ДВНЗ «Донецький національний технічний університет»

Факультет комп'ютерно-інформаційних технологій та автоматизації

Кафедра прикладної математики та інформатики

«До захисту допущено»

Завідувач кафедри ПМІ

_____ **О. А. Дмитрієва**
(підпис) (ініціали, прізвище)

«____»_____2021 р.

Випускна кваліфікаційна робота магістра

на тему «Система шифрованого обміну повідомленнями на основі нейронної мережі з використанням деревовидних машин парності»

Виконав: студент 2 курсу, групи КНм-20
(шифр групи)

спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)

Рябко В. І. _____
(прізвище та ініціали) (підпис)

Керівник доц. каф. ПМІ, к.т.н., доц. Маслова Н.О. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Нормоконтроль:

_____ *І. А. Назарова*
(підпис)

_____
(дата)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Покровськ – 2021 р.

ДВНЗ «Донецький національний технічний університет»

Факультет комп'ютерно-інформаційних технологій та автоматизації
Кафедра прикладної математики та інформатики
Освітній ступінь магістр
Спеціальність 122 Комп'ютерні науки
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри ПМІ
_____ /О. А. Дмитрієва/
«____»_____2021 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ
Рябко Владиславу Івановичу
(прізвище, ім'я, по батькові)

1. Тема роботи Система шифрованого обміну повідомленнями на основі нейронної мережі з використанням деревовидних машин парності

керівник роботи Маслова Наталія Олександрівна, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом від « 22 » вересня 20 21 року № 570
2. Строк подання студентом роботи 7 грудня 2021 року
3. Вихідні дані до роботи результати переддипломної практики
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1) аналітичний огляд предметної області; 2) проектування і розробка системи шифрованого обміну повідомленнями; 3) експериментальні дослідження розробленої системи.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) робота містить 8 таблиць, 32 рисунка, 4 графіка та інший графічний матеріал, у кількості, достатній для демонстрації результатів кваліфікаційної роботи

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтролер	доц. Назарова Ірина Акопівна	–	–

7. Дата видачі завдання 4 жовтня 2021 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз проблемної області, дослідження інформації з різних джерел, постановка вимог і задачі	04.10.2021-14.10.2021	
2	Проектування системи	15.10.2021-01.11.2021	
3	Програмна реалізація системи	02.11.2021-17.11.2021	
4	Тестування та дослідження розробленої системи	18.11.2021-25.11.2021	
5	Оформлення пояснювальної записки та опис створеної системи. Проходження нормоконтролю	26.11.2021-02.12.2021	
6	Перевірка пояснювальної записки антиплагіатною системою. Оформлення презентації до роботи, графічного матеріалу та рецензування роботи	03.12.2021-17.12.2021	
7	Захист роботи	21.12.21	

Студент _____ **Рябко В. І.**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Маслова Н.О.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Рябко Владислав Іванович. Розробка системи шифрованого обміну повідомленнями для співробітників підприємств / Випускна кваліфікаційна робота на здобуття освітнього ступеня «магістр» за спеціальністю 122 Комп'ютерні науки. – ДВНЗ ДонНТУ, Покровськ, 2021.

Дану випускну кваліфікаційну роботу присвячено аналізу та впровадженню шифрованого обміну повідомленнями між співробітниками підприємства для досягнення необхідного рівня захисту від кріптоатак, шляхом застосування методів криптографічного аналізу.

Мета – вибір оптимальних підходів для удосконалення системи шифрування, яка дозволяє підвищити криптостійкість та зменшити час на шифрування повідомлення.

Об'єкт дослідження – процес шифрування.

Предмет дослідження – архітектури штучних нейронних мереж для задачі шифрування даних при передачі повідомлення.

Задачею є дослідження архітектури та принципів роботи штучних нейронних мереж, дослідження архітектури системи шифрування, проведення аналізу мереж для отримання ключу зв'язку.

Результатом проведених досліджень та етапу проектування і розробки настільного застосунку є автоматизована система шифрування повідомлень користувачів на основі деревовидних машин парності.

Ключові слова: Python, HTML, JavaScript, деревовидна машина парності, шифрування повідомлень.

ABSTRACT

Vladislav Riabko. Development of a system of encrypted messaging for employees of enterprises / Graduation qualifying work for obtaining an educational degree «Master» specialty 122 Computer Science. – SHEI DonNTU, Pokrovsk, 2021.

