'єкта до інформації);

– узгодження загального секретного ключа сесії (завдання забезпечення конфіденційності інформації при передачі по відкритих каналах зв'язку);

– шифрування великих обсягів даних без попереднього обміну загальним секретним ключем (Завдання забезпечення конфіденційності інформації).

Деякі алгоритми, наприклад RSA (Rivest – Shamir – Adleman) і ECC (EllipticCurve Cryptography), універсальні, вони підтримують всі перераховані вище операції. Інші алгоритми більш спеціалізовані і підтримують не всі можливості.

До числа алгоритмів шифрування з відкритим ключем відносяться:

– алгоритм електронного цифрового підпису DSA;

– алгоритм DH (Diffie – Hellman), застосовуваний для вироблення загального секретного ключа сесії.

В алгоритмах криптографії з відкритими ключами важливим аспектом є визначення приналежності конкретного відкритого ключа конкретному користувачеві. У більшості випадках відкриті ключі користувачів зберігаються в зальному довіднику відкритих ключів, і існує ймовірність що можуть його перехопити. Для цього потрібен механізм який може забезпечити впевненість в тому що відкритий ключ належить відповідному користувачеві. Один з механізмів таких ключів заснованих на сертифікатах відкритих ключів[4].

Сертифікати відкритого ключа забезпечують механізм надійного зв'язку між відкритим ключем і суб'єктом, якому належить відповідний закритий ключ.

Сертифікат – це цифровий документ, який містить відкритий ключ суб'єкта і підписаний електронним цифровим підписом засвідчує центру видав сертифікат. Сертифікат також містить відомості про власника відкритого ключа, наприклад, інформацію, яка його додатково ідентифікує.

В даний час найбільш часто використовуються сертифікати на основі стандарту Міжнародного союзу телекомунікацій ITU-T X.509 v3 і рекомендацій IETF (Internet Engineering Task Force) RFC 2459.

Центр засвідчення – це служба, яка видає сертифікати. Засвідчує Центр є гарантом зв'язку між відкритим ключем суб'єкта і що міститься в сертифікаті інформацією щодо ідентифікації цього суб'єкта. Різні ЦС встановлюють і гарантують цей зв'язок різними способами, тому перш ніж довіряти сертифікатам того чи іншого ЦС, слід ознайомитися з його політикою і регламентом. Засвідчують центри є однією з основних складових при побудові систем захисту в інформаційній системі з істотно розподіленою структурою.

1.5 Схема підпису електронним цифровим ключем

Протоколи ЕЦП з одного боку відносять до протоколів аутентифікації, тому що гарантують, що повідомлення надійшло від достовірного відправника, а з іншого боку до протоколів контролю цілісності, тому що гарантують, що повідомлення прийшло в неспотвореному вигляді. Більш того, одержувач в подальшому може використовувати ЕЦП як доказ достовірності повідомлення третім особам (арбітру) в тому випадку, якщо відправник згодом спробує відмовитися від нього

Електронний цифровий підпис – реквізит електронного документа, призначений для захисту даного документа від підробки, отриманий в результаті криптографічного перетворення інформації з використанням закритого ключа ЕЦП і що дозволяє ідентифікувати власника сертифіката

ключа підпису, а також встановити відсутність спотворення інформації в електронному документі.

Електронний цифровий підпис – рядок біт, отриманий в результаті процесу формування підпису (ISO / IEC 14888-1: 2008). За ДСТУ поняття "Електронний цифровий підпис", "електронний підпис" і "цифровий підпис" є синонімами.

Як видно з цієї схеми, порядок використання ключів зворотний тому, який використовується при передачі секретних повідомлень.

Спочатку відправник використовує свій закритий ключ, а потім одержувач застосовує відкритий ключ відправника (Таблиця 1.1).

Таблиця 1.1 – Схеми ЕЦП

Схема цифрового підпису	Завдання, що лежить в основі стійкості	Хеш функція
RSA	Розкладання числа на множники	MD4 або MD5 (Message Digest Algorithm - алгоритм короткого викладу повідомлення, Р. Ривест)
DSS (NIST [1] . FIPS Publication 186: Digital Signature Standard (DSS). May 1994) DSS - Федеральний стандарт цифрового підпису США	Дискретного логарифмування за схемою Ель-Гамаля	SHA-1 (NIST. FIPS Publication 180: Secure Hash Standard (SHS). May 1993) SHS - стандарт хеш-функції США SHA - Secure Hash Algorithm - алгоритм хеш-функції

Продовження таблиці 1.1

ECDSA (Elliptic Curve Digital Signature Algorithm) - алгоритм цифрового підпису на еліптичних кривих. Прийнятий як стандарт ISO [2] 14888-3 в 1998 р, ANSI [3] X9.62 - 1999 г., IEEE [4] 1363 - 2000 року і NIST 186-2 - 2000 р (остання редакція - NIST. FIPS Publication 186- 3: Digital Signature Standard (DSS). June 2009)	Дискретного логарифмування в групі точок еліптичної кривої	SHA (NIST. FIPS 180-3: Secure Hash Standard (SHS). October 2008)
ДСТУ 34.10-94 (Інформаційна технологія. Криптографічний захист інформації. Процедури вироблення і перевірки електронного цифрового підпису на базі асиметричного криптографічного алгоритму)	Дискретного логарифмування за схемою Ель-Гамаля	ДСТУ 34.11-94 (Інформаційна технологія. Криптографічний захист інформації. Функція хешування)
ДСТУ Р 34.10-2001 (Інформаційна технологія. Криптографічний захист інформації. Процеси формування та перевірки електронного цифрового підпису)	Дискретного логарифмування в групі точок еліптичної кривої	ДСТУ 34.11-94 (Інформаційна технологія. Криптографічний захист інформації. Функція хешування)
ДСТУ Р 34.10-2012 (Інформаційна технологія. Криптографічний захист інформації. Процеси формування та перевірки електронного цифрового підпису)	Дискретного логарифмування в групі точок еліптичної кривої	ДСТУ Р 34.11-2012 (Інформаційна технологія. Криптографічний захист інформації. Функція хешування)

Говорячи про схему електронного підпису, зазвичай мають на увазі наступну класичну ситуацію:

– відправник знає зміст повідомлення, яке він підписує;

– одержувач, знаючи відкритий ключ перевірки підпису, може перевірити правильність підпису отриманого повідомлення в будь-який час без будь-якого дозволу і участі відправника;

– безпеку схеми підпису гарантується.

На рисунку 1.1 приведено діаграму використання для системи шифрування RSA, яка буде використана при проєктуванні програмного засобу.

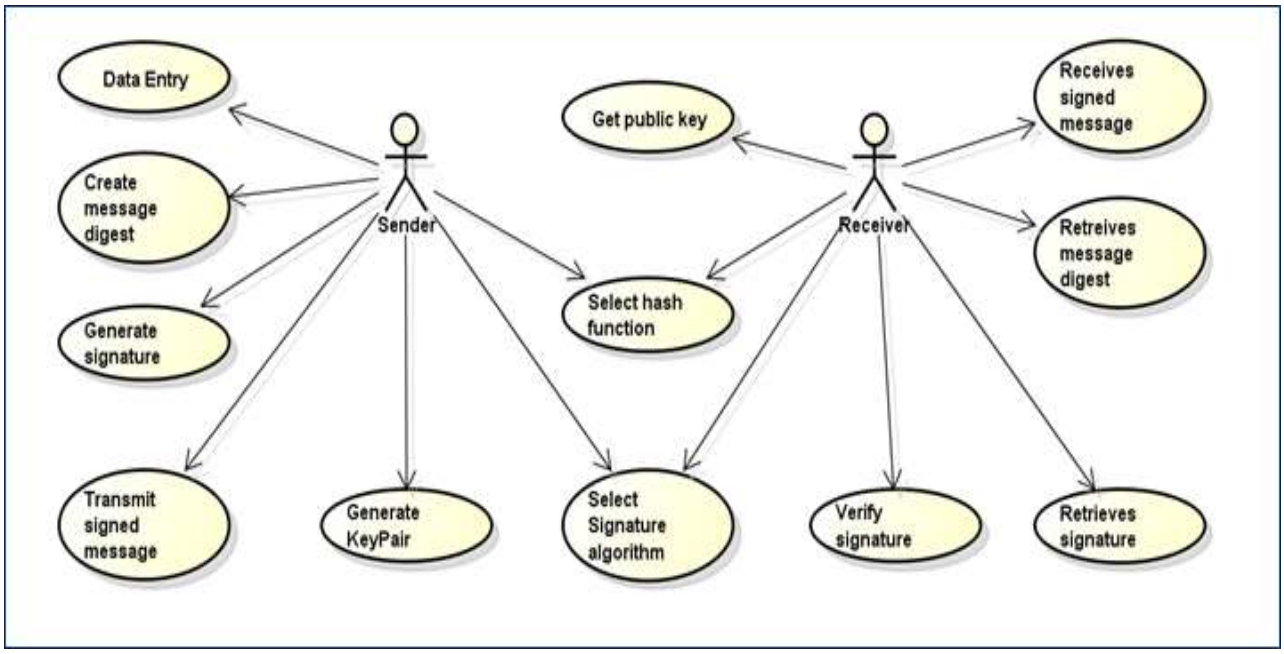

Рисунок 1.1 – Діаграма використання системи шифрування RSA

При створенні електронного підпису за класичною схемою відправник:

– застосовує до вихідного повідомлення T хеш функцію h (T) і отримує хеш-образ повідомлення *r*;

– обчислює електронний підпис *s* по хеш-образу *r* з використанням свого закритого ключа;

– посилає повідомлення *T* разом з електронним підписом *s* одержувачу;

– одержувач, відокремивши електронний підпис від повідомлення, виконує наступні дії:

– застосовує до отриманого повідомленням T хеш функцію h (T) і отримує хеш-образ повідомлення r;

– розшифровує хеш-образ r' з електронного підпису s з використанням відкритого ключа відправника;

– перевіряє відповідність хеш-образів r і r' і якщо вони збігаються, то відправник дійсно є тим, за кого себе видає, і повідомлення при передачі не піддалося спотворенню.

Було виконано проєктування функцій розроблюваного програмного засобу та аналіз можливостей асиметричних криптосистем. На сучасних асиметричних

криптосистемах базуються більшість сучасних криптосистем, а схеми обміну ключами мають багато схожих елементів.

Електронні цифрові підписи мають настільки важливу роль як і саме шифрування, оскільки цифрові підписи давно широкок використовуються в документообізі та прирівнюються до власних.

2 ДОСЛІДЖЕННЯ АЛГОРИТМУ RSA

2.1 Стандарт асиметричного шифрування RSA

Незабаром після появи ранцевого алгоритму Меркле-Хеллмана був створений перший повноцінний алгоритм із відкритим ключем, який можна використовувати і для шифрування та для створення цифрових підписів – алгоритм.RSA. Алгоритм RSA названий на честь трьох винахідників – Рона Рівеста, Аді Шаміра та Леонарда Адлемана і був запропонований 1978. Розробникам даного алгоритму вдалося ефективно втілити ідею односторонніх функцій із секретом. Стійкість RSA базується на складності факторизації великих цілих чисел. Відкритий та закритий ключі є функціями двох великих простих чисел розрядністю 100...200 десяткових цифр і навіть більше. Відновлення відкритого тексту за шифртекстом та відкритим ключем рівносильне розкладанню числа на два великі прості множники. Багато років алгоритм RSA протистоїть численним спробам криптографічного розтину. Криптоаналіз не доводить, не спростовує безпеку алгоритму RSA, тим самим обґрунтовуючи ступінь довіри до алгоритму. У 1993 році алгоритм RSA був прийнятий як стандарт (PKCS # 1: RSA EncryptionStandard).

Алгоритм RSA. Випадковим чином вибираються два великі прості числа p і q. Розраховується добуток $n = p\,q$.

Обчислюється функція Ейлера $\varphi(n) = (p-1)(q-1)$.

Випадковим чином вибирається просте число e – ключ шифрування яке відповідає умовам $e < \varphi(n)$; $(e, \varphi(n)) = 1$.

Обчислюється число d – ключом розшифрування ,є оберненим числу e, тощо.

$$e\,d \equiv 1 \pmod{\varphi(n)}$$

Пара чисел $(e,\ n)$ робиться відкритим ключем і поміщається в загальнодоступний довідник, а числа $p,\ q$ тримаються в секреті, d – секретний ключ. При шифруванні повідомлення M спочатку розбивається на цифрові

блоки розмірами менше n, тобто якщо p *і* q є 100-розрядними простими числами, то n міститиме близько 200 розрядів і кожен блок повідомлення m_i повинен мати близько 200 розрядів завдовжки. Зашифроване повідомлення C буде складатися з блоків c_i тієї ж самої довжини. Формула шифрування:

$$C \equiv M^e \pmod{n}.$$

Розшифровування забезпечується операцією зведення в ступінь d модулю n прийнятого шифртексту C:

$$M \equiv C^d \ (mod\ n)$$

так як $C^d \pmod{n} \equiv M^{ed} \pmod{n} \equiv (M^{k(p-1)(q-1)+1}) \bmod n \equiv (MM^{k(p-1)(q-1)}) \bmod (pq) \equiv M$, як це показано на рисунку 2.1.

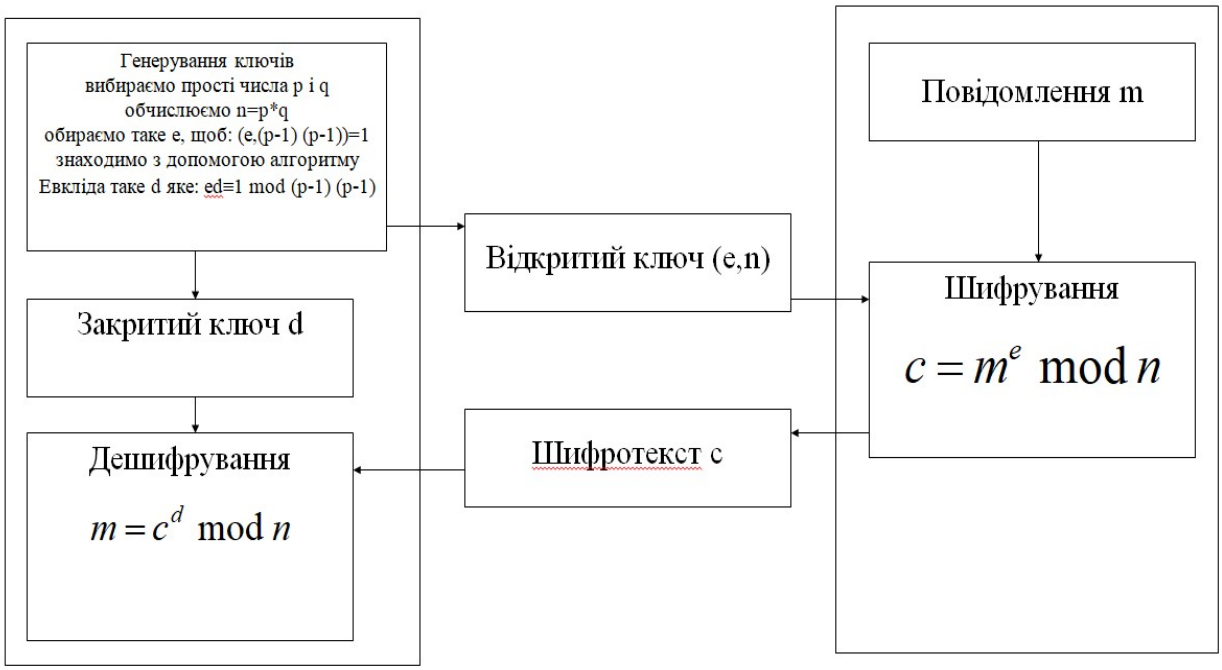

Рисунок 2.1 – Схема шифрування RSA

Для прикладу візьмемо числа p=11, q=5, M=15.

