МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ЦЕНТР ЗАОЧНОЇ, ДИСТАНЦІЙНОЇ ТА ВЕЧІРНЬОЇ ФОРМ НАВЧАННЯ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Кваліфікаційна робота бакалавра

Клієнт-серверний застосунок для анонімного спілкування

Здобувач освіти гр. Інз-81с Владислав ЧУЙКО

Науковий керівник, Галина ОЛЕКСІЄНКО

кандидат ф.-м. наук, доцент

Завідувач кафедри Анатолій ДОВБИШ

доктор технічних наук,професор.

Суми 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Центр заочної, дистанційної і вечірньої форм навчання

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедрою Довбиш А.С.

"_____" _____2022 р.

ЗАВДАННЯ

до кваліфікаційної роботи бакалавра

Студента 4-го курсу, групи Інз-81с спеціальність "122 – Комп'ютерні науки заочної форми навчання Чуйка В.О.

Тема: Клієнт-серверний застосунок для анонімного спілкування

Затверджена наказом по СумДУ

№_____ от _____2022 р.

Зміст пояснювальної записки: 1) Аналіз предметної області; 2) Класифікація підходів побудови криптосистем; 3) Визначення вимог до системи; 4) Дослідження алгоритмів шифрування.4) Оцінка ефективність системи у рамках її функціонального тестування;

Дата видачі завдання "_____"_____2022 р.

Керівник випускної роботи _____ Олексієнко Г.А

Завдання прийняв до виконання _____ Чуйко В.О

РЕФЕРАТ

Записка: 33 стор., 6 рис., 3 табл., 1 додаток, 16 джерел.

Об’єкт дослідження – асиметричні криптосистеми.

Мета роботи – створення криптозахищеної системи для передачі повідомлень між її користувачами.

Методи дослідження — асинхронні методи шифрування даних

Результати – розроблено програмне забезпечення, яке реалізує один із можливих варіантів використання асинхронної криптографічної системи. Виконується захист повідомлень, шляхом їх шифрування та подальшим дешифруванням.

ЗМІСТ

ВСТУП…… 7

1 ІНФОРМАЦІЙНИЙ ОГЛЯД 8

1.1.Класифікація підходів побудови криптосистем. 8

1.2.Дослідження алгоритмів шифрування. 12

1.3.Постановка задачі. 18

2.ОПИС МЕТОДУ ДОСЛІДЖЕННЯ 19

2.1.Визначення вимог до програмного застосунку. 19

2.2.Загальні положення криптографічного алгоритму. 20

3.Програмна реалізація 23

3.1.Умови та обмеження використання. 23

3.2.Тестування та оптимізація програмного застосунку. 24

3.3 Короткий опис програмної реалізації 25

ВИСНОВОК 27

СПИСОК ЛІТЕРАТУРИ 28

ДОДАТОК 30

ВСТУП

Будь-яка інформація має свою цінність, але на відмінну від фізичного об’єкту її складніше зберегти та надійно захистити. Більше того не кожний предмет вдасться відтворити. У свою чергу інформація копіюється надзвичайно легко, достатньо третій особі її почути і вона вже, хоч і у вигляді суб’єктивної інтерпретації, але продубльована у свідомості людини.

Звісно, що найнадійнішим способом передавання цінної інформації є її пряма передача між суб’єктами без посередників. Наприклад, особиста розмова. Проте подібний варіант не можливий у багатьох випадках. Тоді виникає потреба у створенні закритого каналу зв’язку. Найпростіший спосіб його зробити – це почати шифрувати повідомлення.

У сучасному світі кожен з нас взаємодіє з цифровою інформацією, але далеко не кожен розуміє, що для її передачі використовуються криптографічні системи різного рівня складності. Починаючи від зберігання, закінчуючи передачею дані постійно шифруються та дешифруються. Це робиться для того, щоб зберегти цінність інформації. Є безліч алгоритмів та підходів для забезпечення надійного захисту криптографічної системи, у рамках бакалаврської роботи якраз буде розглянуто один із можливих варіантів їх використання.

Під час дослідження буде проведений аналіз підходів для побудови криптосистем, та розглянуто типові алгоритму шифрування. Результатом роботи повинно стати програмне забезпечення, яке здатне передавати повідомлення між користувачами по криптозахищеному каналу зв’язку. При цьому потрібно звернути увагу на характерні проблеми подібних систем.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1. Класифікація підходів побудови криптосистем.

Най надійнішим способом захисту цифрової інформації є її шифрування [1]. Навіть маючи доступ до даних їх не можливо буде використати, оскільки у зашифрованому вигляді вони, зазвичай, не представляють цінності. Звісно, що це може бути використано й для шантажу, коли зловмисники у такий спосіб блокують доступ до особистих даних користувача, але у рамках дослідження буде зосереджена увага лише на захисті інформації.

У загальному вигляді шифрування [1] – це процес перетворення інформацій у форму, яку можуть прочитати лише конкретні користувачі. При цьому отриманий шифротекст повинен задовольняти наступні умови:

- ***Конфіденційність***. Дані повинні бути надійно приховані від сторонніх людей.
- ***Цілісність***. При перетворенні інформації дані не повинні втрачатися.
- ***Ідентифікованість***. Інформація повинна бути надіслана та прочитана лише конкретною людиною.

Більшість користувачів навіть не підозрюють при використанні цифрової інформації, що вона проходить етапи шифрування та розшифрування. Так, наприклад, відбувається захист особистих даних в інтернет ресурсах, захищаються кошти при використанні онлайн-банкінгу і.т.д. У корпоративному середовищі шифрування зберігає конфіденційність інтелектуальної власності. Інакше кажучи, криптосистеми, іноді в неявному вигляді, використовуються для передачі майже будь-якої важливої інформації.