This final qualifying work is devoted to the analysis and implementation of encrypted messaging between employees of the enterprise to achieve the required level of protection against crypto attacks, by applying the methods of cryptographic analysis.

The goal is to choose the best approaches to improve the encryption system, which increases cryptographic resilience and reduces the time to encrypt the message.

The object of research is the encryption process.

The subject of research - the architecture of artificial neural networks for the task of encrypting data during message transmission.

The task is to study the architecture and principles of artificial neural networks, to study the architecture of the encryption system, to analyze the networks to obtain a communication key.

The result of the research and the stage of designing and developing a desktop application is an automated system for encrypting user messages based on tree-like parity machines.

Keywords: Python, HTML, JavaScript, tree parity machine, message encryption.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

АПК – аналіз поведінки користувача

АПКС – аналіз поведінки користувача і сутності

ДМП – деревовидна машина парності

ПЗ – програмне забезпечення

ПП – програмний продукт

СВВ – система виявлення вторгнення

СЗВ – система запобігання вторгнення

ШНМ – штучна нейронна мережа

ВСТУП

В сучасних реаліях конфіденційність в мережі Інтернет є найголовнішою вимогою всіх користувачів. Тому мета досягнення повної конфіденційності в Інтернеті є головною метою роботи відділів кіберзахисту популярних сервісів обміну повідомленнями між користувачами.

Коли користувач спілкується з кимось через сторонній додаток для обміну повідомленнями, йому слід пам’ятати, що хтось, не ваш призначений одержувач, може отримати доступ до тексту повідомлення. Подібно до посилки, залишеної без нагляду на вашому порозі, ризик того, що хтось отримає особисті речі чи інформацію, завжди високий. Конфіденційність в Інтернеті – це головне, тому програми для чату, які проголошують повну безпеку розмов, настільки популярні, незважаючи на деякі недоліки.

Метою даної кваліфікаційної роботи є розробка системи, яка надає можливість користувачеві надсилати повідомлення співрозмовнику без ризику перехоплення тексту цього повідомлення третіми особами. Завданням є дослідження архітектури та принципів роботи ШНМ, дослідження архітектури системи шифрування.

Процес шифрування оснований на методі ДМП. Вибір даного методу зумовлений тим, що він надає високий рівень захисту від вторгнення в канал зв’язку третіх осіб.

До об’єкта дослідження можна віднести процес шифрування на основі нейромережі з використанням ДМП. Предметом дослідження слід визначити архітектури ШНМ для задачі шифрування даних при передачі повідомлення.

Запланованим результатом слід визначити працездатну систему шифрованого обміну повідомленнями з високим рівнем захисту від втручання.

Під час написання пояснювальної записки на кваліфікаційну роботу магістра будуть виконані наступні завдання: проведення аналітичного огляду на обрану тематику, проектування та розробка спроектованої системи

шифрованого обміну повідомленнями, результати теоретичних та експериментальних досліджень та наведення висновків про виконану роботу.

Структура та об’єм випускної кваліфікаційної роботи магістра складається з вступу, трьох розділів, висновку, списку використаних джерел, що містить 50 найменувань, і 3 додатки. Повний обсяг роботи становить 92 сторінки друкованого тексту, з них 74 сторінок основного тексту, ілюстрованого 36 малюнками та 8 таблицями.

РОЗДІЛ 1
АНАЛІТИЧНИЙ ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

В даному розділі кваліфікаційної роботи буде проведений аналітичний огляд проблеми обраної тематики та визначення завдання.

1.1 Проблема конфіденційного обміну повідомленнями

Додатки для обміну повідомленнями все частіше використовуються для навчання, а також спілкування з родиною та друзями. Подобається це компаніям чи ні (або навіть знають про це чи ні), батьки та педагоги використовують ці служби обміну повідомленнями не лише для розваг та особистого спілкування, а й для підтримки дистанційного навчання. Але невідомо чи є пріоритетним конфіденційність при оновленні та розширенні своїх технологій.