Обчислюємо $n = 11 \cdot 5 = 55$.

Визначаємо функцію Ейлера φ (55) = (11-1) (5-1) = 40.

Вибираємо ключ зашифровування $e = 7$, який відповідає умовам $7 < 40$; НОД (7, 40) = 1.

Визначаємо d – ключ розшифровування – з рівняння

$$7d \equiv 1 \pmod{40}.$$

Послідовність дій наведено в таблиці 2.1.

Таблиця 2.1 – Алгоритм RSA

Відкритий ключ: n - добуток двох простих чисел p і q (p і q повинні зберігатися в секреті); e – ключ зашифровування, число взаємно просте із функцією Ейлера $(e, \varphi(n)) = 1$ та $e < \varphi(n)$.
Закритий ключ: $d \equiv e^1 \pmod{\varphi(n)}$ – ключ розшифровування.
Шифрування: $C \equiv M^e (\text{Mod } n)$.
Розшифрування: $M \equiv C^d (\text{Mod } n)$.

Розглянемо кілька способів знаходження d.

Спосіб 1 Для розв'язку рівняння $7d \equiv 1 \pmod{40}$ використовуємоалгоритм Евкліда

$$40 = 7 \cdot 5 + 5;$$
$$7 = 5 \cdot 1 + 2;$$
$$5 = 2 \cdot 2 + 1;$$
$$2 = 1 \cdot 2 + 0.$$

Зворотнє підставлення дає :

$$1 = 5 - 2 \cdot 2 = 5 - (7 - 5 \cdot 1)2 = 5 \cdot 3 + 7(-2) =$$
$$= (40 - 7 \cdot 5)3 + 7(-2) = 40 \cdot 3 + 7(-17).$$

Оскільки $-17 \equiv 23 \pmod{40}$, то $d = 23$.

Спосіб 2 Вираз $7d \equiv 1 \pmod{40}$ представимо у вигляді

$$7d + 40\,k = 1.$$

Для вирішення діофантового рівняння використовуємо алгоритм Евкліда:

$$40 = 7 \cdot 5 + 5;$$

$$7 = 5 \cdot 1 + 2;$$

$$5 = 2 \cdot 2 + 1;$$

$$2 = 1 \cdot 2 + 0.$$

Складаємо ряд l_i коефіцієнтів у зворотному порядку обчислень (пока зано стрілкою), останній рядок не враховується; отримуємо

$$l_1 = 2,\ l_2 = 1,\ l_3 = 5.$$

Складаємо новий ряд h_iкоефіцієнтів, перше значення якого завжди дорівнює одиниці: $h_i = 1$; друге значення ряду h_i дорівнює першому значенню ряду l_1; $h_2 = h_2 = l_1$. Інші значення ряду hi обчислюються за формулою

$$h_i = l_{i-1}h_{i-1} + h_{i-2} \text{ при } i \geq 3.$$

Останні два значення ряду h_i визначають значення коефіцієнтів k та d'. Отримуєм

$$h_2 = 1,\ h_2 = 2,\ h_3 = 3,\ h_4 = 17.$$

Отримуємо $d' = -17,\ k = 3$.

$$7(-17) + 3 \cdot 40 = 1.$$

Оскільки $-17 \pmod{40} \equiv 23$, то $d = 23$.

$$7 \cdot 23 \equiv 1 \pmod{40}$$

Для знаходження d скористаємося визначенням: якщо $x \equiv b \pmod{p}$, то $x \equiv (ba^{\varphi(p)-1}) 1 \bmod p$.

Підставимо значення:

$$d \equiv (7^{\varphi(40)-1}) \bmod 40 \equiv 7^{15} \pmod{40} \equiv (7^8 \cdot 7^4 \cdot 7^2 \cdot 7) \bmod 40 \equiv (1 \cdot 1 \cdot 9 \cdot 7)$$

$$\bmod 40 \equiv \equiv 23.$$

Шифруємо повідомлення M :

$$C \equiv 15^7 \pmod{55} \equiv (15^6 \cdot 15) \bmod 55.$$

Знаходимо:

$$15^2 \pmod{55} \equiv 225 \pmod{55} \equiv 5;$$

$$15^6 \pmod{55} \equiv (15^2 \bmod 55)^3 \bmod 55 \equiv 5^3 \pmod{55} \equiv 15.$$

Тоді:

$$C \equiv (15 \cdot 15) \bmod 55 \equiv 5.$$

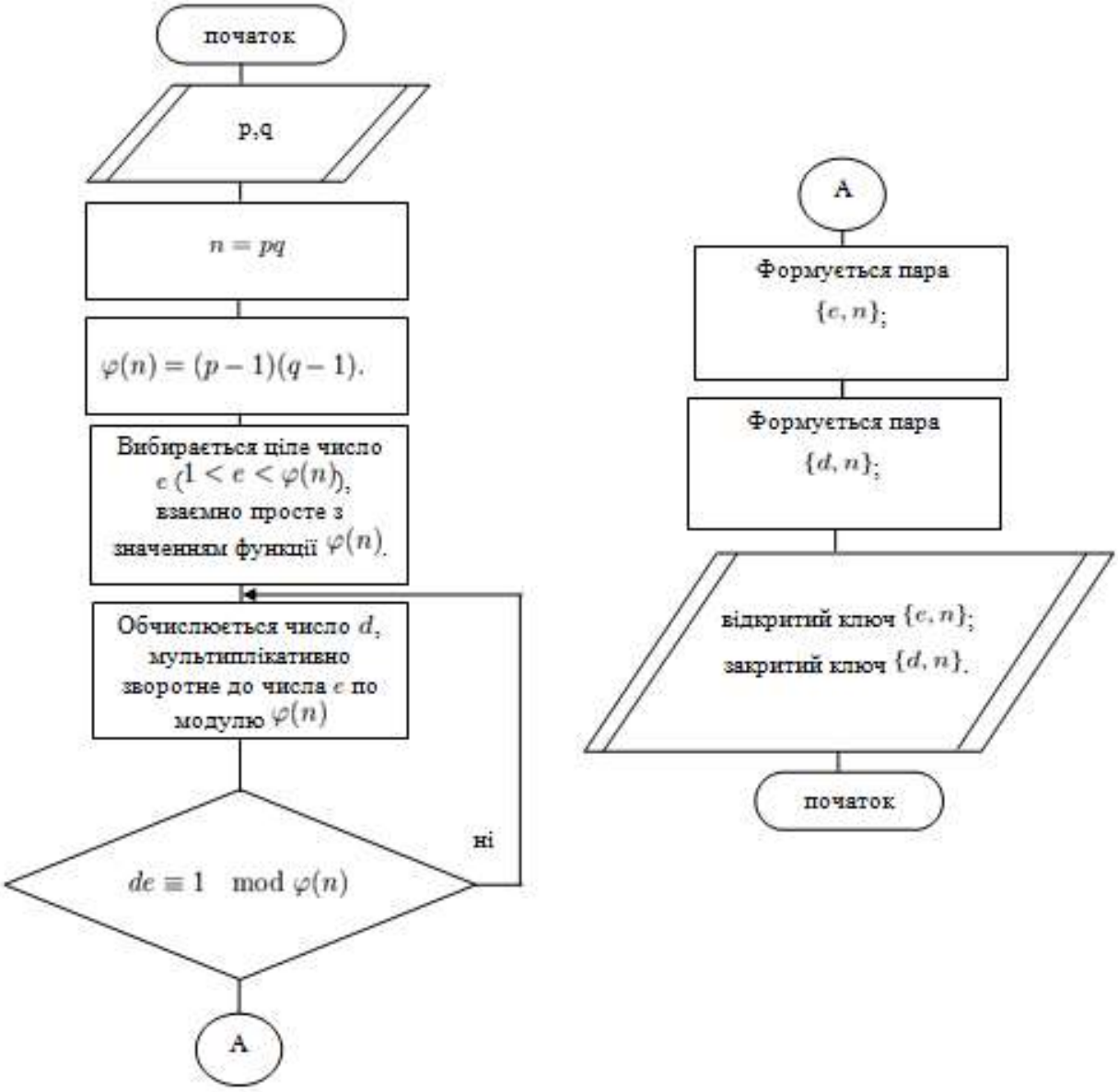

Рисунок 2.2 – Блок-схема алгоритму RSA

Розшифровуєм повідомлення С:

$M \equiv 5^{23} \pmod{55} \equiv (5^{16} \cdot 5^4 \cdot 5^2 \cdot 5) \bmod 55 \equiv (5 \cdot 20 \cdot 25 \cdot 5) \bmod 55 \equiv 15.$

Зашифруємо абревіатуру RSA. Для цього літери R, S і A за кодуємо п'ятивимірними двійковими векторами, скориставшись двійковим записом їх порядкових номерів у англійському алфавіті: R = 18 = 10010;S = 19 = 10011; A = 1 = 00001. Тепер представимо це повідомлення у вигляді за послідовністю чисел, що містяться в інтервалі 0,526. Отримуємо представлення.

RSA = (100101001), (100001) = (M_1 = 297, M_1 = 33).

Нехай p = 17 та q = 31, обчислюємо n = 17 · 31 = 527; φ (527) = (17 -1) (31 - 1) = 480. Вибираємо e = 7 і визначаємо d = 343. Зашифровуємо повідомлення M_1та M_2

$$C^1 \equiv 297^7 \pmod{527} \equiv 474;$$

$$C^2 \equiv 33^7 \pmod{527} \equiv 407.$$

В підсумку отримуємо шифротекст C = (474, 407).

Розшифровуємо повідомлення C.

$$M_1 \equiv 474^{343} \pmod{527} \equiv 297;$$

$$M_2 \equiv 407^{343} \pmod{527} \equiv 33.$$

Повертаючись до буквенного запису, отримуємо після розшифровування RSA.

Припустимо, що ми вирішили використати 8-бітовий код ASCII для символів з розміром блоку, що дорівнює символу або літері. Нам потрібно $n \geq 28$= 256. Нехай p = 41 і q = 73, тож n = 41 · 73 = 2993; φ (2993) = 40 · 72 = 2880. Нехай e = 217, так що НСД (217, 2880) = 1. Використовуючи алгоритм Евкліда для вирішення порівняння 217 $d \equiv 1 \pmod{2880}$, отримуємо d = 1513. У такому разі слово MONEY, зашифроване з використанням RSA-метод з ключем (n, e) = (2993, 217), як приведено в таблиці 2.2.

Таблиця 2.2 - Зашифроване слово

i	Блок	M_i	C_i
1	M	77	1537
2	O	79	79
3	N	78	1246
4	E	69	1529
5	Y	89	235

Кількість чисел від 1 до n, що не піддаються зашифруванню, визначається за формулою.

$$N = (1 + НСД (d – 1, p – 1))(1 + НСД (d – 1, q – 1)).$$

Як видно з прикладу, є число $M_2 = C_2 = 79$ не піддається шифрування за алгоритмом RSA, що є недоліком.

2.2 Аналіз криптостійкості RSA

Стійкість алгоритму RSA залежить від важкості рішення проблеми розкладання на множники великих чисел. З технічної точки зору це неправильно. Твердження, що безпека RSA залежить від проблеми розкладання на множники великих чисел, є гіпотетичним. Ні хто і ніколи не довів математично, що для відновлення M по C і e потрібно розкласти n на множники. Не виключено, що може бути відкритий зовсім інший спосіб криптоаналізу RSA. Але якщо цей новий спосіб дозволить криптоаналітику отримати d, він також може бути використаний для розкладання на множники великих чисел.

Розглянемо “лобовий метод” розкриття системи RSA, який полягає у знаходженні числа d, мультиплікативного зворотного e по модулю $\varphi(n)$. Це легко зробити, якщо відомі числа p та q. Отже, вирішивши завдання

розкладання на множники цілого числа *n* можна дешифрувати систему RSA. Для того щоб ускладнити задачу розкладання *n* на прості помножувачі, числа *p* і *q* повинні вибиратися випадково і мати досить великі значення. Крім того, числа *p* і *q* не повинні бути надто близькими один до одного. Але такий спосіб взламу менш ефективний, ніж навіть спроба розкласти добуток на множники.

Покажемо можливість використання близькості значень *p* та *q*. Без обмеження спільності вважатимуться, що $p > q$. Для величин $x = (p + q) / 2$ $y = (p - q)/ 2$ справедливе відношення $x^2 - y^2 = n$. Щоб знайти розкладання *n* на прості помножувачі, досить вибрати цілі числа x і y.

Перебираючи в порядку зростання варіанти $x > \sqrt{n}$ легко знайти рішення, так як $\mathbf{x} = (p + q)/2$ буде близьким до $\sqrt{n}$ за умови, що *p* і *q* близькі. В підсумку знаходимо: $p = x + y$, $q = x - y$.

Нехай $n = p\ q = 851$. Скористаємося описаним вище способом, щоб розкласти *n* на прості співмножники *p* і *q*. Так як $\sqrt{851} \approx 29{,}17$ вибираємо x = 30, обчислюємо 30^2 - 851 = 49 і знаходимо рішення x = 30, y = 7. Звідси p = 30 + 7 = 37, q = 30 - 7 = 23.

Атака під час використання загального модуля. Можлива реалізація RSA, у якій усім користувачам лунає однаковий модуль *n*, але кожному передається окреме значення показників ступеня *e* і *d*. На жаль, така реалізація не працюватиме. Найбільш очевидна проблема полягає в наступному. Нехай одне й те саме повідомлення колись зашифровувалося різними показниками (з одним і тим самим модулем) і ці два показники – взаємно прості числа (як це зазвичай і буває). Тоді відкритий текст може бути розкритий навіть за відсутності будь-яких відомостей про один із ключів розшифровування.

Нехай M – це відкритий текст повідомлення, e_1 та e_2 – два ключі зашифрування, *n* – загальний модуль. Шифртекстами повідомлення є:

$$C_1 \equiv M^{e_1} (\text{mod } n);$$

$$C_2 \equiv M^{e_2} (\text{mod } n).$$

Криптоаналітику відомі n, e_1, e_2, C_1 і C_2. Ось як він визначає повідомлення *M*. Оскільки e_1 і e_2 – взаємно прості числа, то за допомогою розширеного алгоритму Евкліда можна знайти *r* і *s*, для яких r e_1 + s e $_2$= 1.

Вважаючи s негативним (тут або *r* або *s* має бути негативним; припустимо, що негативним буде *s*), знову слід скористатися розширеним алгоритмом для обчислення C_2^{-1}. Тоді :

$$(C_2^{-1})^{-S} C_1^r \text{ M } (\text{ mod } n).$$

Існують два інші, більш тонкі способи розкриття систем такого типу. Один використовує ймовірнісний метод для розкладання п на множники. Інший є детермінованим алгоритмом обчислення секретного ключа без розкладання модуля на множники .