Ясна річ, що зараз цей процес автоматизований за допомогою сучасних комп’ютерів, але сама ідея шифрування далеко не нова [2]. Історія захисту інформації бере свій початок ще приблизно з 1300 року до н.е. Тоді з’явився один із перших задокументованих представників алгоритмів шифрування – Атбаш. Хоча людство пройшло досить значний шлях розвитку, але принципи

захисту інформації не змінилися [1]. Так для побудови та розшифрування криптограми потрібен деякий ключ. Єдине, що тут є досить не очевидна проблема, а саме ключ повинен знаходитися одразу у двох сторін. Отримувача та відправника. Тобто потрібно попіклуватися про те, щоб забезпечити надійність такої передачі, бо на практиці, зазвичай, вона буде відбуватися по не захищеному каналу зв'язку.

У сучасній літературі прийнято розділяти алгоритми шифрування на: симетричні та асиметричні. У загальному випадку цей поділ відбувається за типом ключа, що впливає на варіанти застосування криптосистеми.

Симетричні системи шифрування [3] передбачають використання одного криптографічного ключа:

$$E_k(P) = C$$
$$D_k(C) = P$$

де E – функція шифрування;
k - це криптографічний ключ;
P – це відкритий текст;
C – шифротекст;
D – функція розшифрування;

При цьому справджується наступна рівність:

$$D_k\big(E_k(P)\big) = P$$

При такій схемі шифрування обмін інформацією відбувається у декілька етапів:

1. користувачі обмінюються спільним ключем, який буде використаний для заздалегідь обраного алгоритму;
2. відправник зашифровує повідомлення та відправляє його отримувачу;
3. отримувач розшифровує повідомлення;

При цьому важливо, щоб симетрична система відповідала ряду вимог. Головна з яких - це видалення будь-якої статистичної закономірності в об'єкті

шифрування. Для цього, зазвичай, використовують складну багаторівневу комбінацію підстановок та перестановок вхідного масиву даних. Таким чином вдається позбутися лінійності інформації. Окрім цього надійність системи можна підвищити шляхом використання унікальних ключів, які будуть змінюватися, наприклад, при кожному новому сеансі.

Усі симетричні алгоритми шифрування можна поділити на групи:

1. **Потокові шифри** [5]. Інформація обробляється послідовно і побітово. Це досить зручно, адже дозволяє перервати шифрування у будь-який момент часу і продовжити його без втрат.
2. **Блочні шифри** [7]. Перетворення інформації відбувається через зміну блоків бітів фіксованої довжини. Зазвичай, цей підхід використовується при пакетній передачі даних чи файлів.

У асиметричних ж алгоритмах шифрування використовується два ключі. Один відкритий. Він потрібний для безпосереднього шифрування і може передаватися по не захищеним каналам зв’язку. Другий ж приватний або закритий. Він використовується для розшифрування даних. Ці два ключа пов’язані між собою деякою функцією, але із-за обчислювальної складності вкрай важко вивести один із іншого. Найбільшу популярність асиметричні системи набули при створенні цифрового підпису.

Схема обміну інформації при асиметричному шифруванні наступна:

- суб’єкт А генерує публічний та приватний ключ. Перший він передає суб’єкту В, а другий ж залишає при собі;
- суб’єкт В за допомогою закритого ключа шифрує своє повідомлення і відправляє суб’єкту А;
- суб’єкт А за допомогою приватного ключа розшифровує повідомлення;

У такий спосіб навіть якщо зловмисник отримає повідомлення, воно саме по собі не буде мати цінності, адже для відтворення початкової інформації потрібен приватний ключ. Це вирішує фундаментальну проблему симетричних

систем. Річ у тім, що ключ, так чи інакше, потрібно передати і якщо він один як для шифрування так й для розшифрування, то це спрощує роботу зловмиснику.

Між симетричним і асиметричним підходом є досить не очевидна відмінність [8]. Це довжина ключів. При симетричному шифруванні вона обирається випадковим чином, але мало коли перевищує 128 або 256 бітів, це вже більше залежить від бажаного рівня безпеки. Але в асиметричній системі є два ключі і між ними повинний бути математичний зв'язок, який утворюються через деяку функцію. Але із-за цього зловмиснику легше розкрити шифр, бо він може вивести цю математичну закономірність. Саме тому асиметричні ключі характеризуються більшим розмірами, а сам алгоритм шифрування виконується довше ніж це роблять симетричні криптосистеми. Таким чином, щоб забезпечити приблизно еквівалентний рівень безпеки асиметричні алгоритми повинні використовувати 2048-бітні ключи, порівняно з 128-бітними в симетричних системах.

Завдяки своїй швидкості симетричні криптосистеми використовуються в переважній більшості державних та воєнних установах. Наприклад, Advanced Encryption Standard (AES) [10] застосовується урядом США для передачі секретної інформації.

У свою чергу асиметричні системи використовуються, коли потрібно надати інформацію одразу багатьом користувачам. При цьому швидкість чи обчислювальне навантаження не є пріоритетом. До таких задач можна віднести вже згаданий цифровий підпис або шифрування електронної пошти.

Основним критерієм при оцінці будь-якого криптографічного алгоритму є його стійкість. Цю властивість можна описати як здібність системи протистояти криптоаналізу. У найкращому випадку шифр потребує від зловмисника майже не можливих обчислювальних ресурсів або часу, який буде витрачений на розкриття, зробить інформацію не актуальною. Цікавим є те, що стійкість більшості сучасних криптосистем майже не можливо підтвердити математично, але можна довести її вразливості.

Криптоалгоритм вважається ідеально стійкими, якщо для його розкриття потрібно перебрати усі можливі варіанти ключів. Враховуючи, що вірогідність підібрати правильний ключ після перебору половини варіантів, згідно теорії ймовірностей, дорівнює $\frac{1}{2}$, то для розкриття ідеально стійкого шифру з ключем, який має довжину N, потрібно $2N-1$ перевірок. Це означає, що складність алгоритму частково, а іноді й повністю, залежить від довжини ключа і зростає експоненціально. Навіть якщо припустити наявність спеціального обладнання, де в багатопроцесорній системі за рахунок діагонального паралелізму на перевірку 1 ключа йде тільки 1 такт, то на розкриття 128 бітного ключа піде не менше ніж 1021 рік. Звичайно, що це відноситься лише до ідеально стійких шифрів.

1.2. Дослідження алгоритмів шифрування.