Програми для обміну повідомленнями стали необхідністю в повсякденному житті. За таких обставин діти та учні можуть використовувати батьківський мобільний пристрій та обліковий запис батьків, щоб надсилати повідомлення вчителям або однокурсникам. Це може призвести до збору конфіденційної інформації про їхнє приватне спілкування, що може призвести до ризиків та шкоди для конфіденційності, які можуть вплинути на дітей, студентів та сім’ї. Насправді, багато сімей використовують програми для обміну повідомленнями, щоб спілкуватися один з одним, навіть коли вони всі вдома, і діти можуть писати одне одному, сидячи поруч на дивані. Це не просто заміна листів чи електронної пошти, а скоріше фундаментальна реконструкція спілкування від розмов віч-на-віч і невербальної комунікаційної системи жестів до віддаленої, онлайн- та візуальної системи спілкування з використанням тексту та символів, таких як емодзі.

Ризики перехоплення повідомлень з боку правоохоронних органів або авторитарних урядів є значними, навіть якщо вони обмежені певними людьми

або певними країнами. Розуміння того, що деякі повідомлення можуть бути прочитані, обмежує, хто може використовувати деякі продукти, якщо їх конфіденційність та безпека є слабкими. Тому шифрування надзвичайно важливе. Шифрування або навіть припущення про те, що шифрування існує, дозволяє мовленню вільно передаватись, не турбуючись про те, що ці повідомлення будуть перехоплені, прочитані та використані не в тих цілях, які передбачав відправник.

Враховуючи кількість даних, якими поширюються програми для обміну повідомленнями, та обмежений захист, який пропонує політика конфіденційності, розглянемо можливі ризики та шкоду. Те, що дані захищені та зашифровані, не означає, що вони повністю приватні від інших користувачів або для всіх. Наприклад, умови Signal – це безкоштовна програма для спілкування, яка дозволяє користувачам надсилати та отримувати приватні високоякісні повідомлення, брати участь у голосових/відеодзвінках HD та досліджувати зростаючий набір нових функцій, які допомагають користувачам залишатися на зв’язку з друзями та родиною, говорять про те, що дзвінки та повідомлення між користувачами завжди зашифровані і ніколи не можуть бути доступними чи переглянутими ніким, окрім користувача та передбачуваних одержувачів.

Однак використання служби обміну повідомленнями може дозволити одержувачу зробити особисту інформацію або вміст загальнодоступним для інших. Користувачі також повинні знати, що якщо вміст їхніх повідомлень захищено наскрізним шифруванням і заблокований від перехоплення компанією для обміну повідомленнями або правоохоронними органами, це не означає, що ризиків немає.

Інформація про наскрізний зашифрований зв’язок може все ще описувати метадані зв’язку, такі як платформа, яка використовується для надсилання повідомлення, час надсилання та отримання повідомлення, ідентифікація відправника та одержувача, а також кількість надісланих та отриманих даних повідомлення.

Підсумовуючи вищесказане можна виділити:

а) додатки для обміну повідомленнями вважаються надійними службами і можуть збирати значну кількість поведінкових даних перегляду та особистої інформації;

б) федеральний закон встановлює певні вимоги до операторів веб-сайтів або онлайн-сервісів, призначених для дітей віком до 13 років, а також до операторів інших веб-сайтів чи онлайн-сервісів, які фактично знають, що вони збирають особисту інформацію в Інтернеті від дитини віком до 13 років;

в) окрім того, додатки для обміну повідомленнями можуть зберігати метадані необмежено, та використовувати їх за власним бажанням.

Враховуйте ваше рішення, чи використовувала програма для обміну повідомленнями наскрізне шифрування, а також можливість зловживати вашою особистою інформацією, місцезнаходженням або метаданими для компанії – і будь-яких сторонніх компаній [1].

1.2 Сутність поняття штучної нейронної мережі

Нейронні мережі, також відомі як ШНМ, по суті, є штучним мозком – тисячами чи мільйонами обчислювальних одиниць, які працюють разом, щоб вирішувати проблеми, розпізнавати закономірності та приймати рішення. Так само, як і людський мозок, нейронну мережу потрібно навчити виконувати ці функції.

Цей процес навчання часто називають зворотним розповсюдженням, і він передбачає передачу нейронної мережі змодельованим завданням або проблемам, а потім надання їй зворотного зв’язку про те, чи робить вона щось правильно чи неправильно, дозволяючи ШНМ змінювати свою поведінку для досягнення наміченого результату.

Після багатьох симуляцій і вивчення прикладів нейронна мережа врешті-решт навчиться правильно реагувати, навіть якщо вона отримує абсолютно нові дані.

По суті, ШНМ використовує минулий досвід, щоб зробити правильний висновок, як і людина.

Схематичне представлення штучного нейрону наведено на рис. 1.1.