Припустимо, що шифротекст $C \equiv 1537$ зашифровувався з використанням RSA-методу з ключем (*n*, *e*) = (2993, 217) (див. приклад 2.7). Криптоаналітик вибирає число *x* = 207 та обчислює:

$$Y \equiv 207^{217} (\text{mod } 2993) \equiv 1594;$$

$$Z \equiv (1594 \cdot 1537) \text{ mod } 2993 \equiv 1704;$$

$$t \equiv 207^{-1} (\text{mod } 2993) \equiv 882$$

Потім криптоаналітик просить підписати повідомлення *Z* закритим ключем *d* і в результаті отримує *u*≡ 974.

Криптоаналітик обчислює *M*:

$$M \equiv (882 \cdot 974) \text{ mod } 2993 \equiv 77.$$

На підставі розглянутих атак можна зробити наступні обмеження для алгоритму RSA:

– Знання однієї пари секретного відкритого показників для даного модуля дозволяє криптоаналітику розкласти модуль на множники;

– Знання однієї пари секретного/відкритого показників для даного модуля дозволяє криптоаналітику обчислити інші пари показників, не розкладаючи модуль на множники;

– У протоколах мереж зв'язку, що застосовують RSA, не повинен використовуватись загальний модуль;

– Секретні показники мають бути більшими числами;

– Не достатньо використовувати стійкий криптографічний алгоритм: Мають бути безпечними вся криптосистема та криптографічний протокол.

Алгоритм RSA є стандартом де-факто, прийнятим майже у всьому світі. Організація ISO розробила стандарт цифрового підпису на основі RSA; RSA є інформаційним доповненням стандарту ISO 9796 . Багато компаній використовують алгоритм PKCS, створений компанією RSA Data Security, Inc. Алгоритм RSA запатентовано США. Термін дії патенту минув 20 вересня 2000 року.

2.3 Алгоритм Генерування відкритого і закритого ключів RSA

2.3.1 Генерація відкритого ключа

Під RSA, відкриті ключі складаються з простого числа e, так само як і N. Число e може бути будь що між 1 і значенням для $\lambda(N)$, що в нашому прикладі становить 349,716.

Оскільки відкритий ключ передається відкрито, це не так важливо для e бути випадковим числом. На практиці, e зазвичай встановлюється на 65537, тому що коли випадковим чином вибираються набагато більші числа, шифрування стає не ефективним. Для прикладу ми створимо невеликі цифри, щоб зробити обчислення ефективними.

Виберемо e значення:

$$e = 11$$

Наші останні зашифровані дані називаються зашифрованим текстом (c). Ми виводимо це з нашого відкритого тексту повідомлення (m), застосовуючи відкритий ключ за такою формулою:

$$\mathrm{c} = me \cdot mod(n)$$

Для e є вже значення і N присутнє також. Єдине, що треба пояснити, це модифікація. Це по суті означає залишок, що залишився, коли ділитиме одну сторону на іншу. Наприклад:

$$10\ (mod\ 3) = 1$$

Це тому, що 3 входить у 10 три рази із остачею 1.

Повернемося до нашого рівняння. Для простоти, скажемо, що повідомлення (m), яке ми хочемо зашифрувати і зберегти в таємниці це просто одне число, 4. Давайте виберемо все:

$$c = me\ (mod\ N)$$

$$c = 411(mod\ 701111)$$

$$c = 4\ 194\ 304\ (mod\ 701111)$$

Знову ж таки, щоб зробити операції по модулю легко, ми будемо використовувати онлайн калькулятор. Обчислимо залишок числа 4 194 304 в онлайн-калькулятор, він дає нам(рисунок 2.4):

$$c = 68874$$

Коли використовується RSA для шифрування нашого повідомлення,, з нашим відкритим ключем, це дає нам зашифрований текст 688,749. Попередні кроки могли здатися надто складними.

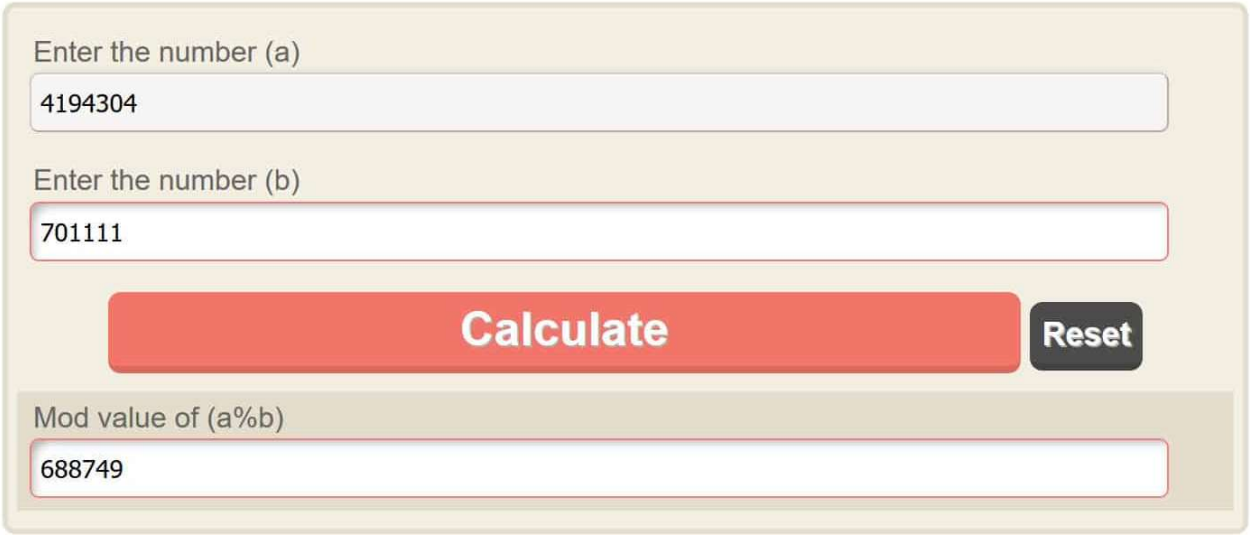

Рисунок 2.4 - Знаходження залишку

Ми мали повідомлення, яке потрібно було зашифрувати. Ми застосували до нього відкритий ключ, який дав нам зашифрований результат 688749. Тепер, коли він зашифрований, можна безпечно відправити номер 688749 власнику пари ключів. Це є одна особа , яка зможе розшифрувати її за допомогою свого закритого ключа. Коли розшифрують його, то побачать повідомлення, яке ми справді надсилали.

2.3.2 Генерація закритого ключа

У RSA-шифруванні після перетворення даних або повідомлень у зашифрований текст з відкритим ключем їх можна розшифрувати тільки за допомогою закритого ключа з тієї ж пари ключів. Закриті ключі складаються з d і N. Ми вже знаємо N, і наступне рівняння використовується, щоб знайти d:

$$d = \frac{1}{e} \, mod \, \lambda(N)$$

Генерація відкритого ключа розділ вище, ми вже вирішили, що в нашому прикладі, e буде дорівнює 11. Так само ми знаємо, що $\lambda(N)$ дорівнює 349716 . Все стає трохи складніше, коли ми стикаємося із цим розділом формули:

$$\frac{1}{e} mod$$

Це рівняння може бути так, ніби воно просить вас розділити 1 на 11, але це не так. Натомість це просто символізує, що нам потрібно розрахувати модуль зворотнього e (який у даному випадку дорівнює 11) та $\lambda(N)$ (що в даному випадку становить 349716).

Зазвичай це зустрічається в розширеному евклідовому алгоритмі, але це трохи виходить за рамки цього, тому ми просто обійдемо і використаємо для цього онлайн-калькулятор. Тепер, коли розуміємо все, що відбувається, включимо інформацію до формули:

$$d = \frac{1}{11} (mod\ 349716)$$

Щоб виконати цю операцію, просто введіть 11 (або будь-яке значення, яке можна мати для e якщо пробуєте самостійно), це є ціле число і 349716 (або будь-яке значення, яке можна отримати для $\lambda(N)$ якщо пробуєте самостійно), де йдеться Модуль в онлайн калькуляторі, який був пов'язаний вище (рисунок 2.3.1).

Якщо все зроблено правильно, повинно отримати результат, де:

$$d = 254\,339$$

Тепер, коли ми маємо значення для d, ми можемо розшифрувати повідомлення, які були зашифровані за допомогою нашого відкритого ключа, використовуючи таку формулу:

$$m = cd\ (mod\ n)$$

Тепер ми можемо повернутися до зашифрованого тексту, який ми зашифрували під Генерацію закритого ключа. Коли було зашифрувне повідомлення відкритим ключем, це дало нам значення 688749. Вище відомо, що d дорівнює 254339.

Також відомо, що N дорівнює 701111.На виході отримуємо:

$$m = 688\ 749\ 254\ 339\ (mod\ 701111)$$

У калькуляторі, вказаному вище, введіть 701111, де написано Модуль живлення: N, 254,399, де йдеться Ключ розшифровки: D, і 688749, де йдеться про Зашифроване повідомлення в числовій формі, як показано нижче (рисунку 2.5):

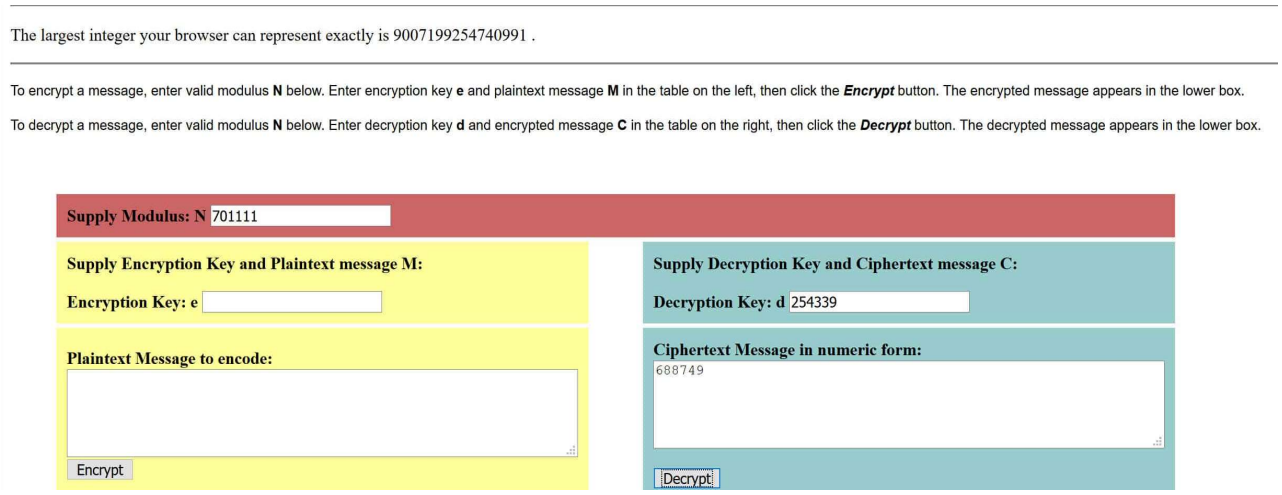

Рисунок 2.5 - Розшифрування

Після введення даних натисніть Розшифрувати, який помістить числа через формулу розшифрування, яка була вказана вище. Це дасть оригінальне повідомлення у полі нижче. Якщо ви все зробили правильно, ви повинні отримати відповідь 4, яка була вихідним повідомленням, яке ми зашифрували з відкритим ключем.

3 РОЗРОБКА ТА ТЕСТУВАННЯ АЛГОРИТМУ ШИФРУВАННЯ ФАЙЛІВ RSA

3.1 проєктування інтерфесу програмного засобу

При проєктуванні програмного інтерфейсу були поставлені вимоги створити зручний для користувача інтуїтивно зрозумілий інтерфейс, що надає доступ до розроблених функцій. На цьому етапі розробки програмного продуку уся технічна інформація разом з ключами відображається у текстовому вигляді, для того що зручно було перевіряти правильність виконання алгоритму та пройти етап верифікації програмного засобу. На рисунку 3.1 приведено екранну форму головного вікна.

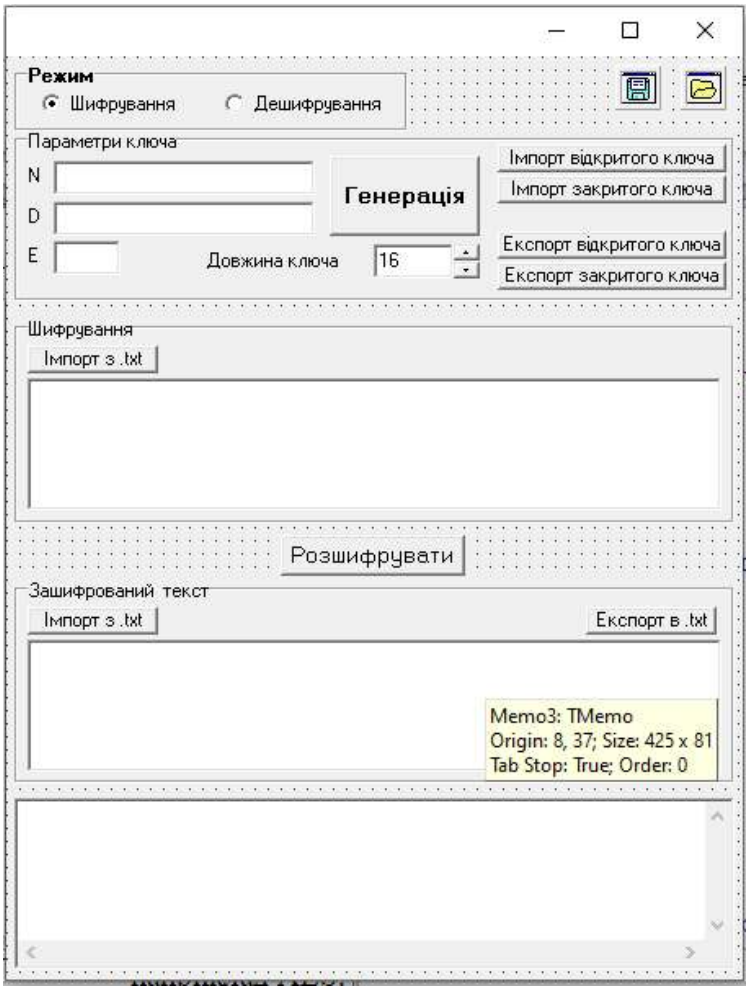

Рисунок 3.1 – Головне вікно програмного продукту на етапі розробки

Для створення програмного засобу окрім мови C++ було обрано середовище RAD Studio від Embarcadero. Середовище є досить зручним швидким засобом розробки як невеликих проєктів так і корпоративних рішень.

Оскільки шифрування файлів асиметричними алгоритмами потребує високої швидкодії коду було обрано саме середовище, що дозволяє компілювати код під певну платформу.

3.2 проєктування основних функцій

Алгоритм асиметричного шифрування RSA заснований на практичній складності факторизації великих чисел, що робить його на сьогоднішній день одним із найпопулярніших криптографічних алгоритмів.