Досить велику кількість сучасних алгоритмів займають блочні шифри, які представляють із себе симетричну криптосистему. Їх особливістю є, як не дивно, робота з блоками даних, які вони перетворюють у ланцюжки зашифрованих байтів. Це дозволяє їм шифрувати пакети інформації з не обмеженою довжиною. Цю особливість використовують у гібридних системах, де блочний алгоритм нівелює низьку швидкість асиметричного шифрування. Окрім цього між блоками шифротексту немає кореляційного зв’язку, тому їх можна використовувати, наприклад, для обчислення контрольної суми пакетів даних або при хешуванні паролів.

Крім цього група блочних шифрів виділяється ще однією цікавою особливістю. При відомому вихідному та зашифрованому блоку даних ключ, який виконував перетворення повинен бути знайдений лише повним перебором. Річ у тім, що ситуації, коли відомо частина початкового повідомлення досить розповсюджена. Це можуть бути заголовки, шаблоні записи і т.д. Таким чином стійкі блочні шифри повинні відповідати наступним додатковим вимогам:

- функція шифрування повинна бути зворотною;

- не повинно існувати жодних можливостей прочитати вхідне повідомлення за відомим блоком, окрім повного перебору усіх можливих ключів;
- не повинно існувати жодних можливостей відтворити вхідне повідомлення з шифротексту, окрім повного перебору усіх можливих ключів;

Стандарт шифрування даних DES (Data Encryption Standard) [12] – блочний шифр зі симетричними ключами, який був розроблений Інститутом Стандартів і Технологій (NIST – National Institute of Standards and Technology).

Після публікації алгоритм жорстко критикувався по двом причинам. Перша: відносно маленький розмір ключа в 56 бітів, що могло зробити криптосистему вразливою до атаки «грубою силою». Друга причина: критики переживали за скриту побудову внутрішньої структури DES. Вони вважали, що блоки мають скритий недолік через який Національне агентство по безпеці США зможе розшифровувати повідомлення без ключа. Пізніше IBM, щоб унеможливити криптоаналіз, допрацювала внутрішню структуру.

Федеральний регістр об’явив DES стандартом шифрування і він швидко став найбільш вживаним блочним шифром. Пізніше NIST запропонував оновлену версію, яка представляла із себе трикратно повторений шифр DES. В 2000 році новий стандарт AES замінив DES.

Алгоритм перетворює 64-бітне повідомлення M у 64-бітний шифротекст C. Якщо кожен блок шифрується індивідуально, то такий підхід має назву: Electronic Code Book (ECB) [13]. Проте, існують більш продвинуті парадигми, які пропонують робити кожен блок повідомлення залежним від попередніх, наприклад: Chain Block Coding (CBC) або Cipher Feedback (CFB).

DES – це блочний шифр, тобто він працює з блоками відкритого тексту M заданого розміру (64 біта) і повертає блоки шифротексту C того ж розміру. Таким чином, DES зводиться до перестановки 2^{64} можливих конфігурацій із 64 бітів, кожен із яких може приймати значення 1 або 0. Блоки діляться на дві рівні частини L і R, кожна по 32 біти.

Також існують методи, які використовуються спільно з основною криптосистемою. Так, наприклад, Алгоритм Діффі-Хеллмана [14] допомагає обмінюватися секретним ключем для симетричних криптосистем з використанням каналу, захищеного від модифікації, тобто дані, передані цим способом, можуть бути прослухані, але не змінені. У є можливість створити однаковий секретний ключ, жодного разу не передавши його по мережі.

Припустимо, що обом учасникам обміну відомі деякі два натуральні числа M і N, де N – просте число. Вони можуть бути відомі й усім іншим зацікавленим особам. Для того щоб створити невідомий більше нікому секретний ключ, обидві сторони генерують випадкові числа: перший абонент – число X, другий абонент – число Y. Потім перший учасник обчислює L_A за наступною формулою.

$$L_A = M^X(mod(N))$$

Саме L_A буде передано отримувачу, але він, в свою чергу, теж може створити зашифроване повідомлення L_B:

$$L_B = M^Y(mod(N))$$

Схематично процес передачі зашифрованих ключів можна показати наступним чином, рисунок 1.1.

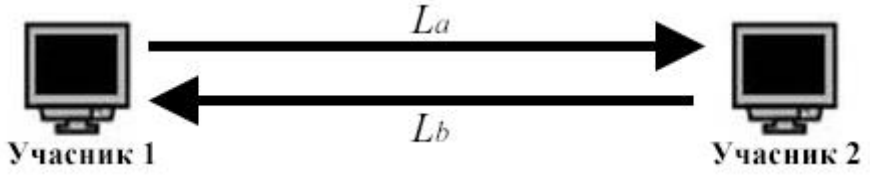


Рисунок 1.1. Процес передачі зашифрованих ключів

На другому етапі перший абонент, на основі наявного в нього числа X й отриманого по мережі числа L_B, зможе розшифрувати переданий йому ключ. Теж саме може зробити і другий учасник але з повідомленням від першого.

Слід зазначити, що обидва користувача, після дешифрування, отримають однакові повідомлення $K_A = K_B$. Це можна перевірити наступним чином:

$$K_A = L_B^X(mod(N)) = (M^Y)^X(mod(N)) = M^{YX}(mod(N)) = M^{XY}(mod(N)) = (M^X)^Y(mod(N)) = L_A^Y(mod(N)) = K_B$$

Числа K_A, K_B, можуть використовуватися як ключі при обміні даними з використанням традиційних систем шифрування.

При спробі визначити значення чисел X та Y за перехопленим значенням L_A і L_B супротивник зіштовхнеться із задачею дискретного логарифмування, що є важко розв'язною. Проте, слід завжди пам’ятати, що алгоритм Діффі-Хеллмана може повноцінно працювати лише в умовах каналу, який захищений від модифікацій даних.

Як зазначалося раніше, один із можливих варіантів використання асиметричних систем є цифровий підпис, який також використовується в сучасній криптографії. Elliptic Curves Digital Signature Algorithm (ECDSA) [15] – це алгоритм, який на основі еліптичних кривих і кінцевих полів створює підпис для даних. При цьому будь-яка людина може верифікувати його, але не підробити. Це використовується, наприклад, при транзакціях криптовалют, коли виконується передача права власності. Цей метод застосування має свою власну назву Elliptic Curve Cryptography (ECC).