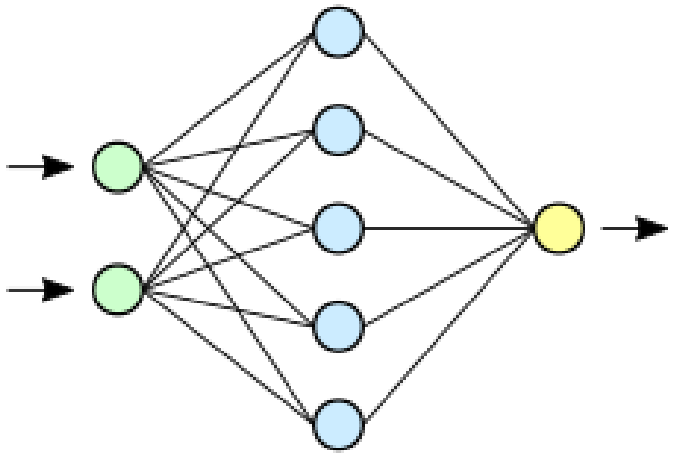


Рисунок 1.1 – Модель штучного нейрону

Застосування нейронних мереж стосуються багатьох різних галузей, включаючи виробництво, банківську діяльність та охорону здоров'я. Однак одним з найбільш широко використовуваних додатків для ШНМ є кібербезпека. Зокрема, системи виявлення та запобігання вторгненням (СВВ/СЗВ) і брандмауери нового покоління все частіше використовують передові машинне навчання та нейронні мережі, щоб допомогти виявляти аномалії, мінімізуючи хибні результати.

1.3 Забезпечення безпеки даних за допомогою ШНМ

ШНМ використовуються кількома способами, щоб підвищити безпеку даних. Далі наведені приклади застосування технологій захисту даних, які вдосконалюються за допомогою нейронних мереж.

СВВ та СЗВ.

Традиційно системи виявлення та запобігання вторгненням використовували алгоритми машинного навчання або виявлення на основі сигнатур для моніторингу активності мережі та запобігання вторгненням. Ці

системи страждають від низки обмежень, включаючи високий рівень помилкових спрацьовувань і нездатність виявляти нові атаки або ознаки розширених постійних загроз.

Деякі нові рішення СВВ та СЗВ почали використовувати технологію нейронних мереж, включаючи глибоке навчання, згорткові нейронні мережі та рекурентні нейронні мережі, щоб аналізувати мережевий трафік з більшою точністю та усувати інші обмеження традиційних систем. ШНМ краще розпізнають закономірності та ідентифікують порушення, навіть якщо вони не дотримуються встановлених векторів атаки, а ШНМ можуть автоматично вирішувати багато проблем без взаємодії з людиною. Це означає, що ваша команда витрачатиме менше часу на переслідування помилкових спрацьовувань і пом’якшення незначних загроз.

Аналіз поведінки користувачів і сутностей.

Окрім виявлення порушень, ви також можете використовувати ШНМ для моніторингу та аналізу поведінки авторизованих користувачів у вашій мережі. Внутрішні загрози становлять величезну небезпеку для безпеки ваших даних, але оскільки вони надходять від внутрішніх облікових записів користувачів, авторизованих у вашій мережі, вони часто залишаються непоміченими традиційними методами безпеки. Інструменти аналізу поведінки користувачів (АПК) існували деякий час, використовуючи алгоритми машинного навчання для виявлення аномальної поведінки облікового запису користувача в мережах.

Інструменти аналізу поведінки користувачів і сутностей (АПКС) розширюють технологію АПК, використовуючи нейронні мережі для моніторингу облікових записів користувачів, а також таких машин, як кінцеві точки, маршрутизатори та сервери. Рішення АПКС вивчають базові лінії для нормальної діяльності у вашій мережі, щоб вони могли легко помічати аномальні або підозрілі дії, такі як незвичайний час входу або велика передача даних. Інструменти АПКС вимагають величезної кількості введених даних, щоб проаналізувати вашу мережу та створити базові лінії, тому вони часто

використовуються разом із інформацією про безпеку та керуванням подіями або іншими інструментами моніторингу та аналізу даних.

Антишкідливе програмне забезпечення.