Однак він має свої негативні сторони, наприклад серед них досить низька швидкість шифрування, тому часто використовують змішану криптосистему, в якій дві сторони передають симетричний ключ за допомогою RSA, а потім шифрують повідомлення за допомогою якого-небудь симетричного алгоритму, наприклад AES.

RSA може шифрувати числа так званого модуля, який є частиною ключа. В даний час використовуються модулі довжиною 1024, 2048 і навіть 4096 біт. Для того, щоб повідомлення при шифруванні багаторазово не збільшувалося в розмірі, шифрувати його треба блоками по декілька біт. При цьому кожен блок перейде в K -бітне число, тобто на великих файлах приріст розміру складе всього лише раз, і чим більше ключ - тим менший цей приріст.

У програмі розподілом масиву чисел однієї бітності на числа іншої бітності займається функція resize, доповнюючи відсутні біти нулями.

Шифруванням і одночасно дешифруванням займається функція process_bytes, так як у RSA обидва ці процеси мають ідентичні алгоритми,

що відрізняються тільки розміром блоку входу та виходу. Для цього використовується алгоритм швидкого піднесення у степінь.

Також програма може генерувати ключі на підставі встановлених простих чисел (в майбутньому випадкових), або на підставі простих чисел, введених користувачем. Для цього використовується знаходження зворотнього елемента в кільці за модулем використовуючи розширений алгоритм Евкліда.

На рисунку 3.2 приведено фрагмент коду на етапі розробки, що описує структуру ключів для RSA.

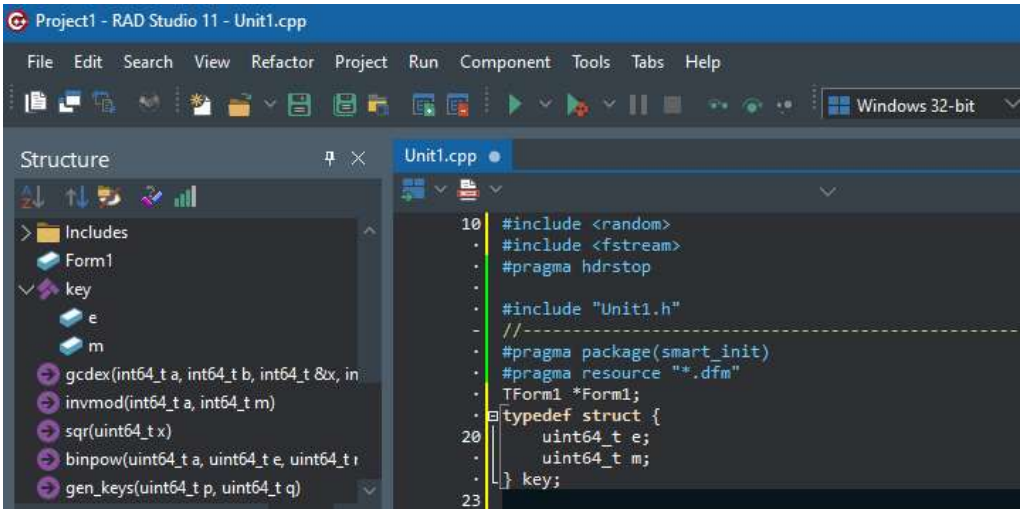

Рисунок 3.2 – Структура ключів RSA

Для обчислення найбільшого спільного дільника будемо використовувати розширений алгоритм Евкліда. Його код відображений на рисунку 3.3.

```
int64_t gcdex(int64_t a, int64_t b, int64_t &x, int64_t &y) {
    if (a == 0) {
        x = 0;
        y = 1;
        return b;
    }
    int64_t x1, y1;
    int64_t d = gcdex(b % a, a, x1, y1);
    x = y1 - (b / a) * x1;
    y = x1;
    return d;
}
```

Рисунок 3.3 – Розширений алгоритм евкліда

В алгоритмі найбільш трудомісткою операцією можна назвати операцію пошуку оберненого елемента за модулем, оскільки її обчислення базується на повільному розширеному алгоритмі Евкліда. Функція для пошуку оберненого елемента за модулем приведена на рисунку 3.4.

```
int64_t invmod(int64_t a, int64_t m) {
    int64_t x, y;
    gcdex(a, m, x, y);
    x = (x % m + m) % m;
    return x;
}
```

Рисунок 3.4 – Обернений елемент за модулем

Також в криптографічних перетвореннях необхідною операцією є операція приведенння до степеня. Оскільки операнди в сучасних криптосистемах можуть мати більше 1024 біт то обчислювальна складність, яка експоненційно зростає, має досить велике значення та впливає на роботу всього алгоритму. Для цієї операції створено функцію для швикого піднесення до степеня, як показано на рисунку 3.5.

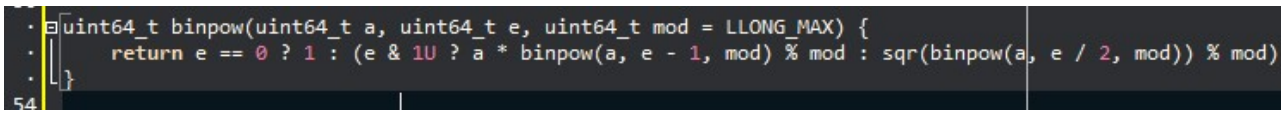

Рисунок 3.5 – Швидке піднесення до степеня

Чи не основною операцією для асиметричного шифрування є генерація закритого та відкритого ключів. Для цієї мети створено функцію що на рисунку 3.6.

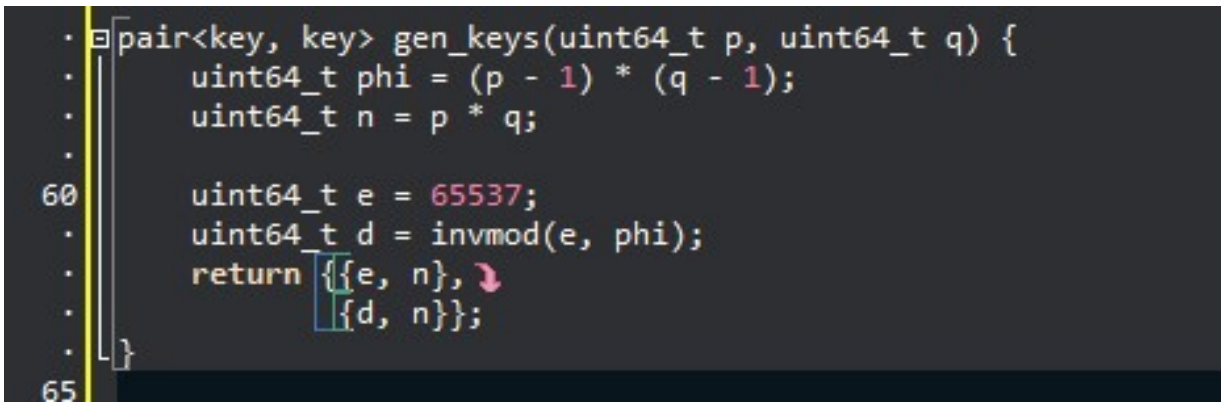

Рисунок 3.6 – Генерація пари ключів

Для функції розбиття на блоки створено технічну функцію визначення розміру блоку у байтах. Функція приведена на рисунку 3.7.

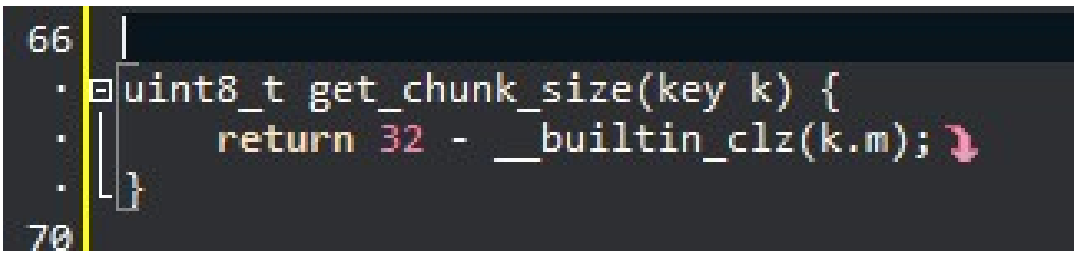

Рисунок 3.7 – Розмір блоку шифрування

Бітову послідовність потрібно поділити на блоки однакового розміру, а ті що заповненні не до кінця заповнити нулями. Для цієї задачі створено функції, що на рисунку 3.8.

```
vector<uint64_t> resize(const vector<uint64_t> &data, uint8_t in_size, uint8_t out_size) {
    vector<uint64_t> res;
    uint8_t done = 0;
    uint64_t cur = 0;
    for (uint64_t byte: data)
        for (uint8_t i = 0; i < in_size; i++) {
            cur = (cur << 1U) + (((uint64_t) byte & (1U << (uint64_t) (in_size - 1 - i))) != 0);
            done++;
            if (done == out_size) {
                done = 0;
                res.push_back(cur);
                cur = 0;
            }
        }

    if (done != 0)
        res.push_back(cur << (uint64_t) (out_size - done));
    return res;
}
```

Рисунок 3.8 – Розбиття даних на блоки

Основною функцією, що реалізує криптографічні перетворення є функція, що забезпечує процес шифрування та дешифрування. Її виконано універсально для обох процесів, котрі можна перемкнути аргументом функції.

Процес шифрування та опційно дешифрування можна реалізувати функцією, що на рисунку 3.9.

```
vector<uint8_t> process_bytes(const vector<uint8_t> &data, key k, bool encrypt) {
    vector<uint64_t> data_64(data.size());
    for (int i = 0; i < data.size(); i++)
        data_64[i] = (uint64_t) data[i];
    vector<uint64_t> resized_data = resize(data_64, 8, get_chunk_size(k) - encrypt);
    for (int i = 0; i < resized_data.size(); i++)
        encrypted_data[i] = binpow(resized_data[i], k.e, k.m);
    vector<uint64_t> result_64 = resize(encrypted_data, get_chunk_size(k) - !encrypt, 8);
    vector<uint8_t> result(result_64.size());
    for (int i = 0; i < result_64.size(); i++)
        result[i] = (uint8_t) result_64[i];
    return result;
}
```

Рисунок 3.9 – Шифрування та дешифрування

Важливим елементом програмного засобу є функції для роботи з файлами. З цією метою створено функції для читання та запису в файл, що приведені на рисунку 3.10.

```
vector<uint8_t> read_bytes(const char *filename) {
    ifstream fin(filename);
    fin.seekg(0, ios::end);
    size_t len = fin.tellg();
    auto *bytes = new char[len];
    fin.seekg(0, ios::beg);
    fin.read(bytes, len);
    fin.close();
    vector<uint8_t> ret(len);
    for (int i = 0; i < len; i++)
        ret[i] = (uint8_t) bytes[i];
    delete[] bytes;
    return ret;
}

void write_bytes(const char *filename, const vector<uint8_t> &data) {
    ofstream fout(filename);
    char *buf = new char[data.size()];
    for (int i = 0; i < data.size(); i++)
        buf[i] = (char) data[i];
    fout.write(buf, data.size());
    fout.close();
}
```

Рисунок 3.10 – Функція для роботи з файлами

Після реалізації основних функцій булос створено ієрархію класів для реалізації в проєкті, що відповідає SOLID принципам програмування та забезпечує основний набір функцій та обєктів. Ієрархія класів приведена на рисунку 3.11.

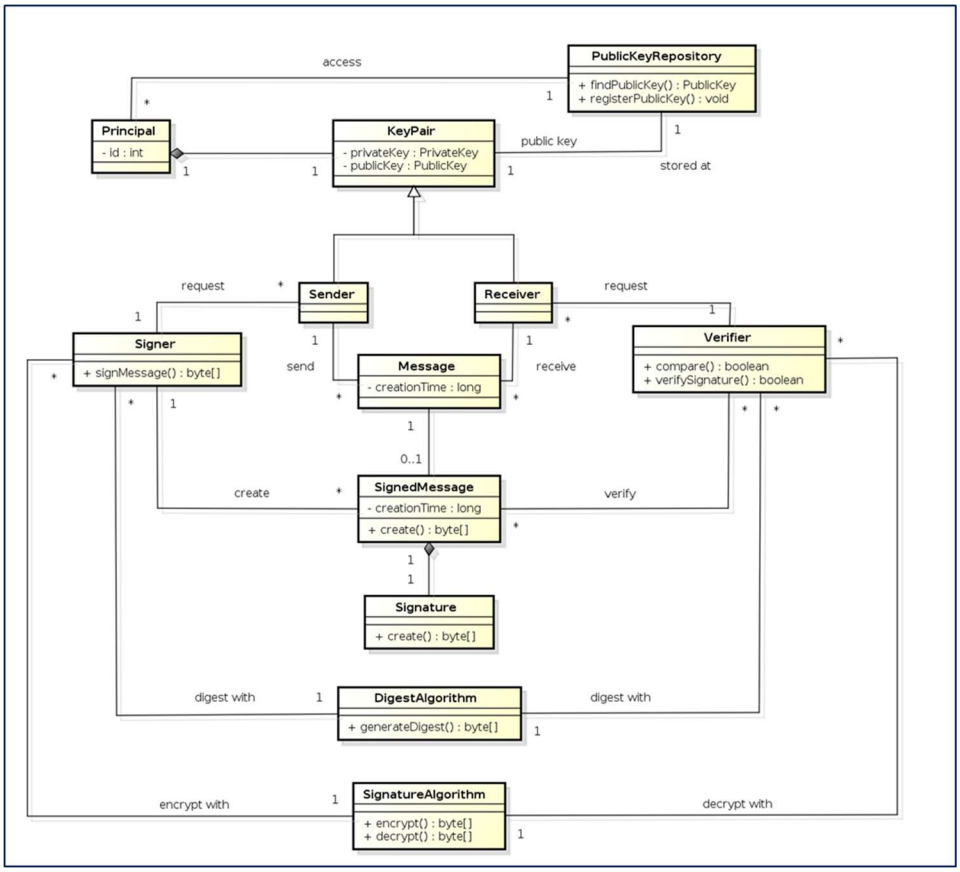

Рисунок 3.11 – Діаграма класів розробленого програмного засобу

Розроблені функції та класи реалізовано в вишляді програмних модулів в середовищі RAD Studio та забезпечують необхідний набір функцій для асиметричного шифрування.

3.3 Тестування програми

Тестування будь-якого продукту є дуже важливим в кінцевій його здачі, цей процес потрібен для виявлення всіх помилок і недоліків будь чого. Без тестування продукт або в цьому випадку програма не зможе вийти до кінцевого користувача і працює вірно. В цьому процесі дуже багато методів і оцінки програмного забезпечення.

Під час тестування будь якого програмного засобу відбувається процес, дослідження того, наскільки дійсна реакція системи , та чи відповідає вона вимогам. Під час тесту використовуються такі перевірки що було виявлено будь який дефект програми. Потрібно зробити так щоб продукт який створюється працював належними чином, не залежно від обставин і відповідати поставленим вимогам.

Життєвий цикл ПЗ – це період від часу появи створення програми до етапу завершення його підтримки розробником який супроводжував продукт.

Основними етапами програмного тестування:

- аналіз вимог;
- процес створення дизайну;
- розробка;
- процес тестування та дебаггінгу;
- експлуатація та підтримка.