Концепцію ECC незалежно один від одного запропонували математики Ніл Кобліц та Віктор С. Мілер у 1985 році. Але свою популярність ця модель почала отримувати після 2000-го року, коли її поступово почали використовувати інтернет-провайдери і вже потім в системах криптовалют. Головною перевагою моделі є відносно, як для асиметричного алгоритму, не великий розмір ключа.

Сама ідея криптовалюти досить цікава і відображає тенденції сучасного світу. У традиційному своєму вигляді термін «володіти» означає або безпосередньо зберігати певний об’єкт, або передати його до довіреної структури. Але це не відноситься до криптовалюти, вони існують у вигляді записів блокчейну, копії яких розподілені між пов’язаними комп’ютерами. «Володіти» криптовалютою – це мати можливість надавати доступ до неї третім

особам. Якраз для цього використовується ECDSA, який реалізує дві процедури: одна для підпису, а інша для верифікації. При цьому в першому випадку потрібний приватний ключ, а в другому публічний.

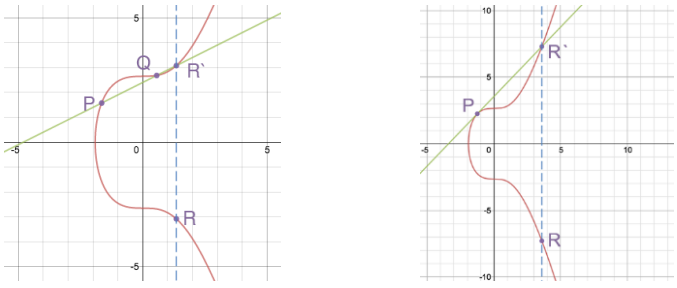
Алгебраїчно еліптичну криву, яка є однією із основ ECDSA, можна представити наступним рівнянням:

$$y^2 = x^3 + ax + b$$

Ця функція має ряд унікальних властивостей, наприклад, невертикальна лінія, яка проходить через будь-які дві точки кривої завжди буде проходити і через третю. По-друге, невертикальна дотична лінія завжди перетне деяку точку на кривій. Ці властивості можна використати для реалізації двох важливих операцій.

Перша з них це складання. Тобто через, наприклад, точки P і Q проводиться деяка пряма, яка однозначно перетне криву в точці R'. Після цього знаходиться точка на кривій, яка симетрична, відносно осі x, до R', воно і буде представляти із себе суму P і Q, рисунок 1.2.а.

Аналогічним чином при подвоєнні точок проводиться пряма, яка є дотичною до еліптичної кривої в деякій точці P і породжує, згідно описаної вище властивості, ще R'. Залишається знайти лише точку R, яка повинна бути симетрична, відносно осі x, до R'. Це і буде результатом подвоєння P, рисунок 1.2.б.



а б

Рисунок 1.2. Приклад операцій над еліптичною кривою

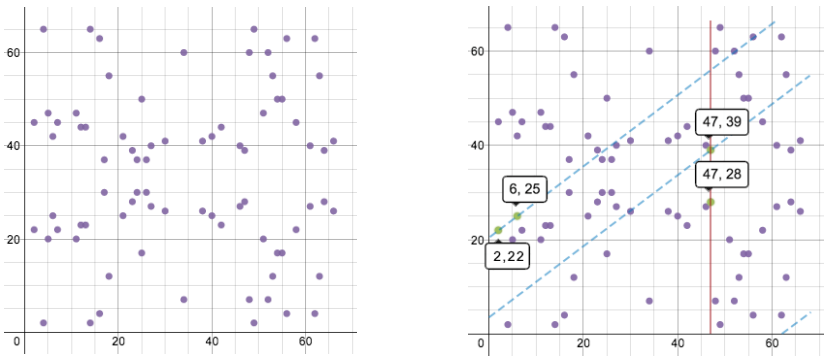
Ці дві операції необхідні для спрощення множення на скаляр, де деяку точку R ми додаємо саму на себе x разів. Наприклад, операцію 7P можна

замінити на послідовне виконання спочатку двох дій подвоєння, а після додавання.

$R = 7P$
$R = P + 6P$
$R = P + 2(3P)$
$R = P + 2(P + 2P)$

Другою важливою концепцією ECDSA є кінцеві поля, які можна представити у вигляді деякого діапазону можливих додатних значень в який повинен потрапляти результат. При цьому алгоритм працює до тих пір, поки кожне число не стане частиною кінцевого поля. Наприклад, якщо ми візьмемо операцію $9/7$, то отримаємо 1 з залишком $9\,mod(7) = 2$. У цьому випадку кінцеве поле буде від 0 до 6, і всі числа по модулю 7, будуть потрапляти в цей діапазон. Це і називається кінцевим полем.

ECDSA використовує еліптичні криві у вигляді як раз кінцевого поля, це дещо їх видозмінює, але ключові властивості зберігаються. Наприклад, кінцеве поле для операції по модулю 67 буде мати наступний вигляд, рисунок 1.3.а. Тепер це набір точок, де x і y приймають цілочисельне значення від 0 до 66. Складання та подвоєння візуально будуть виглядати дещо інакше. Обернені значення будуть знаходитися в іншій частині поля, зберігаючи той самий нахил. Наприклад, складання точок (2; 22) і (6; 25) буде мати наступний вигляд, рисунок 1.3.б. Обернена пряма, яка проходить через задані точки, перетне (47; 39), а симетрична, по осі x, до неї точка це (47; 28).



а б

Рисунок 1.3. Приклад операцій над кінцевим полем

Розглянемо приклад, як ECDSA працює з біткойном. Протокол обирає набір параметрів для еліптичної кривої і репрезентує його на деяке кінцеве поле. Перед усім обирається саме рівняння прямої, просте значення модуля поля і базову точку на кривій. При цьому для їх обрання використовуються над великі числа, цим забезпечується надійність протоколу.

І так можна стверджувати, що сучасні алгоритми шифрування виконують ряд найрізноманітніших задач, які у тому чи іншому вигляді пов’язані з захистом інформації. При цьому іноді алгоритми чи парадигми доповнюють один одного перекриваючи свої загальновідомі недоліки.