Традиційні антивірусні та антивірусні рішення працюють для запобігання вірусам та іншим видам шкідливого програмного забезпечення (також відомого як шкідливе програмне забезпечення), порівнюючи файли з базою даних відомих загроз, щоб визначити, чи є вони небезпечними. Точність цього виявлення на основі сигнатур залежить від того, як часто ця компанія оновлює свою базу даних новими загрозами – і як часто користувач оновлює визначення свого антишкідливого програмного забезпечення. Цей метод також не працює проти експлойтів нульового дня, розширених постійних загроз та нових шкідливих програм, яких раніше не було.

Рішення для захисту від шкідливих програм із використанням нейронних мереж уникають цієї проблеми кількома цікавими способами. По-перше, ви можете використовувати ШНМ для моніторингу систем і мереж, щоб виявити будь-яку аномальну поведінку, яка може вказувати на зараження або порушення зловмисного програмного забезпечення. По-друге, нейронні мережі можуть вчитися з минулих заражень (або з баз даних сигнатур загроз) і екстраполювати, як може вести себе нове шкідливе програмне забезпечення. Пам’ятайте, що однією з ключових особливостей ШНМ є їх здатність вивчати та застосовувати минулий досвід у нових ситуаціях, що дає їм явну перевагу перед системами захисту від шкідливих програм на основі сигнатур.

Виявлення спаму та фішингу.

Фільтри для спаму та фішингу – це ще один метод безпеки на основі сигнатур, який можна покращити за допомогою нейронних мереж. Зазвичай фільтри спаму працюють, оцінюючи електронні листи та вкладення на основі таких факторів, як метадані, IP-адреси, загальні ключові слова, типи файлів та посилання. Спам-фільтри порівнюють вашу електронну пошту з базою даних відомих спамів і фішингових листів, щоб визначити, наскільки ймовірно це повідомлення бути спамом. Ми всі знайомі з недоліками фільтрів спаму –

рівень помилкових спрацьовувань відносно високий, і велика кількість спамів і фішингових листів все ще проходять через більшість фільтрів.

ШНМ можуть використовувати обробку природної мови для аналізу вмісту повідомлення електронної пошти. Це означає, що замість того, щоб сканувати електронний лист для пошуку конкретних ключових слів, пов’язаних зі спамом, спам-фільтр ШНМ фактично читає повідомлення та аналізує його значення, щоб визначити, чи здається воно підозрілим. Ця технологія також адаптується до окремих користувачів та їхніх звичок спілкування електронною поштою та уподобань. Наприклад, якщо ви володієте англійською мовою, то електронний лист, написаний іншою мовою і надходить з іноземної IP-адреси, швидше за все, буде спамом або спробою фішингу. Однак ваш колега зі Східної Азії може часто надсилати й отримувати такі законні електронні листи. Спам-фільтри ШНМ можуть визначити різницю, як читаючи саме повідомлення, так і застосовуючи знання про вашу минулу поведінку та дії [2].

1.4 Недоліки використання нейронних мереж для захисту даних

Незважаючи на те, що методи захисту даних ШНМ є багатообіцяючими, є деякі проблеми, про які слід знати, якщо плануєте реалізувати їх у своєму середовищі. По-перше, треба бути дуже обережними під час навчання нейронної мережі, щоб уникнути зміщення моделі. По суті, за допомогою моделювання та зворотного зв’язку, який надається ШНМ на етапі навчання, можна ненавмисно навчити її отримувати результати, які ви хочете отримати, а не точні результати. Перш ніж розгорнути нейронну мережу і почати процес навчання, вам потрібно повністю зрозуміти природу ваших даних і пам’ятати про те, як ви визначаєте набори функцій, щоб ви могли переконатися, що ваша ШНМ працює так, як ви плануєте.

Інша проблема з нейронними мережами називається проблемою «чорного ящика». В основному, як тільки ваша ШНМ починає давати результати, може бути дуже важко, якщо взагалі неможливо, визначити, як

вона досягла цих результатів. Алгоритми, що використовуються в розширеному машинному навчанні, стають настільки складними, що навіть інженери, які їх створюють, не повністю розуміють, як вони працюють або чому поводяться певним чином. Дослідники працюють над тим, що називається «з'ясовним штучний інтелект», який прагне пояснити, як алгоритми нейронної мережі приходять до рішень та висновків, не потребуючи розриву та реверс-інжинірингу ШНМ, тож, можливо, ця проблема буде вирішена в майбутньому.

1.5 Протоколи обміну ключем

Обмін ключами за протоколом Діффі-Хеллмана.