Тестування це необхідний процес, тестування розпочинається ще до програми. Тестувальники отримують настанови, макети, та будь які вимоги які є не є видимими. Проблеми повинні бути вирішенні ще до старту розробки програми.

Під час цього етапу враховуються такі фактори як вимоги, етапи тестування, не передбачувані обставини під час тестування програми.

Під час цього тестування програми для шифрування фалів було визначено:

– Різновид програми яка визначається її функціоналом, в програмі яка спрямована на шифрування та дешифрування текстових файлів.

– Особи для який призначена ця програма (Звичайні персони, ФОП, фірми які працюють з даними, охоронні служби, військові.).

– Поширення ПЗ(Мережа інтернет).

Функціональне тестування, направленні на те щоб переконатись що тестування працює правильно і співпадає вимогам. Під час цього етапу перевіряється чи виконуються функції (рисунок 3.12).

Рисунок 3.12 – Етапи тестування

При тестуванні програмного забезпечення було провірено весь функціонал програми. Програма підтримується операційною системою Windows 10 а також старіша версії такі як Windows 7 і Windows 8 і 8.1.

Тестування використання ресурсу програми встановленого ПЗ відзначається місцем його встановлення. Даний додаток не мішає роботі інших програних засобів та не сильно споживає оперативну пам’ять, розмір який займає ця програма становить 701 Кб.

Перевірка на сумісність програми потрібне для забезпечення продуктивності застосування на різних пристроях. Також було проведено юзабіліті-тестування яке акцентувало увагу на створення умов користування програми, та створення інтерфейсу який підходить до стандартів. Головна критерія оцінювання програмного забезпечення є оцінка безпосередньо від користувача.

Під час тестування визначено, що кнопки в програмі мають достатній розмір і розташовані у зручній області програми. Текст в програмі читабельний і зручний у використанні. Підпис кожної кнопки простий і зрозумілий, і по ньому видно що виконує кнопка. Користувач може вибрати чи шифрувати чи дешифрувати текст, можна вибрати довжину ключа від 8 до 512 символів, можна імпортувати текст та зашифрувати його, можна вводити ключі, імпортувати та експортувати закриті і відкриті ключі.

Під час запуску програми показується головний інтерфейс що підходить для наших задач. Ось як виглядає інтерфейс програми (рисунок 3.13)

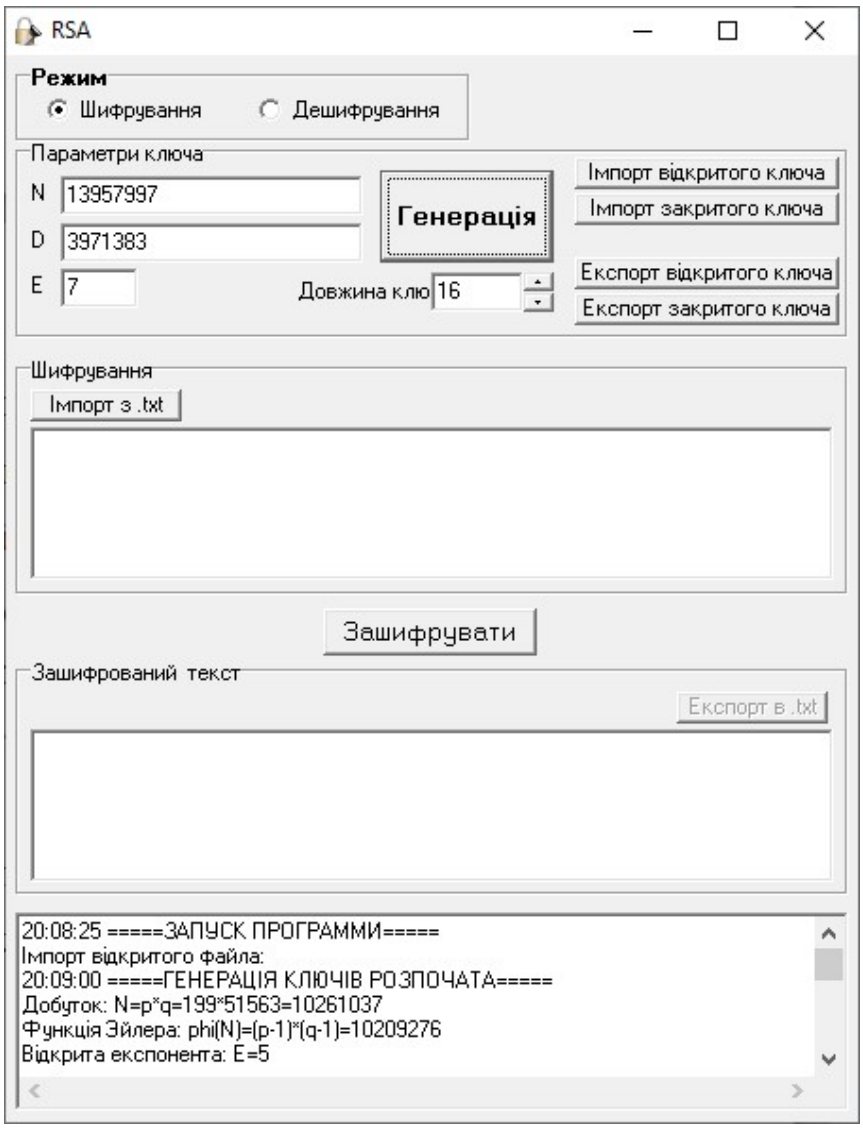

Рисунок 3.13 – Інтерфейс програми

Розпочнемо з того що ми згенеруємо ключ який ми зможемо потім використовувати. Для того щоб створити ключ , потрібно вибрати його довжину в спеціальному вікні , максимально допустима довжина ключа становить 512 символів, більше не можливо.

Виберемо довжину ключа 32 символів і натиснемо на кнопку «Генерація». Після того як було нажато кнопку, в нижній частині програти в спеціальному вікні написано повідомлення що інформує нас про початок генерації ключа (рисунок 3.14).

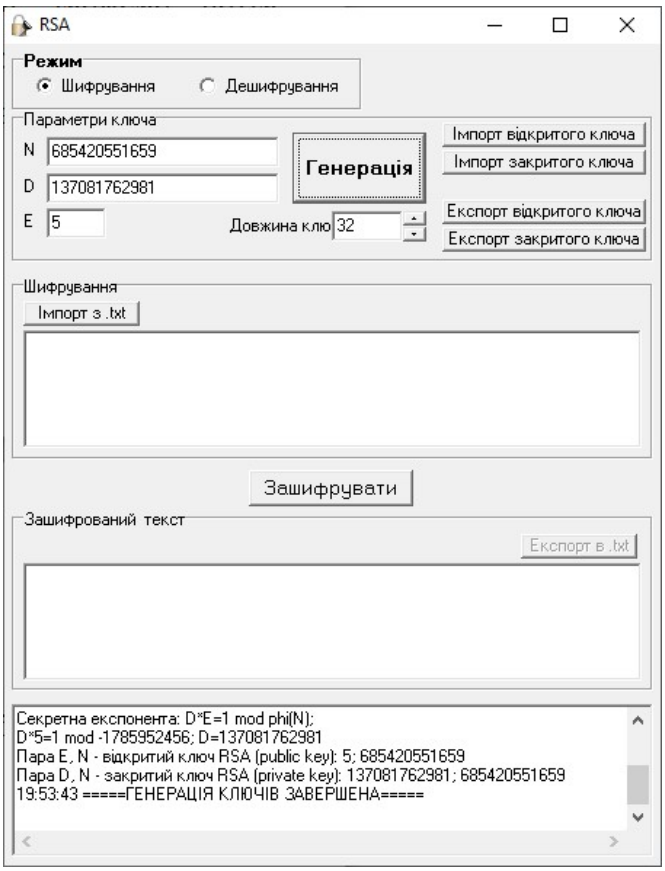

Рисунок 3.14 – Генерація ключа

У зв’язку з тим, що у нас довжина згенерованого паролю не є великою, то програма працює швидко. Після того як програма створила пароль за нашим параметром, а саме довжина 32 символи, було про це з інформовано в нижній частині екрана у спеціально відведеному вікні. У цьому вікні нам доступно велика кількість інформації про ключ (рисунок 3.15).

Рисунок 3.15 – Інформаційна частина програми

В даному вікні ми бачимо інформацію: Секретна експонента, формула яка створює значення D для закритого ключа за формулою, пару E, N

відкритий ключ, та пару ключів D, N, що є даними закритого ключа, та інформацію про те що програма закінчила генерацію ключа (рисунок 3.16).

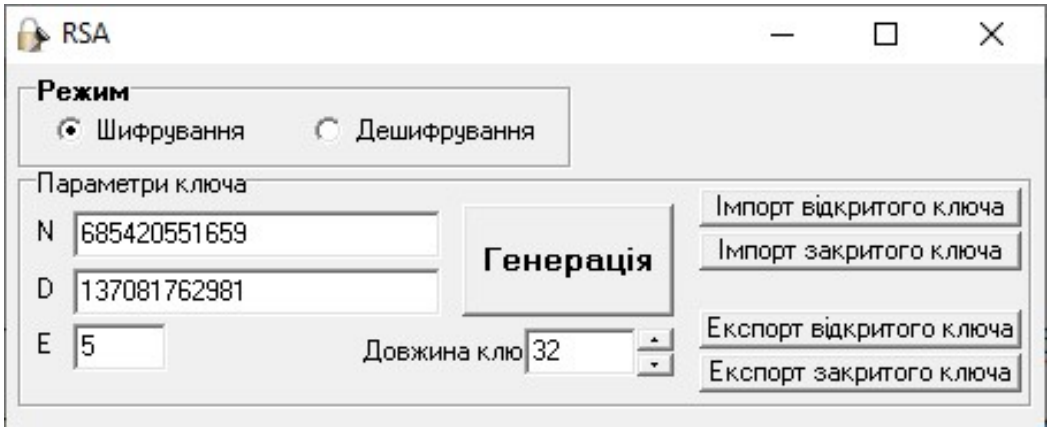

Рисунок 3.16 – Верхня частина програми

У верхній частнині програми чи бачимо поля, режими а саме «Шифрування», «Дешифрування», та параметри ключа, кнопка якою ми виконуємо дію генерації і під нею довжина ключа, яку можна змінювати. Також з лівої сторони по парно розділено кнопки за допомогою яких можна отримати та закинути в програму, що свідчить їхні назви. Для того щоб отримати один з ключів потрібно натиснути на кнопку «Експорт відкритого ключа» або «Експорт закритою ключа», в другому випадку буде файл в якому буде зашифрований ключ (рисунок 3.17).

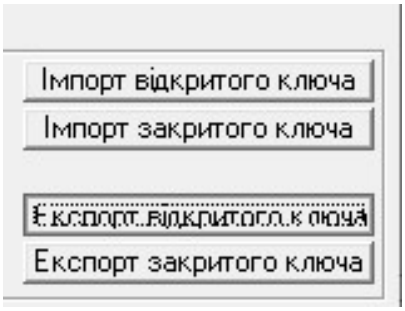

Рисунок 3.17 – Експорт закритого ключа

Оберемо простіший варіант і натиснемо кнопку «Експорт відкритого ключа», після натискання нам вікривається меню в яку ми можемо обрати куди зберегти ключ, та дати йому назву.

Збережемо його на робочий стіл і дамо йому назву «Відкритий ключ» (рисунок 3.18).

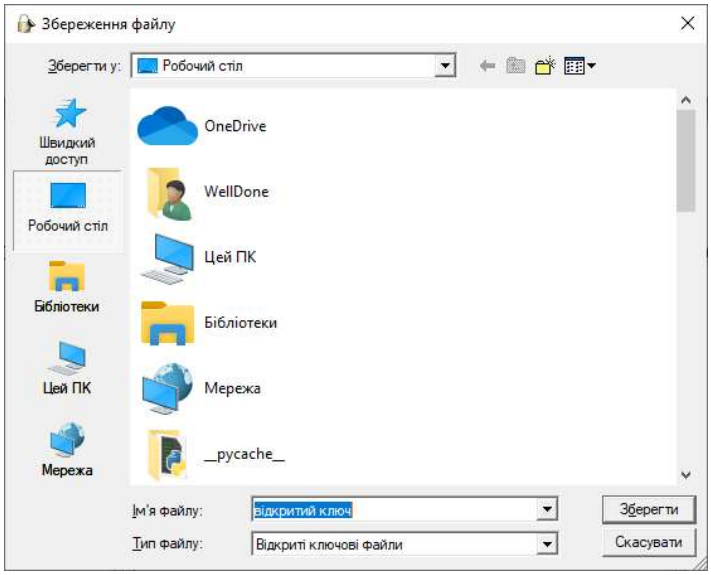

Рисунок 3.18 – Збереження файлу

Після натискання кнопки «Зберегти» ключ зберігається на робочому столі з назвою «відкритий ключ » та з файловий розширенням «pubkey», що означає publickey- відкритий ключ(рисунок 3.19).

Рисунок 3.19 – Іконка збереженого файлу

Цей файл на самому початку має іконку пустого аркуша, на даному рисунку вибрано програму для читання «SublimeText», в цій програмі зручно переглядати ключі.

Якщо відкрити цей файл, то побачимо що в ньому знаходяться значення які є під буквами N, E і його довжину.(рисунок3.20)

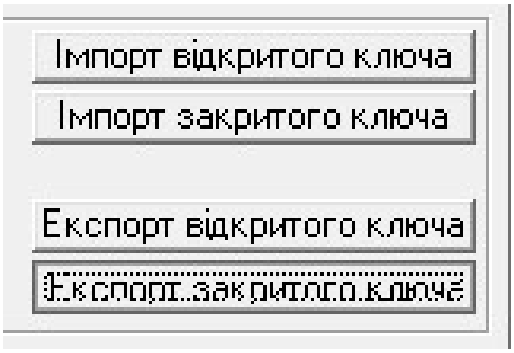

Рисунок 3.20 – Інформація про ключ

Далі ми виберемо другу кнопку і натиснемо на «Експорт закритого ключа»(рисунок 3.21)

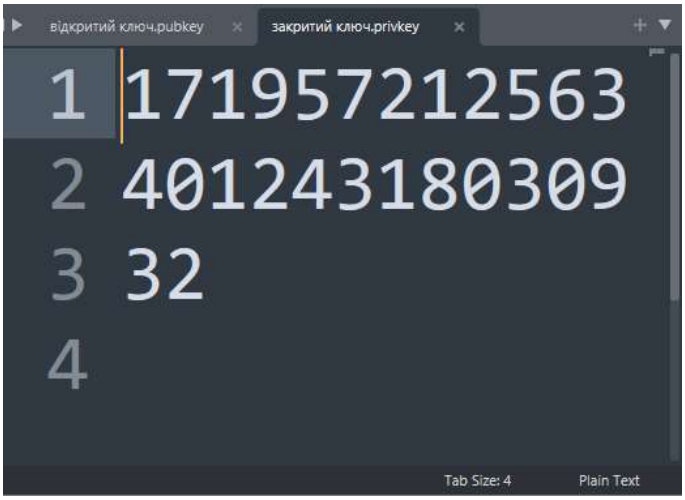

Рисунок 3.21 – Експорт закритого ключа

В цьому випадку коли ми створюємо закритий ключ там не показує значення E, але замість нього ми отримуємо значення N.(рисунок3.22)

Ці файли можна імпортувати в програму для того щоб розшифровувати і зашифровувати файли. Це потрібно для того щоб хтось інший з використанням цієї програми міг розшифрувати.