1.3. Постановка задачі.

У рамках бакалаврської роботи необхідно розробити програмне забезпечення, яке б дозволяло обмінюватися приватними повідомленнями між користувачами. При цьому потрібно звернути увагу на типові проблеми подібних систем та запропонувати їх вирішення:

- захист конфіденційної інформації;
- забезпечення надійності каналу зв’язку;
- ідентифікованість повідомлень;

У загальному вигляді план виконання роботи був розбитий на наступні етапи:

1. Визначення вимог до програмного застосунку;
2. Обрання методу шифрування даних;
3. Розробка програмного забезпечення на мові програмування C#;
4. Тестування та оптимізація програмного застосунку;
5. Визначення перспектив для подальшого покращення роботи;

2. ОПИС МЕТОДУ ДОСЛІДЖЕННЯ

2.1. Визначення вимог до програмного застосунку.

Вимоги до технології розробки:

Продукт розробляється і ітеративному режимі, з врахуванням технологій уніфікованого процесу створення програмного забезпечення. До етапу здачі проекту потрібно надати, як мінімум, один варіант повноцінно працюючого прототипу, який би виконував базові функції продукту. Програма повинна розроблятися на мові програмування C#, з використанням фреймворку Microsoft.Net.

Вимоги до інформаційної та програмної сумісності:

Інтерфейс програми повинен бути оформлений українською мовою. Користувач повинен мати змогу отримувати повідомлення та відображати їх у відповідному вікні. У нього повинна бути можливість ініціювати канал зв’язку, тобто виступати у ролі хосту або підключитися до вже створеного. Будь-який елемент інтерфейсу повинен бути попередньо узгоджений.

Програмний застосунок створений для роботи під платформу Windows у межах локальної мережі. Для його коректного використання необхідний .Net Framework версії 4.5 або вище.

Вимоги до функціональних характеристик:

Програмний застосунок повинен виконувати наступні базові функції:

- обмін повідомленнями між користувачами;
- забезпечення конфіденційності інформації:
  - шифрування повідомлення;
  - надійне передання криптографічних ключів;
- створення захищеного каналу зв’язку між користувачами;

2.2. Загальні положення криптографічного алгоритму.

Найпершою криптосистемою з публічним ключем, із запропонованих у відкритій літературі, в 1978 році стала система Райвеста, Шаміра і Едлмана. Нині відома як RSA [5, 16].

RSA представляє із себе блочний шифр, де шифрований і відкритий текст подаються як цілі числа із діапазону від 0 до n-1. Надійність алгоритму ґрунтується на труднощі факторизації великих чисел і обчисленні дискретних логарифмів.

У рамках алгоритму RSA для шифрування, та розшифрування, використовується пара ключів – відкритий та приватний. Для їх створення нам необхідні два простих числа p і q, тобто таке, що ділиться без остачі тільки на себе і на одиницю. Тут важливо підмітити, що довжина таких чисел може бути буквально будь-якою. Але для простоти прикладу візьмемо наступні значення:

$$p = 11;\ \ q = 13$$

Наступним етапом при створені ключів є отримання деякого числа n, яке має назву модуль порівняння. По своїй суті це добуток p на q, , для нашого випадка:

$$n = p * q = 11 * 13 = 143$$

Тепер необхідно знайти деяку величину *F(n),* яка має назву функція Ейлера. Вона представляє із себе кількість натуральних чисел, які менші ніж n, але при цьому є взаємно простими з ним. Знайти *F(n)* можна за досить простою формулою:

$$F(n) = (p - 1) * (q - 1) = 10 * 12 = 120$$

Бажано, щоб *F(n)* було простим числом. У даному випадку, для перевірки отриманого результату його необхідно розбити на прості множники:

$$120 = 2 * 2 * 2 * 3 * 5$$

Наступним крок – підбір числа e, яке повинно відповідати двом критеріям:

1) Бути меншим від n;

2) Не мати спільних множників з *F(n)*;

Це число e має назву відкрита експонента. Вважається, що краще обрати його найменше значення з усіх можливих. Для нашого випадку було обране наступне:

$$e = 7$$

Останнє, що потрібно для роботи RSA – це число *d,* так звана, закрита експонента. Вона повинна мати таке значення, щоб при підставленні його в рівняння $e * d - 1$ результат націло ділився на *F(n).* У нас було обране наступне число:

$$d = 103$$

На цьому етапі підготовка для відправлення повідомлень завершується. У складені два ключі:

Приватний – (n,e) = (143, 7)
Публічний – (n,d) = (143, 103)

Тепер припустимо, що у нас є деяке повідомлення M, яке слід відправити по не захищеному каналу зв'язку. Тоді потрібно передати адресату публічний ключ, за допомогою якого він зможе зашифрувати конфіденційне повідомлення за наступною формулою:

$$\mathrm{c}_i = m_i^e (mod(n)), \text{ де}$$

m_i- це і-й елемент (блок) повідомлення;
c_i – це і-й елемент (блок) криптограми;
(n, e) – елементи відкритого ключа.

Розшифрування буде відбуватися на стороні одержувача за допомогою приватного ключа, згідно наступної формули:

$$m_i = c_i^d (mod(n)), \text{ де}$$

m_i- це і-й елемент (блок) повідомлення;
c_i – це і-й елемент (блок) криптограми;
(n, d) – елементи приватного ключа.

Уявімо, що зловмисник зміг отримати публічний ключ, який ми передали по незахищеному каналу зв’язку. У подальшому він може перехопити і повідомлення, які були зашифровані цим ключем, але дешифрувати їх йому не вдасться, бо це можна зробити лише за допомогою приватного криптоключа, який не передавався між сторонами, рисунок 2.1.

Рисунок 2.1. Схема асиметричного шифрування RSA

Чи вдасться зловмиснику отримати приватний ключ, якщо він має публічний? Звісно, що це можливо, адже між криптоключами в асиметричній системі є певний математичний взаємозв’язок. Але для розкриття потрібно розкласти отриманий ключ на прості множники, а це далеко не просте завдання, особливо коли річ йде про числа, які складаються із десятків чи навіть сотні елементів. Власне за рахунок цього алгоритм RSA досі активно використовується.