Алгоритм Діффі-Хеллмана – це протокол обміну ключами, який дає змогу двом сторонам, які спілкуються через публічний канал, встановити взаємний секретний ключ без передачі його через Інтернет. Протоколом Діффі-Хеллман дозволяє двом використовувати відкритий ключ для шифрування та дешифрування їх розмови або даних за допомогою симетричної криптографії.

Діффі-Хеллмана зазвичай пояснюється двома зразковими сторонами, «Алісою» та «Бобом», які починають діалог. Кожен має певну інформацію, якою хоче поділитися, зберігаючи її таємницю. Для цього вони погоджуються на публічну частину доброякісної інформації, яка буде змішана з їхньою конфіденційною інформацією, коли вона передається через незахищений канал. Їхні секрети змішуються з публічною інформацією, або відкритим ключем, і в міру обміну секретами інформація, якою вони хочуть поділитися, змішується із загальною таємницею. Коли вони розшифровують повідомлення іншого, вони можуть витягти публічну інформацію і, знаючи власну таємницю, вивести нову інформацію, яка була перенесена. Хоча опис цього методу здається нескладним, коли для приватних і відкритих ключів використовуються довгі рядки чисел, дешифрування сторонньою стороною,

яка намагається підслухати, математично нездійсненна навіть при наявності значних ресурсів.

Діффі-Хеллмана є однією з перших практичних реалізацій криптографії з відкритим ключем. Його опублікували в 1976 році Вітфілд Діффі та Мартін Хеллман. Серед інших учасників, яким приписують розвиток Діффі-Хеллмана, належать Ральф Меркл та дослідники розвідувальних служб Сполученого Королівства (близько 1969 р.) [3].

Обмін ключами за протоколом Ель-Гамаля.

Протокол Ель-Гамаля за рахунок попереднього поширення відкритого ключа однієї зі сторін забезпечує аутентифікацію ключа для цієї сторони. Можна гарантувати, що тільки власник відповідного закритого ключа зможе обчислити сесійний ключ. Однак підтвердження факту отримання ключа не є частиною протоколу.

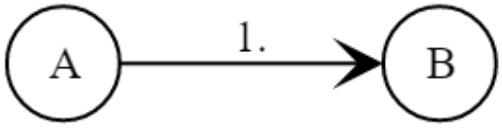


Рисунок 1.2 – Протокол обміну ключа Ель-Гамаля

1. Клієнти A і B вибирають загальні параметри p і g, де p – велике просте число, а g – примітивний елемент поля.

Клієнт B створює пару з закритого і відкритого ключів b і K_B:

$b: 2 \leq b \leq p-1, K_B = g^b mod\ p.$

Відкритий ключ K_B знаходиться в загальному відкритому доступі для всіх сторін. Криптоаналітика не може підмінити його – підміна буде помітна.

2. Клієнт A вибирає секрет x і обчислює сесійний ключ K:

$K = K_B^{bx} = g^{bx} mod\ p.$

$A \rightarrow \{g^x mod\ p\} \rightarrow B.$

3. Клієнт B розраховує сесійний ключ:

$K = (g^x)^b = g^{bx} mod\ p.$

Протокол не забезпечує гарантію вибору нового сесійного ключа в кожному сеансі протоколу, а використання «майстер»-ключа K_B для передачі сесійного ключа дозволяє зловмисникові обчислити всі сесійні ключі з минулих сеансів при компрометації закритого ключа b [4-5].

1.6 Процес шифрування на основі деревовидних машин парності

Безпека в Інтернеті стає все більш необхідною разом із поширенням Інтернет-з'єднань по всьому світу. Наявність комерційних комунікаційних систем обумовлює потенційний ризик їх неправомірного використання з боку супротивника. Існує кілька сценаріїв того, як комунікацію можна зловживати. Одним з найпоширеніших криптографічних методів, пов’язаних зі штучним інтелектом у нейронних мережах, є ДМП.

ДМП є найпоширенішою топологією нейронної мережі для нейронної криптографії. Він складається з $K \times N$ вхідних нейронів, K прихованих нейронів і одного вихідного нейрона o [6-8]. В основі нейронної криптографії використовуються дві однакові ШНМ, які здатні синхронізуватися після взаємного навчання. На початку кожен ДМП генерував випадкові значення ваг (w_{kj}), які є секретними. Процес навчання складається з генерування випадкових вхідних даних, однакових для обох ДМП, потім обчислення вихідних даних кожного ДМП та їх взаємного порівняння [6-9].