А зараз приступимо до шифрування та дешифрування текстового файлу і просто текстового повідомлення яке можна ввести в самій програмі.

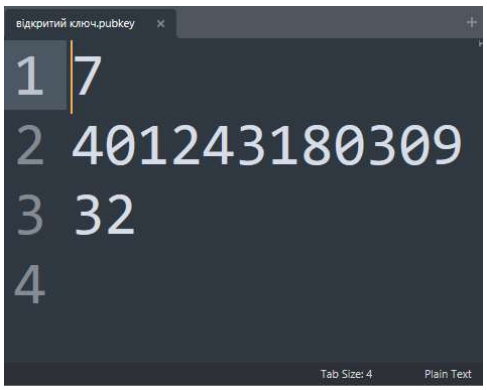

Рисунок 3.22 – Інформація про закритий ключ

Розпочнемо з того що ми створимо текстовий файл в якому буде написана якась інформація.

Створюємо текстовий документ в якому в нас буде інформація студентів і їх теми дипломів (рисунок 3.23).

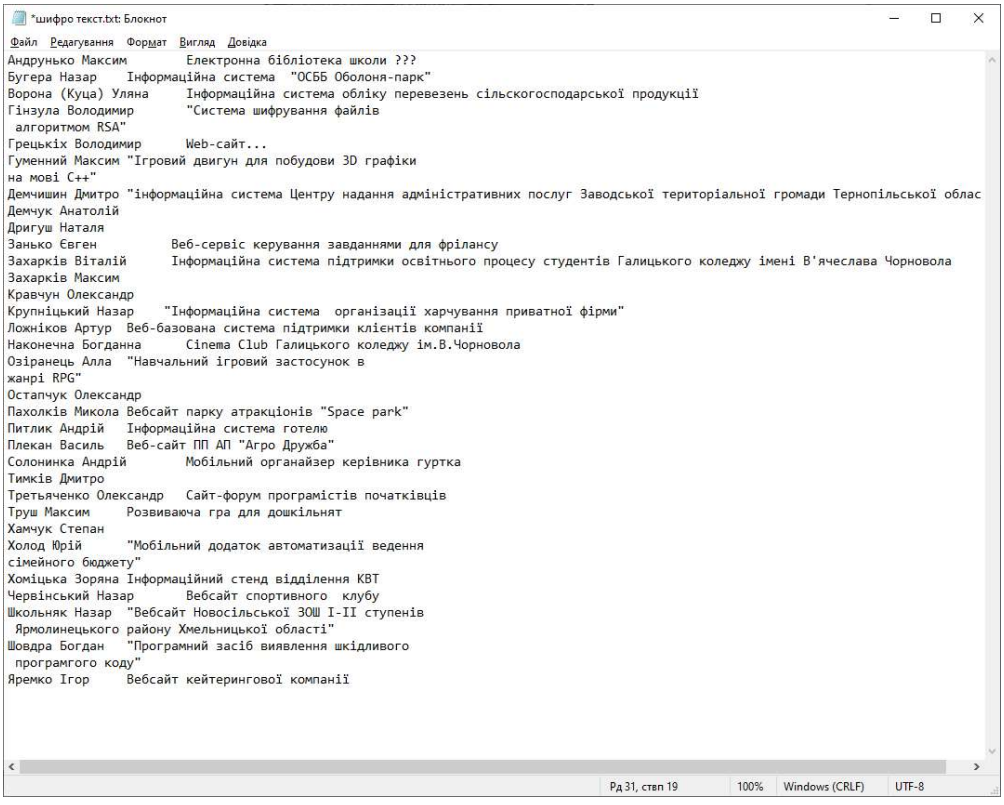

Рисунок 3.23 – Список тем дипломів

Перенесемо інформацію з текстового документу в прогамний засіб для шифрування за допомогою кноки імпорту. Процес імпорту приведений на рисунку 3.24.

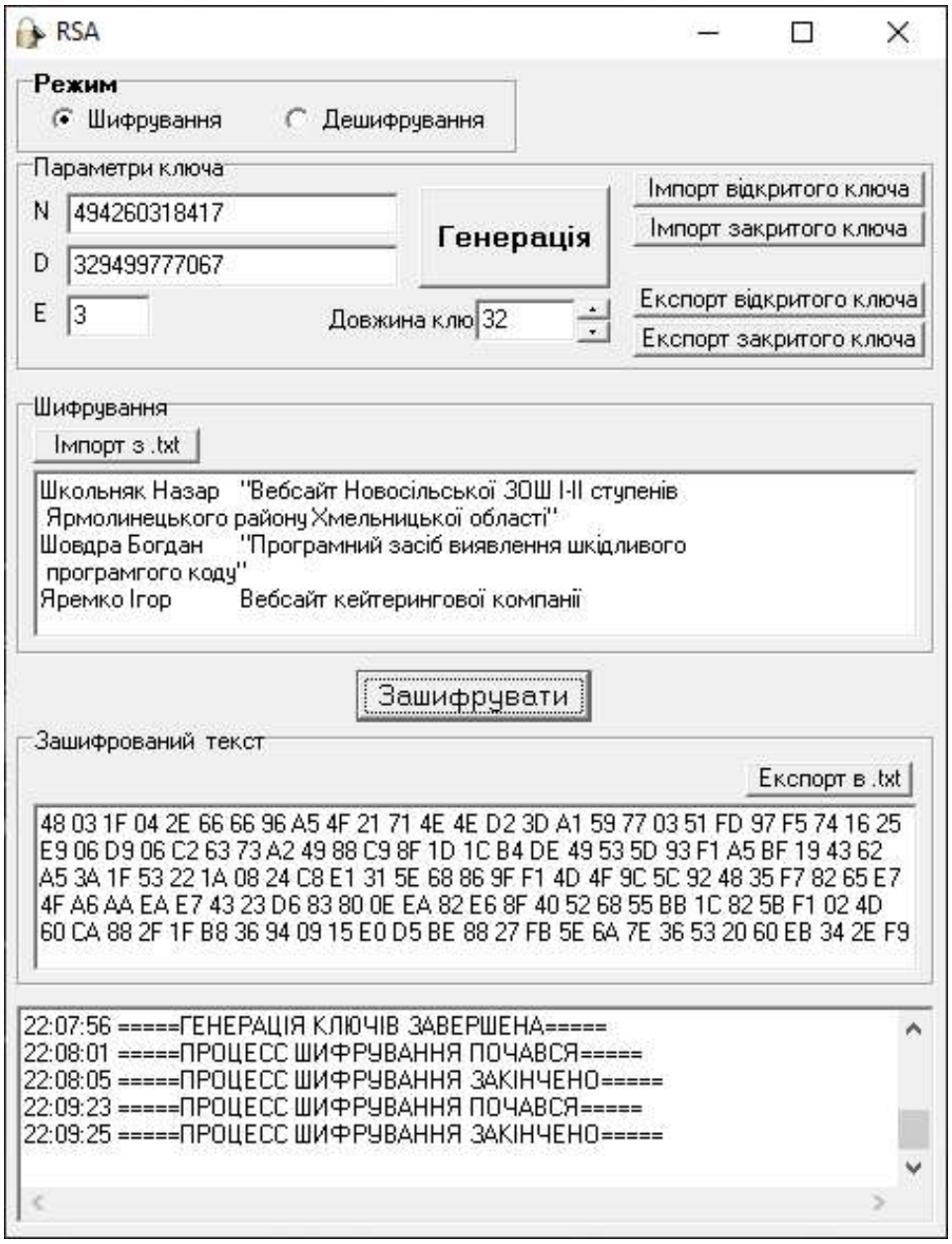

Рисунок 3.24 – Шифрування імпортованого файлу

Дальші цей файл ми імпортуємо в програму за домогою копки «Імпорт txt». Після цього весь вміст цього файлу нам показується в цьому вікні (рисунок 3.24).

Видно що даний текст було зашифровано і ми бачимо як він

виглядає після шифрування. Тепер ми можемо цей текст експортувати в txt і отримати файл в якому буде це зашифроване повідомлення, як приведено на рисунку 3.25.

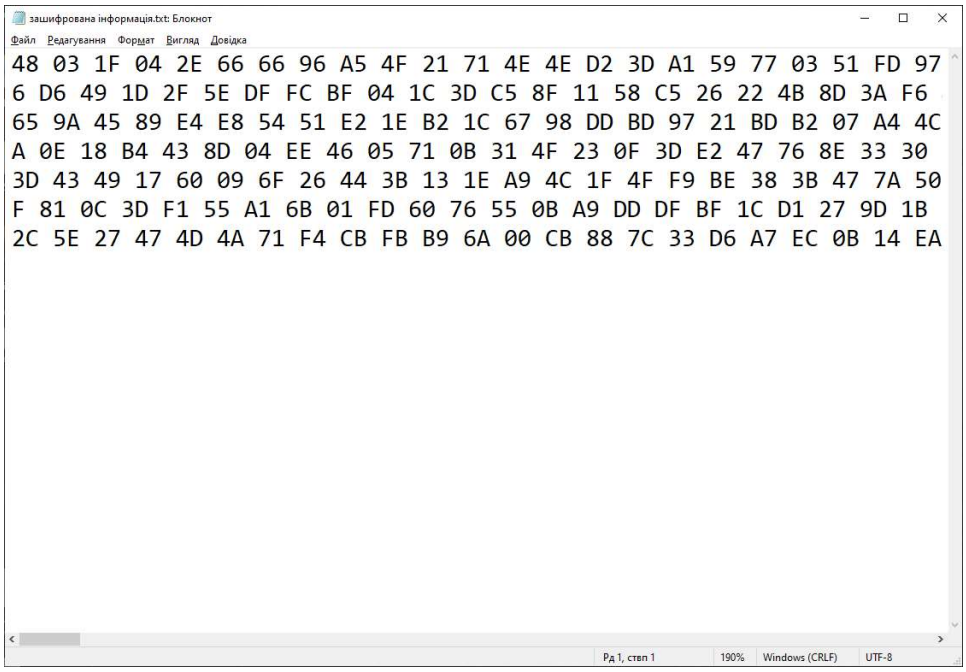

Рисунок 3.25 – Текст для шифрування

Після того як було зашифроване повідомлення і його зберегли на своєму комп’ютері можна приступити до того що будемо розшифровувати його але закриємо програму ніби то розшифровує його інший користувач, той кому було надіслано це зашифроване повідомлення. І перевіримо чи програма зашифрувала(рисунок 3.26).

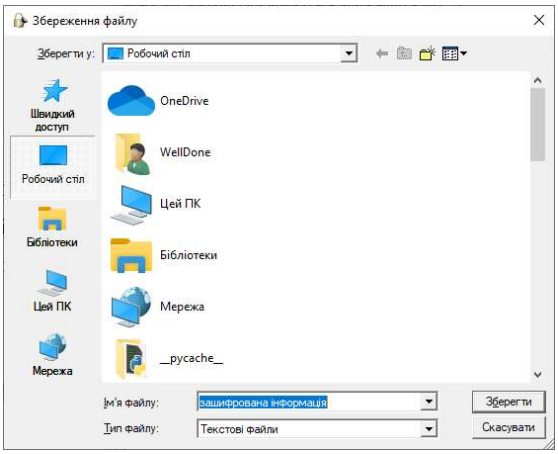

Рисунок 3.26 – Збереження зашифрованого повідомлення

Дальше нам потрібно витягнути ключі для того щоб надіслати їх тому хто буде розшифровувати повідомлення, для цього натискаємо експорт обох видів ключів і використаємо їх для того щоб розшифрувати повідомлення, але спочатку нам потрібно закрити програму , щоб імпортувати ключі та перевірити працює справно (рисунок 3.27).

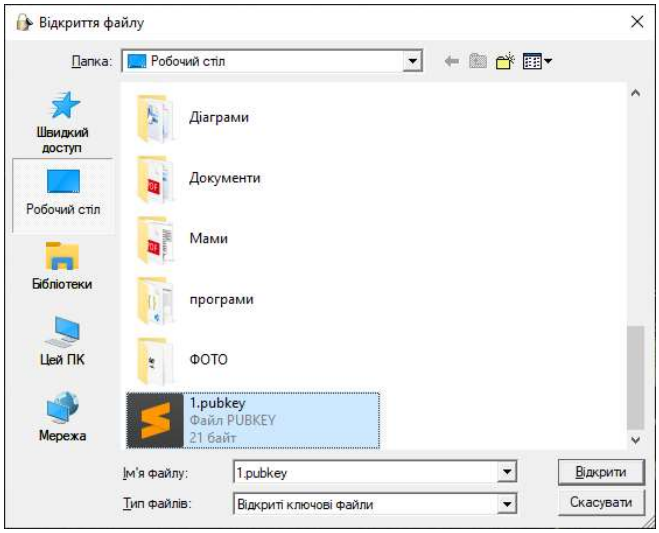

Рисунок 3.27 – Імпорт відкритого ключа

Імпортуємо публічний ключ(рисунок 3.27) щоб можна було розшифрувати повідомлення та перевіряємо чи за заповнились колонки відповідними даними (рисунок 3.28).

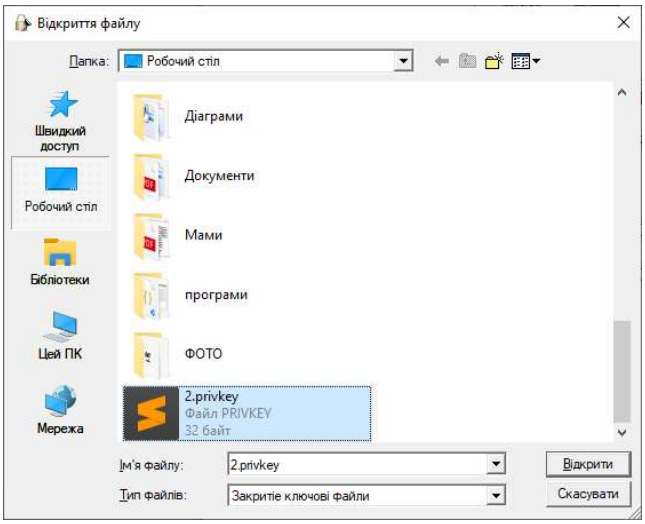

Рисунок 3.28 - Імпорт закритого ключа

Наступним кропок потрібно імпортувати закритий ключ що до заповнити даним які потрібно для того ми могли розшифрувати текстовий файл (рисунок 3.29).

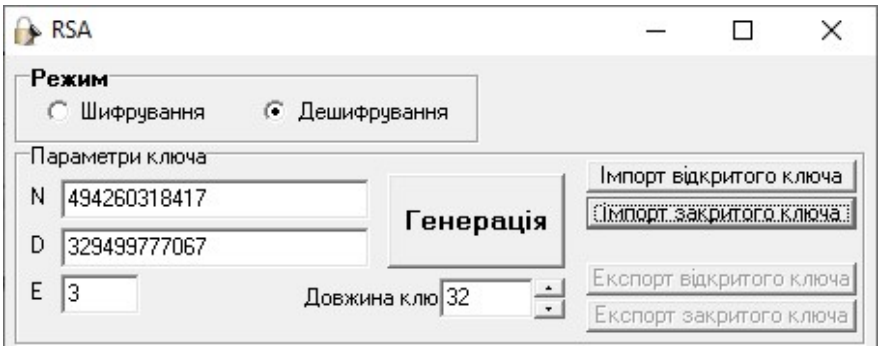

Рисунок 3.29 – Імпорт закритого ключа

Перевіряємо чи заповнились відповідні колонки даними і розшифровуємо (рисунок 3.29).