Серед недоліків алгоритму RSA можна виділити, характерний для всіх асинхронних методів, довгий час роботи. Так на етапі генерації ключів нам потрібно не просто два випадкових числа, а встановити чи прості вони. Нажаль на сьогоднішній день немає достатньо швидких алгоритмів визначення цієї властивості чисел. Але тут є ще одна проблема. Річ у тім, що криптоключі повинні бути не залежні від будь-яких чинників, але сам факт того, що нам потрібні безпосередньо лише прості числа, порушує цю умову.

Також спільною проблемою для всіх асинхронних методів є довжина їх ключів, бо вона напряму впливає як на надійність так й на швидкодію алгоритму.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1. Умови та обмеження використання.

Розроблений програмний застосунок призначений для роботи в локальній мережі, де дозволяє двом користувач вести обмін повідомленнями по криптозахищеному каналу. Для початку роботи потрібно запустити програму та обрати свою роль. Можна бути або хостом, і очікувати на підключення користувача, або клієнтом, тоді потрібно під'єднатися до існуючого хосту. У першому випадку необхідно явно вказати айпі адрес комп’ютера клієнта та порт, через який він буде встановлювати зв’язок з нами, рисунок 3.1. У другому ж сценарії необхідно вказати айпі адрес серверу та порт через який можливо з ним з’єднатися.

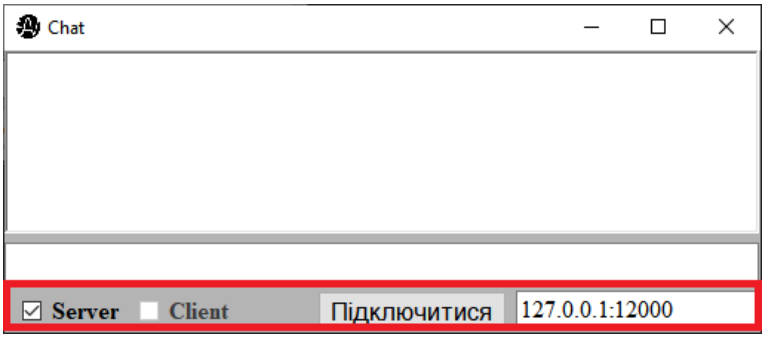


Рисунок 3.1. Приклад створення хосту

Важливо щоб хост відкрив доступ до підключення раніше ніж клієнт відправить запит, бо інакше він не з’єднатися з сервером. Якщо ж підключення пройшло успішно, то обидва комп’ютери можуть почати спілкування. Текст вводиться у спеціальне поле як це показано на рисунку 3.2. І виводиться у інше з вказанням псевдоніму відправника. Програма обмежується лише двома абонентами, тобто «ви» і «гость» і на це є свої причини.

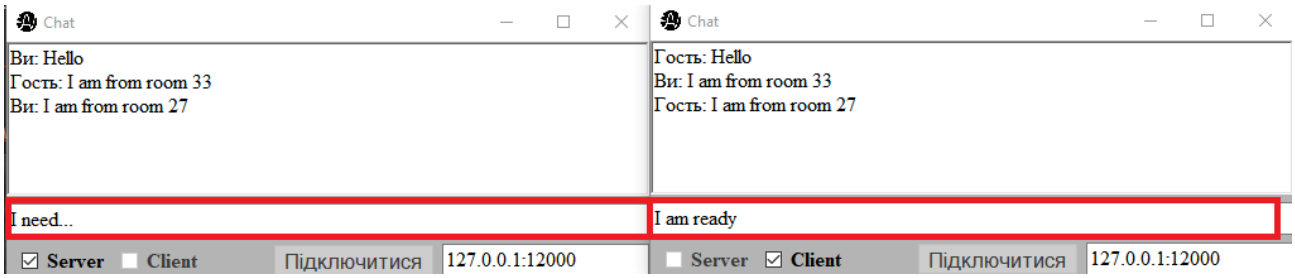


Рисунок 3.2. Приклад переписки між абонентами

Конфіденційність інформації забезпечується за рахунок створення криптозахищеного каналу зв’язку через шифрування повідомлень алгоритмом RSA. При цьому кожен абонент має власну пару ключів. Так, наприклад, суб’єкт А відправляє свій публічний криптоключ суб’єкту В, а він, в свою чергу, надає власний відкритий ключ користувачу А. Таким чином вони обмінюються повідомленнями, які зашифровані двома різними способами, що дещо підвищує надійність захисту.

Обмеження кількості користувачів, на даному етапі, дозволяє зменшити обчислювальне навантаження на систему, бо інакше б кожному абоненту потрібно було б на власному пристрої одночасно розшифровувати довільні за розміром повідомлення від усіх учасників спілкування.

3.2. Тестування та оптимізація програмного застосунку.

У ході використання програми виникла необхідність у розпаралелюванні функцій. Таким чином, очікування повідомлення від співрозмовника відбувається в окремому потоці. Це було зроблено для того, щоб користувач мав можливість виконувати інші дії, поки програма чекає на відповідь від абонента.

```
void read() {
    while (reading)
    {
        //не можемо прочитати повідомлення, коли не має зв'язку
        string text = "";
        try
        {
            text = sr.ReadLine();
        }
        catch
        {
            MessageBox.Show("Зв'язок з гостьовим комп'ютером втрачений.
Програма припиняє роботу",
            "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
            reading = false;
        }

        if (text != null)
        {
            string message = cipher.decryption(text);
            Action action = () => richTextBox1.AppendText("Гость: " + message);
            var chatText = (string)richTextBox1.Invoke(new Func<string>(() =>
richTextBox1.Text));
```

```
            if (chatText.Length != 0)
                if (chatText[chatText.Length - 1] != '\n')
                    action = () => richTextBox1.AppendText(Environment.NewLine
+ "Гость: " + message);
            Invoke(action);
        }
    }
}
```

Окрім цього в коді були зроблені виключення для деяких не передбачуваних ситуацій та виведення інформаційних повідомлень, у випадку збою програми.

У таблиці 3.1. наведений перелік програмних функцій, які можна реалізувати при подальшій розробці застосунку.

Функція	Опис
Робота з багатьма користувачами	Реалізувати переписку, яка ведеться одночасно для оптимальної кількості абонентів. При цьому, можна враховувати їх унікальні імена, які б вони обирали самі для себе
Шифрування публічних ключів	Для забезпечення додаткового рівня захисту можна виконувати шифрування публічних ключів при передачі їх по каналу зв'язку
Динамічна зміна ключів	Зміна ключів в залежності від сеансу, дня тижня тощо. Тобто при, наприклад, кожному з'єднані використувавти нову пару криптоключів.
Відправка псевдоповідомлень	Нехай, час від часу користувачі будуть обмінюватися псевдоповідомленнями, які будуть містити не зв'язний зашифрований текст. Такі обманки можуть підвищити надійність захисту системи, бо зловмисник отримавши їх буде витрачати марно час на розкриття. Також вони вплинуть на статистичну інформацію, яку той може збирати.
Відправка файлів	Як переспективне оновлення можна додати можливість відправляти файли, але тоді потрібно попіклуватися про додатковий метод їх шифрування.

3.3 Короткий опис програмної реалізації

Програмне забезпечення було написане на мові програмування C#. Нижче наведений опис реалізованих класів. У своєму повному обсязі код знаходиться в Додатку.

Клас Form виконує взаємодію із користувацьким інтерфейсом, отримує вхідні дані.

Таблиця 3.2 - Опис складових класу Form

ЗМІНІ			
Назва	Тип	Модифікатор доступу	Призначення
threads	Thread[]	private	Список усіх створених потоків
sw	StreamWriter	private	Потік через який відбувається відправка повідомлень
sr	StreamReader	private	Потік через який відбувається читання повідомлень
cipher	Cipher	private	Об’єкт, який здійснює шифрування та розшифрування повідомлень
reading	bool	private	Умова роботи потоку на читання

Методи	
Назва	Призначення
button1_Click(object sender, EventArgs e)	Парсінг адреси та встановлення зв’язку з абонентом
server(string ip, int port)	Налаштування користувача як хосту
client(string ip, int port)	Налаштування користувача як клієнта
write(string text)	Відправка повідомлення
read()	Очікування повідомлення від абонента
Form_FormClosed(object sender, System.Windows.Forms.FormClosedEventArgs e)	Закриття форми
textBox2_KeyPress(object sender, KeyPressEventArgs e)	Формування тексту на відправлення

Клас Cipher виконує шифрування та дешифрування повідомлення.

Таблиця 3.3 - Опис складових класу Cipher

ЗМІНІ			
Назва	Тип	Модифікатор доступу	Призначення
n	int	private	Модуль порівняння RSA
e	int	private	Відкрита експонента RSA
d	int	private	Закрита експонента RSA

Методи	
Назва	Призначення
string encryption(string message)	Шифрування повідомлення
string decryption(string cryptogram)	Дешифрування повідомлення

ВИСНОВОК

1) Розглянуто підходи та варіанти використання криптографічних систем для збереження цінності цифрової інформації. При цьому зосереджено увагу на огляді недоліків та слабких місць способів захисту даних.

2) Розроблено програмне забезпечення, яке реалізує один із можливих варіантів використання асинхронної криптографічної системи. Виконується захист повідомлень, шляхом їх шифрування та подальшим дешифруванням.

3) Як метод шифрування був обраний RSA. При цьому особливість реалізації полягає в тому, що алгоритм працює з двох сторін. Тобто кожен учасник має свою пару відкритих та приватних ключів. Це дозволяє забезпечити спілкування між ними, а також надає додатковий рівень захисту.

4) Під час розробки програмний застосунок був протестований та змінений для позбавлення виявлених недоліків.

5) За результатами аналізу роботи програми було спланово список майбутніх покращень, які включають в себе підвищення рівня надійності захисту інформації. Окрім цього можливе впровадження розробленої системи в інші сфери діяльності, наприклад, оборонну [4, 9, 11]

СПИСОК ЛІТЕРАТУРИ

1. Kessler, Gary (November 17, 2006). "An Overview of Cryptography". Princeton University

2. "History of Cryptography". Binance Academy. [Електронний ресурс] – https://academy.binance.com/en/articles/history-of-cryptography

3. "Demystifying symmetric and asymmetric methods of encryption". Cheap SSL Shop. 2017-09-28.

4. Protsenko O., Savchenko T., Myronenko M., Prikhodchenko O. Informational and extreme machine learning for onboard recognition system of ground objects. Proceedings - 2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies, DESSERT 2020. Pages: 213-218

5. Matt J. B. Robshaw. Stream Ciphers Technical Report TR-701, ver. 2.0, RSA Laboratories, 1995.

6. Буняк В.М., Шелепало (Крайнічук) Г.В. Аналіз криптосхеми над квазігруповими кільцями // Матеріали L науково-технічної конференції ВНТУ. 2021

7. Горбенко, І. Д., Олійников, Р. В., Казимиров, О. В., Руженцев, В. І., Кузнєцов, О. О., Горбенко, Ю. І., ... & Мордвінов, Р. І. (2015). Симетричний блоковий шифр Калина"–новий національний стандарт України. Радиотехника, (181), 5-22.

8. Гвоздецька, К. П., & Філіпєва, М. В. (2021). Порівняння симетричного і асиметричного шифрування.

9. Dovbysh, A., Naumenko, I., Myronenko, M., Savchenko, T. Information-extreme machine learning on-board recognition system of ground objects with the adaptation of the input mathematical description. 3rd International Workshop on Computer Modeling and Intelligent Systems, CMIS 2020; National University "Zaporizhzhia Polytechnic "Zaporizhzhia; Ukraine; 27 April 2020 to 1 May 2020; CEUR Workshop Proceedings, Volume 2608, 2020, Pages 913-925.

10. Daemen, J., & Rijmen, V. (1999). AES proposal: Rijndael.

11. Naumenko, I. Information-extreme machine training of on-board recognition system with optimization of RGB-component digital images / I. Naumenko, M. Myronenko, T. Savchenko // Radioelectronic and Computer Systems. — 2021. — № 4. — C. 59-70.