Тоді нам потрібно імпортувати імпортувати зашифрований файл в якому містяться зашифровані дані, для того щоб можна було розшифрувати повідомлення (рисунок3.30).

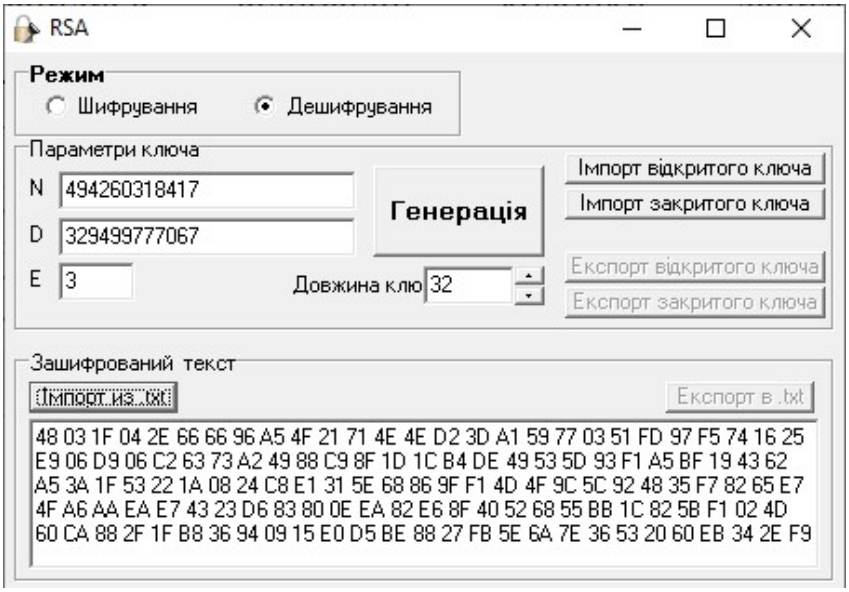

Рисунок 3.30 - Імпортований зашифрований текст

Файл було успішно завантажено і цей фалй готовий до розшифрування звотнім користувачем, для того щоб розшифрувати дані, залишилось нажати

кнопку «Розшифрувати», в нижній колонці, це зайняло декілька секунд де було отримано розшифроване повідомлення. Яке ми можемо експортувати назад у текстовий формат(рисунок 3.31).

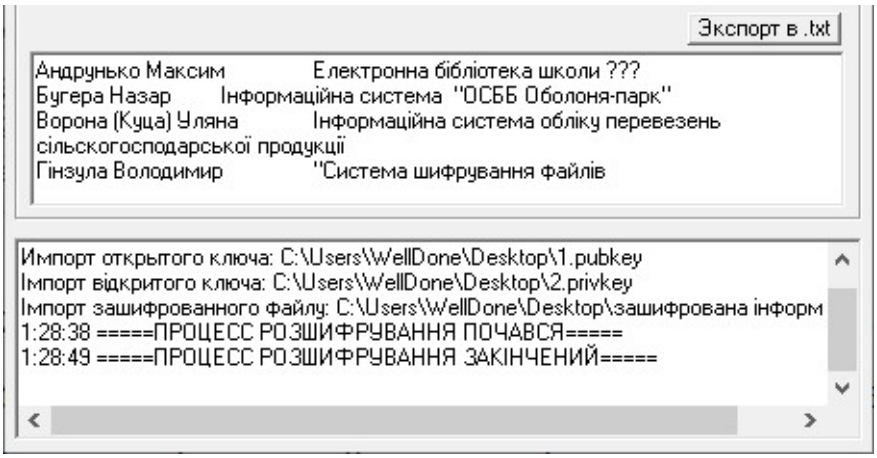

Рисунок 3.31 – Розшифроване повідомлення

Вагома частина є перевірення усіх ключів , якщо буде відсутність ключа це не дасть змогу розшифрувати файл, а це значить що дані були перехоплені частново і попали не в ті руки , отже захист присутній на достатньому рівні

Отже тестування є вагомою частиною при створенні програмного застосунку, було проведено тести які показали спавність роботи програми, цілісність файлів, швидко дію програми.

Тестування пройдено згідно плану, перевірка інтерфейсу користувача та функціонал. Програма працює правильно без видимих промлем. Встановлення застосунку не становить перешкод і запускається належним чином, при виконані поставлених завдань про програма працює корентно.

Таким чином, проведення тестування програмного забезпечення є ключовим етапом у його створенні. Процес у якому проводиться тестування має критерії та особливості, на які слід звернути свою увагу. Додаток для шифрування та розшифрування повідомлень легкий у використанні та не потребує якигось професійних навичок. Користувачі зможуть безпечно передавати свої повідомлення або файли не боячись їх розсекречення.

4 ТЕХНІКОЕКОНОМІЧНЕ ОБГРУНТУВАННЯ

4.1 Аналіз ринку

Програмне забезпечення захисту інофрмації та збереження цілісності файлів є групою програмного забезпечення у яких реалізують можливе збереження та захист даних, можливість їх надсилання у зашифрованому вигляді, а також не потрапляння до не бажаних рук. Програмний засіб було створено та с проєктовано з метою виконання шифрування файлів алгоритмом RSA. Розвиток інформаційних технологій які займаються захистом, надають широкий спектр можливостей щодо захисту даних, проте гнучкі адаптивні засоби асиметричного шифрування мають досить високу вартість використання, тому проєктування власного засобу є економічно вигідним та обгрунтованим. Інформація специфічний ресурс, не маючи доступу до іформації сучасна людита не зможе існувати

Будь яка створена програма чи реальний продук чи віртуальний продукт він потребує нагляду. Так само із програмами для шифрування, якщо не догялдати програми чи робоити нових оновлень то це може призвести до зменшення користувачів або зменшення захисту інформації.

4.2 Розрахунок витрат на розробку проєкту

Розробка та заснування проєкту це складний проєкт який включає в себе урахування усіх економічних портеб. Проведення дослідження на попиту: обсягу, динамічного розвитку ринку, наявність потреби шифро програми, затвердило вагомість заснування проєкту. Динамічне зростання ринку вільна торгівля на цсьому ринку, присутність користувачів – є першоначальними критеріями підготовки до розрахування витрат на проєктування програми.

Підрахунок витрат які в будь якому випадку буде використано для виконання цього проєкту наведенно в таблиці 4.1.

Таблиця 4.1 – Список витрат на створення проєкту

Призначення позицій витрат	Сума, грн	Пояснення
1. Заробітня плата програміста	9000	
2. Вирахування на соціальні потреби	990	ЄСП – 11%
3. Контрагентські роботи і послуги	3510	39%
4. Витрати на відрядження	0	
5. Інші прямі витрати	1080	9000 – 12%
6. Усього прямих витрат	4500	3510 + 990
7. Накладні витрати	945	4500 – 21%
8. Планові накопичення	935,55	(3510 + 945) - 21%
9. Усього, кошторисна вартість проєкту	10925,55	9000+990+935,55
10. Податок на додану вартість	2.185,11	10925 – 20%
11. Загалом, договірна ціна розробки ПЗ	13110,55	10925,55+2.185

Заробітня плата робітникам калькулюється на одного учасника, розробника, як це показано в таблицы 4.2.

Таблиця 4.2 - розрахунок платні

N	Посада	Оклад	Податок	Кількість		
п/п	виконавця	грн/міс	грн/міс	Чол.	Місяців	з/п, грн
1	Програміст - Juior	5000	1000	1	3	4000
2	Програміст - middle	7000	1400	1	2	5600
3	Тестувальник	4500	900	1	1	3900

4.3 Обгрунтування необхідності та розробки

У сучасномі світі розвиток ІТ є дуже швидким, і з різким темпом ростуть системи захисту інформації, а для того щоб тримати дані у безпеці потрібні відповідні програми захисту. Саме ця мета створення даного проєкту – захист даних. Створення таких програм є сукупністю методів захисту інформації. Дана програма забезпечує достаній захист важливих текстових документів а з урахуванням легкості в користуванні, що не є у всіх програмних засобах вона стає ще кращою і дає змогу користуватись усім охочим. Шифрування файлів є одним з найбезпечніших методів захисту інформації. Інформація специфічний ресурс, не маючи доступу до іформації сучасна людита не зможе існувати, захист інформації в даному середовищі є дуже важливим, маючи доспут до інформації будуть і люди які мають змогу отримати вигоду з цієї інформації, створюючи шифрувальні прогмами відбувається захист тої самої інформації яку можна вкрасти та використати проти людити з метою вимагання або шантажу.

Для того щоб бути високо на ринку програм які займаються захистом, потрібно створювати відповідні програми, ці програми повинні ретельно шифрувати та захищати інформацію щоб було більше користувачів відповідно і дохід а з ним і спонсорування.

При правилному поставленому завданні, та при правильному підході до її створення така програма буде мати не аби який попит в захисті інформації.

ВИСНОВКИ

У дипломному проєкті було проаналізовано предметну області та поставлено завдання, а саме розробка програмного засобу для шифрування файлів алгоритмом RSA. При постановці завдання було визначено функціонал програми який вона повинна виконувати. Було розглянуто інфраструктура відкритих ключів що використовується в асиметричний криптографії. Також описано симетричні алгоритми.

Проведено дослідження алгоритму RSA як стандарту асиметричного шифрування. Проаналізовано стійкість алгоритму RSA та обчислювальної складності проблем на яких він базується. Було досліджено створення закритого та відкритого ключа та цей ключ потрібен для шифрування повідомлення а щоб розшифрувати його нам потрібно закритий ключ.

Проведено розробку та тестування програмного засобу для шифрування алгоритмом RSA. Виконано проєктування інтерфейсу програмного засобу, проєктування основних функцій алгоритму RSA та проведено тестування в якому перевірений увесь функціонал програми на працездатність та пробне шифрування файлів для перевірки коректності роботи програми.

Виконано техніко-економічне обґрунтування де було проведено аналіз ринку засобів шифрування та способи реалізації розробленого програмного засобу, проведено розрахунок витрат на проєктування та обґрунтування необхідності розробки даного проєкту.

Перелік джерел посилань

1. Що таке шифрування RSA і як це працює? Веб-сайт. URL: - Режим доступу:https://instagalleryapp.com/informacijna-bezpeka/shho-take-shifruvannja-rsa-i-jak-ce-pracjue/ (дата звернення : 10.05.2022)

2. Що таке симетричне шифрування? Веб-сайт. URL: - Режим доступу: https://uk.theastrologypage.com/symmetric-encryption (дата звернення: 11.05.2022)

3. Асиметритчні медоти шифрування. Веб-сайт. URL: - Режим доступу: https://dut.edu.ua/uploads/l_491_94183247.pdf (дата звернення : 11.05.2022)

4. RSA – Шифрування на пальцях. Веб-сайт. URL: - Режим доступу: http://www.michurin.net/computer-science/rsa.html (дата звернення : 12.05.2022)

5. Програмування C++ для початківців. Веб-сайт. URL: - Режим доступу: https://purecodecpp.com/uk/ (дата звернення : 12.04.2022)

6. Документація по C++. Веб-сайт. URL: - Режим доступу: https://stackoverflow.com/questions (дата звернення : 14.04.2022)

7. Розробка графічного інтерфейсу Веб-сайт. URL: - Режим доступу: https://cxemotexnika.org/uk/2020/12/rozrobka-grafichnogo-interfejsu-za-dopomogoyu-wxwidgets/ (дата звернення : 16.04.2022)

8. Навчальні посібники по RAD Studio 11 Веб-сайт. URL: - Режим доступу:https://docwiki.embarcadero.com/RADStudio/Sydney/en/Tutorials (та зверненн:19.04.2022)

Додатки

Додаток А

Лістинг програмного засобу

```
//---------------------------------------------------------------
---------------

#include <vcl.h>

#pragma hdrstop

//---------------------------------------------------------------
---------------

USEFORM("UTForm.cpp", Form1);

//---------------------------------------------------------------
---------------

WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)

{

        try

        {

                Application->Initialize();

                Application->Title = "RSA";

                Application->HelpFile                          =
"C:\\Users\\Dim565\\Desktop\\1352473611_icon.bmp";

                Application->CreateForm(__classid(TForm1),
&Form1);

                Application->Run();

        }

        catch (Exception &exception)

        {

                Application->ShowException(&exception);

        }

        catch (...)

        {

                try
```

```
                {
                        throw Exception("");
                }
                catch (Exception &exception)
                {
                        Application-
>ShowException(&exception);
                }
        }
        return 0;
}
//---------------------------------------------------------------
---------------

#include <vcl.h>
#pragma hdrstop


#include "UTForm.h"
#include "UTRSA.h"
//---------------------------------------------------------------
---------------

#pragma package(smart_init)
#pragma resource "*.dfm"
extern _INFO Info;
TForm1 *Form1;
//---------------------------------------------------------------
---------------

__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
randomize();
}
```

```
//---------------------------------------------------------------
---------------

void __fastcall TForm1::Button1Click(TObject *Sender)

{

History->Lines->Add(TimeToStr(Time())+" =====ГЕНЕРАЦІЯ КЛЮЧІВ
РОЗПОЧАТА=====");

TRSA a(Edit4->Text.ToInt());

char* tmp = a.KeyModul;

KEY_N->Text = AnsiString(tmp);

tmp = a.OpenKey;

KEY_E->Text = AnsiString(tmp);

tmp = a.SecretKey;

KEY_D->Text = AnsiString(tmp);

History->Lines->Add("Добуток:
N=p*q="+(IntToStr(Info.InfoP))+"*"+IntToStr(Info.InfoQ)+"="+KE
Y_N->Text);

History->Lines->Add("Функція     Эйлера:     phi(N)=(p-1)*(q-
1)="+FloatToStr(Info.InfoPhi));

History->Lines->Add("Відкрита експонента: E="+KEY_E->Text);

History->Lines->Add("Секретна експонента: D*E=1 mod phi(N);");

History->Lines->Add("D*"+KEY_E->Text+"=1 mod "+Info.InfoPhi+";
D="+KEY_D->Text);

History->Lines->Add("Пара E, N - відкритий ключ RSA (public
key): "+KEY_E->Text+"; "+KEY_N->Text);

History->Lines->Add("Пара D, N - закритий ключ RSA (private
key): "+KEY_D->Text+"; "+KEY_N->Text);

History->Lines->Add(TimeToStr(Time())+" =====ГЕНЕРАЦІЯ КЛЮЧІВ
ЗАВЕРШЕНА=====");

Button2->Enabled=True;

Export_Publ->Enabled=True;

Export_Priv->Enabled=True;

}

//---------------------------------------------------------------
---------------
```

```
void __fastcall TForm1::Button2Click(TObject *Sender)

{

if(Memo1->Text != "")

{

if (KEY_N->Text !="" || KEY_E->Text !="")

{

History->Lines->Add(TimeToStr(Time())+"           =====ПРОЦЕСС
ШИФРУВАННЯ ПОЧАВСЯ=====");

TNumber n(KEY_N->Text.c_str());

TNumber e(KEY_E->Text.c_str());

TNumber d(KEY_D->Text.c_str());

THexString text;

text.AsCharString = Memo1->Text;

  TRSA a(n,e,Edit4->Text.ToInt());

  a.PlainText = text;

  Memo3->Text = a.CryptedText.AsHexString;

  Button6->Enabled = True;

  Button4->Enabled=True;

  History->Lines->Add(TimeToStr(Time())+"         =====ПРОЦЕСС
ШИФРУВАННЯ ЗАКІНЧЕНО=====");

}

else

ShowMessage("Відсутній відкритий ключ");

}

else

ShowMessage("Відсутній текст для шифрування");

}

//---------------------------------------------------------------
---------------

void __fastcall TForm1::Button4Click(TObject *Sender)

{
```

```
if(Memo3->Text != "")

{

if (KEY_N->Text !="" || KEY_D->Text !="")

{

History->Lines->Add(TimeToStr(Time())+"            =====ПРОЦЕСС
РОЗШИФРУВАННЯ ПОЧАВСЯ=====");

TNumber n(KEY_N->Text.c_str());

TNumber e(KEY_E->Text.c_str());

TNumber d(KEY_D->Text.c_str());

THexString text;

text.AsHexString = Memo3->Text;

  TRSA a(n,e,d,Edit4->Text.ToInt());

  a.CryptedText = text;

  Memo2->Text = a.PlainText.AsCharString;

  Button5->Enabled = True;

  History->Lines->Add(TimeToStr(Time())+"          =====ПРОЦЕСС
РОЗШИФРУВАННЯ ЗАКІНЧЕНИЙ=====");

  }

else

ShowMessage("Відсутній відкритий ключ");

}

else

ShowMessage("Відсутній текст для дешифрування");

}

//---------------------------------------------------------------
---------------

void __fastcall TForm1::Button3Click(TObject *Sender)

{

TOpenDialog *od = new TOpenDialog(this);

od->Filter = "Текстові файли|*.txt";

if (od->Execute()) {
```

```
  Memo1->Lines->LoadFromFile(od->FileName);


}
History->Lines->Add("Імпорт відкритого файла: "+od->FileName);
delete od;
od = NULL;
}
//---------------------------------------------------------------
---------------
void __fastcall TForm1::Button5Click(TObject *Sender)
{
TSaveDialog *od = new TSaveDialog(this);
od->Filter = "Текстовые файли*.txt";
if( od ->Execute())
 {
      Memo2->Lines->SaveToFile(od ->FileName + ".txt");
      History->Lines->Add("Експорт   розшифрованного   файлу:
"+od->FileName);
 }
delete od;
od = NULL;
}
//---------------------------------------------------------------
---------------
void __fastcall TForm1::Button6Click(TObject *Sender)
{
TSaveDialog *od = new TSaveDialog(this);
od->Filter = "Текстові файли|*.txt";
if(od ->Execute())
 {
      Memo3->Lines->SaveToFile(od ->FileName + ".txt");
```

```
      History->Lines->Add("Експорт зашифрованного файла: "+od-
>FileName);

 }

delete od;

od = NULL;

}

//---------------------------------------------------------------
---------------

void __fastcall TForm1::Export_PublClick(TObject *Sender)

{

if (KEY_N->Text !="" || KEY_E->Text !="")

{

TStringList*s=new TStringList;

s->Add(KEY_E->Text);

s->Add(KEY_N->Text);

s->Add(Edit4->Text); //длина ключа

TSaveDialog *od = new TSaveDialog(this);

od->Filter = "Відкриті ключові файли |*.pubkey";

if(od ->Execute())

 {

      s->SaveToFile(od ->FileName + ".pubkey");

      delete s;

      History->Lines->Add("Експорт  відкритого  ключа  :  "+od-
>FileName+".pubkey");

 }

delete od;

od = NULL;

}

else

ShowMessage("Значення N і E пусті!");

}
```

```
//---------------------------------------------------------------
---------------
void __fastcall TForm1::Export_PrivClick(TObject *Sender)
{
if (KEY_N->Text !="" || KEY_D->Text !="")
{
TStringList*s=new TStringList;
s->Add(KEY_D->Text);
s->Add(KEY_N->Text);
s->Add(Edit4->Text); //довжина ключа
TSaveDialog *od = new TSaveDialog(this);
od->Filter = "Закриті ключові файли|*.privkey";
if(od ->Execute())
 {
      s->SaveToFile(od ->FileName + ".privkey");
      delete s;
      History->Lines->Add("Експорт  закритого  ключа:  "+od-
>FileName+".privkey");
 }
delete od;
od = NULL;
}
else
ShowMessage("Значення N і D пусті!");
}
//---------------------------------------------------------------
---------------
void __fastcall TForm1::Import_PublClick(TObject *Sender)
{
TStringList*s=new TStringList;
TOpenDialog *od = new TOpenDialog(this);
```

```
od->Filter = "Відкриті ключові файли|*.pubkey";
if (od->Execute()) {
  s->LoadFromFile(od->FileName);
  KEY_E->Text=s->Strings[0];    //Перенос значення E
  KEY_N->Text=s->Strings[1];    //Перенос значенн N
  Edit4->Text=s->Strings[2];       //Перенос  значенн  довжини
ключа
  delete s;
  Button2->Enabled=True;
}
History->Lines->Add("Импорт открытого ключа: "+od->FileName);
delete od;
od = NULL;
}
//---------------------------------------------------------------
---------------
void __fastcall TForm1::Import_PrivClick(TObject *Sender)
{
TStringList*s=new TStringList;
TOpenDialog *od = new TOpenDialog(this);
od->Filter = "Закритіе ключові файли|*.privkey";
if (od->Execute()) {
  s->LoadFromFile(od->FileName);
  KEY_D->Text=s->Strings[0];    //Перенос значення D
  KEY_N->Text=s->Strings[1];    //Перенос значення N
  Edit4->Text=s->Strings[2];       //Перенос  значення  довжини
ключа
  delete s;
  Button4->Enabled=true;
}
History->Lines->Add("Імпорт відкритого ключа: "+od->FileName);
```

```
delete od;
od = NULL;
}
//---------------------------------------------------------------
---------------
void __fastcall TForm1::Button7Click(TObject *Sender)
{
TSaveDialog *od = new TSaveDialog(this);
od->Filter = "Текстові файли|*.txt";
if( od ->Execute())
 {
      Memo3->Lines->SaveToFile(od ->FileName + ".txt");
      History->Lines->Add("Експорт   зашифрованного   тексту:
"+od->FileName);
 }
delete od;
od = NULL;
}
//---------------------------------------------------------------
---------------
void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
if (RadioGroup1->ItemIndex==0)
{
ShowMessage("Вітання");
}
}
//---------------------------------------------------------------
---------------
void __fastcall TForm1::RadioButton1Click(TObject *Sender)
{
if (RadioButton1->Checked)
```

```
{
Group_Decode->Visible=False;
Group_Code->Visible=True;
Button4->Visible=False;
Button3->Visible=True;
Button2->Visible=True;
Button8->Visible=False;
Group_Shifr->Top=320;
}
}
//---------------------------------------------------------------
---------------
void __fastcall TForm1::RadioButton2Click(TObject *Sender)
{
if (RadioButton2->Checked)
{
Group_Decode->Visible=True;
Group_Decode->Top=320;
Group_Code->Visible=False;
Button3->Visible=False;
Button4->Visible=True;
Button2->Visible=False;
Button8->Visible=True;
Group_Shifr->Top=160;
}
}
//---------------------------------------------------------------
---------------
void __fastcall TForm1::Button8Click(TObject *Sender)
{
 TOpenDialog *od = new TOpenDialog(this);
```

```
od->Filter = "Текстові файли|*.txt";

if (od->Execute()) {

  Memo3->Lines->LoadFromFile(od->FileName);

   Button4->Enabled=True;

}

History->Lines->Add("Імпорт зашифрованного файлу: "+od-
>FileName);

delete od;

od = NULL;

}

//---------------------------------------------------------
---------------

void __fastcall TForm1::Edit4Change(TObject *Sender)

{Button2->Enabled=false;

}//---------------------------------------------------------
----------------

void __fastcall TForm1::FormCreate(TObject *Sender)

{

History->Lines->Add(TimeToStr(Time())+"            =====ЗАПУСК
ПРОГРАММИ=====");

}

//---------------------------------------------------------
---------------

void __fastcall TForm1::UpDown1Click(TObject *Sender,
TUDBtnType Button)

{

KEY_N->Text="";

KEY_D->Text="";

}

//---------------------------------------------------------
---------------

#pragma hdrstop

#include "UTHexString.h"
```

```
#include <sysutils.hpp>
#include <math.hpp>
#include <math.h>
//---------------------------------------------------------------
---------------
#pragma package(smart_init)
//---------------------------------------------------------------
---------------
THexString::THexString()                    //
{
el = new unsigned int[1];
dim = 0;
}
//---------------------------------------------------------------
---------------
THexString::THexString(const AnsiString Source)
{                                           //
el = new unsigned int[1];
AsHexString = Source;
}
//---------------------------------------------------------------
---------------
THexString::THexString(const THexString &Source)
{                                           //
dim = Source.dim;
el = new unsigned int[dim];
for (int i = 0; i < dim; i++)
  el[i] = Source.el[i];
}
//---------------------------------------------------------------
---------------
THexString::~THexString()                   //
```

```
{
delete[] el;
}
//---------------------------------------------------------------
--------------
int THexString::HexCharToInt(const char &h)
{                                              //
if (h == '0') return 0;
if (h == '1') return 1;
if (h == '2') return 2;
if (h == '3') return 3;
if (h == '4') return 4;
if (h == '5') return 5;
if (h == '6') return 6;
if (h == '7') return 7;
if (h == '8') return 8;
if (h == '9') return 9;
if (h == 'A' || h == 'a') return 10;
if (h == 'B' || h == 'b') return 11;
if (h == 'C' || h == 'c') return 12;
if (h == 'D' || h == 'd') return 13;
if (h == 'E' || h == 'e') return 14;
if (h == 'F' || h == 'f') return 15;
return -1;
}
//---------------------------------------------------------------
--------------
char THexString::IntToHexChar(unsigned int digit)
{                                              // 1
if (digit > 15)
throw Exception("hex must be [0:15]");
```

```
if (digit <=  9) return AnsiString(digit)[1];
if (digit == 10) return 'A';
if (digit == 11) return 'B';
if (digit == 12) return 'C';
if (digit == 13) return 'D';
if (digit == 14) return 'E';
return 'F';
}
//---------------------------------------------------------------
---------------
void THexString::SetHexString(AnsiString Source)
{                                                    // Запись как 16-
ричная строка
Source = Source.Trim();
if ((Source.Length() + 1)%3 != 0)
  throw Exception("Длина строки не соответствует формату");
delete[] el;
dim = (Source.Length() + 1)/3;
el = new unsigned int[dim];
unsigned int digit1,digit2;
for (int i = 1; i <= dim; i++)
  {
  digit1 = HexCharToInt(Source[3*i-2]);
  digit2 = HexCharToInt(Source[3*i-1]);
  if (digit1 == -1 || digit2 == -1)
    throw Exception("'"+Source+"' is not string of hex.");
  if (i*3 <= Source.Length())
    if (Source[i*3] != ' ')
      throw Exception("'"+Source+"' is not string of hex.");
  el[i-1] = digit1*16 + digit2;
  }
```

```
}
//---------------------------------------------------------------
---------------
AnsiString THexString::GetHexString()      //Перевод исходного
текста в HEX-код
{
AnsiString result;
result = "";
int digit1,digit2;
for (int i = 0; i < dim; i++)
  {
  digit2 = el[i]%16;
  digit1 = (el[i] - el[i]%16) / 16;
  result += IntToHexChar(digit1);
  result += IntToHexChar(digit2);
  result += " ";
  }
result = result.Trim();
return result;
}

//---------------------------------------------------------------
---------------
AnsiString THexString::GetCharString()     //
{
AnsiString result = "";
for (int i = 0; i < dim; i++)
  result += (char)el[i];
return result;
}
//---------------------------------------------------------------
---------------
```

```
void THexString::SetCharString(AnsiString Source)
{                                                //
dim = Source.Length();
delete[] el;
el = new unsigned int[dim];
for (int i = 0; i < dim; i++)
  el[i] = (unsigned char)Source[i+1];
}
//---------------------------------------------------------------
---------------
int THexString::GetDim()                         //
{return dim;}
//---------------------------------------------------------------
---------------
void THexString::XOR(const THexString &op) //
{
if (dim != op.dim)
  throw Exception("Ошибка!");

for (int i = 0; i < dim; i++)
  el[i] = el[i]^op.el[i];
}
//---------------------------------------------------------------
---------------
void THexString::AddWOExtend(const THexString &op)
{                                                //
if (dim != op.dim)
  throw Exception("Ошибка!");
int rem = 0;
for (int i = 0; i < dim; i++)
  {
```

```
  el[i] = op.el[i] + el[i] + rem;
  rem = (el[i] - el[i] % 256)/256;
  el[i] = el[i] % 256;
  }
}
//---------------------------------------------------------------
---------------
THexString& THexString::operator = (THexString &a)
{                                              //
AsHexString = a.AsHexString;
return *this;
}
//---------------------------------------------------------------
---------------
AnsiString  THexString::GetBinString()
{
AnsiString Result = "";
for (int i = 0; i < dim; i++)
  {
  AnsiString res;
  int todiv = el[i];
  int rem;
  for (int j = 0; j < 8; j++)
    {
    rem = todiv%2;
    todiv = Floor(todiv/2);
    res = AnsiString(rem) + res;
    }
  Result += res;
  }
return Result;
```

```
}
//---------------------------------------------------------------
--------------
void THexString::SetBinString(AnsiString Source)
{if (Source.Length() % 8 != 0)
  throw Exception("Ошибка!");
dim = Source.Length() / 8;
delete[] el;
el = new unsigned int[dim];
for (int i = 0; i < dim; i++)
  {
  unsigned int value = 0;
  for (int j = 0; j < 8; j++)
    {
    if (Source[i*8 + j + 1] != '0' && Source[i*8 + j + 1] !=
'1')
      throw Exception("Строка должна состоять из 0 и 1");
    if (Source[i*8 + j + 1] == '0')
      value = value * 2;
    else
      value = value * 2 + 1;
    }
  el[i] = value;
  }
}
//---------------------------------------------------------------
--------------
THexString& THexString::SubString(int pos, int count)
{AnsiString ResStr;
ResStr = this->AsCharString;
ResStr = ResStr.SubString(pos,count);
```

```
THexString *Result;
Result = new THexString();
Result->AsCharString = ResStr;
return *Result;
}//---------------------------------------------------------------
----------------
THexString& THexString::operator + (THexString &a)
{AnsiString res = this->AsCharString;
res += a.AsCharString;

THexString *Result;
Result = new THexString();
Result->AsCharString = res;
return *Result;
}
//---------------------------------------------------------------
---------------
unsigned int& THexString::operator[] (int index)
{return el[index];}
//---------------------------------------------------------------
---------------
```