12. Matsui, M. (1993, May). Linear cryptanalysis method for DES cipher. In Workshop on the Theory and Application of of Cryptographic Techniques (pp. 386-397). Springer, Berlin, Heidelberg.

13. Kumari, M., Gupta, S., & Malik, A. (2020). A superlative image encryption technique based on bit plane using key-based electronic code book. Multimedia Tools and Applications, 79(43), 33161-33191.

14.Merkle, R. C. (1978). Secure communications over insecure channels. Communications of the ACM, 21(4), 294-299.

15. Johnson, D., Menezes, A., & Vanstone, S. (2001). The elliptic curve digital signature algorithm (ECDSA). International journal of information security, 1(1), 36-63.

16. Rivest, R. L., Shamir, A., & Adleman, L. M. (2019). A method for obtaining digital signatures and public key cryptosystems. In Secure communications and asymmetric cryptosystems (pp. 217-239). Routledge.

ДОДАТОК

```
using System;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Threading;
using System.Windows.Forms;

namespace Client
{
    public partial class Form : System.Windows.Forms.Form
    {
        public Form()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            richTextBox1.ReadOnly = true;
        }

        private void checkBox1_CheckedChanged(object sender, EventArgs e)
        {
            if (checkBox1.Checked)
                checkBox2.Enabled = false;
            else
                checkBox2.Enabled = true;
        }

        private void checkBox2_CheckedChanged(object sender, EventArgs e)
        {
            if (checkBox2.Checked)
                checkBox1.Enabled = false;
            else
                checkBox1.Enabled = true;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            if (textBox1.Text != "" && (checkBox1.Checked ||
checkBox2.Checked)) {
                //парсінг строки
                int indSplit = textBox1.Text.IndexOf(':');
                string ip = textBox1.Text.Substring(0, indSplit);
                int port =
Convert.ToInt16(textBox1.Text.Substring(indSplit+1));

                //як сервер
                if (checkBox1.Checked)
                    server(ip, port);
                //як клієнт
                else
                    client(ip, port);
```

```
                button1.Enabled = false;
            }
        }
        //створення двох потоків
        Thread[] threads;
        StreamWriter sw;
        StreamReader sr;
        Cipher cipher;
        private void server(string ip, int port) {
            //створюємо канал зв'язку
            TcpListener listner;
            //Очікування підключення
            listner = new TcpListener(new IPEndPoint(IPAddress.Parse(ip),
port));
            listner.Start();

            //Очікування підключення
            TcpClient client = listner.AcceptTcpClient();
            sr = new StreamReader(client.GetStream());
            sw = new StreamWriter(client.GetStream());
            sw.AutoFlush = true;

            cipher = new Cipher();
            //запуск потоків
            threads = new Thread[1];

            threads[0] = new Thread(() => { read(); });
            threads[0].Start();
        }

        private void client(string ip, int port)
        {
            //створюємо канал зв'язку
            TcpClient client = new TcpClient();
            try
            {
                client.Connect(new IPEndPoint(IPAddress.Parse(ip), port));
            }
            catch
            {
                MessageBox.Show("Невдалося встановити зв'язок з заданим
пристроєм",
                    "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }

            //міжмережеве написання тексту
            sw = new StreamWriter(client.GetStream());
            sw.AutoFlush = true;

            //міжмережеве зчитування тексту
            sr = new StreamReader(client.GetStream());
            cipher = new Cipher();
            //запуск потоків
            threads = new Thread[1];
```

```
            threads[0] = new Thread(() => { read(); });
            threads[0].Start();
        }
        void write(string text) {
            Action action = () => richTextBox1.AppendText("Ви: " + text);
            if (richTextBox1.Text != "")
                if (richTextBox1.Text[richTextBox1.Text.Length - 1] != '\n')
                    action = () => richTextBox1.AppendText(Environment.NewLine
+ "Ви: " + text);
            Invoke(action);
            string cryptogram = cipher.encryption(text);
            sw.WriteLine(cryptogram);
        }

        //дозвіл на читання
        bool reading = true;
        void read() {
            while (reading)
            {
                //не можемо прочитати повідомлення, коли не має зв'язку
                string text = "";
                try
                {
                    text = sr.ReadLine();
                }
                catch
                {
                    MessageBox.Show("Зв'язок з гостьовим комп'ютером втрачений.
Програма припиняє роботу",
                    "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                    reading = false;
                }

                if (text != null)
                {
                    string message = cipher.decryption(text);
                    Action action = () => richTextBox1.AppendText("Гость: " +
message);
                    var chatText = (string)richTextBox1.Invoke(new
Func<string>(() => richTextBox1.Text));

                    if (chatText.Length != 0)
                        if (chatText[chatText.Length - 1] != '\n')
                            action = () =>
richTextBox1.AppendText(Environment.NewLine + "Гость: " + message);
                    Invoke(action);
                }
            }
        }

        private void Form_FormClosed(object sender,
System.Windows.Forms.FormClosedEventArgs e)
        {
            reading = false;
            sr.Close();
            threads[0].Abort();
```

```
        }

        private void textBox2_KeyPress(object sender, KeyPressEventArgs e)
        {
            if (e.KeyChar == 13 && textBox2.TextLength > 0)
            {
                string text = textBox2.Text;
                write(text);
                textBox2.Clear();
            }
        }
    }
}



using System;
using System.Numerics;

namespace Client
{
    class Cipher
    {
        //Складові для шифрування
        int n = 143, e = 7;
        //Складові для дешифрування
        int d = 103;

        public Cipher() {

        }
        public string encryption(string message) {
            string cryptogram = "";
            foreach (char letter in message)
                cryptogram += Convert.ToChar((Int64)(Math.Pow((int)letter, e) %
n));
            return cryptogram;
        }
        public string decryption(string cryptogram) {
            string message = "";

            foreach (char letter in cryptogram)
            {
                BigInteger res = BigInteger.Pow((int)letter, d) % n;
                message += Convert.ToChar((int)res);
            }
            return message;
        }
    }
}
```