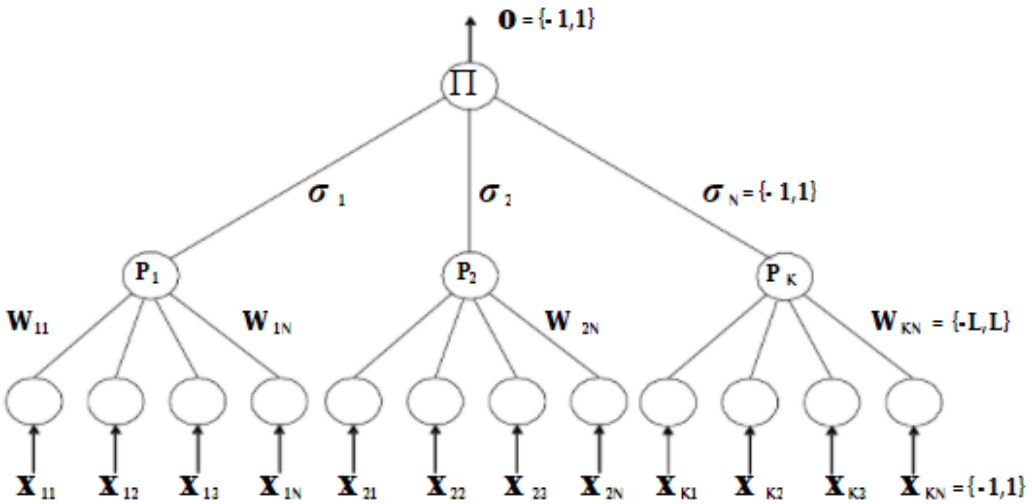


Рисунок 1.3 – Архітектура ДМП

ДМП (див. рис. 1.3) складається з вхідних бітів x, вагових векторів w (w – значення синаптичної ваги) та вихідного біта o. Крім того, ДМП складається з K прихованих нейронів і N входів для кожного прихованого нейрона. Значення синаптичних ваг w генеруються як випадкові дискретні значення L. Отже, w_{kj} може брати числа з $-L, -L+1, \dots, L$ [10]. Передбачається, що K, N, L і w є секретними для зловмисника, який намагається перехопити зашифроване повідомлення з загальнодоступного каналу. Наступне припущення полягає в тому, що K, N, L мають бути однаковими для обох ДМП, які беруть участь у процесі синхронізації.

Для обміну ключами між двома абонентами найбільш часто використовується алгоритм Діффі-Хеллмана. Його більш безпечна заміна заснована на синхронізації двох ДМП. Синхронізація цих машин схожа на синхронізацію двох хаотичних осциляторів в теорії хаотичних зв'язків [11-12].

ДМП як найпоширеніша топологія нейронної мережі в нейронній криптографії має серйозний недолік, а саме знаходження стану, що дві ДМП синхронізовані. Пряме порівняння ваг w по загальнодоступному каналу може призвести до розкриття класифікованих значень, таких як кількість нейронів K, N кількість вхідних даних, діапазон значень L, яких можна досягти за допомогою ваг і самих ваг. Вони використовуються як ключі для шифрування та дешифрування. Це причина, чому абсолютно необхідно знайти безпечний спосіб перевірити, чи синхронізовані дві машини паритету дерева чи ні.

Для підвищення безпеки синхронізації необхідно використовувати запити. Запит складається з: вхідні вектори, які корелюють із поточним вектором ваги w_{kj}. Випадкові вхідні дані замінюються запитами, які A та B обирають по черзі відповідно до власних векторів ваги. На непарних (парних) кроках часу партнер A (B) генерує вхідний вектор, який має певне перекриття його ваг wA (wB).

Запит базується на вхідному векторі, створеному на основі локального поля вагових векторів. Запити генеруються по черзі A і B і обмінюються. Запит замінює загальнодоступний вхід, згенерований за допомогою генератора

псевдовипадкових чисел, який використовується в базовому алгоритмі нейронної криптографії.

З включенням до запиту процес синхронізації тепер залежить не тільки від синаптичного відділу ДМП, а й від запитів. Процедура генерації запиту описується наступним чином. Оскільки обидві ваги w_{kj} і входи x_{kj} є дискретними, існує лише $2L + 1$ можливостей для $w_{kj} \cdot x_{kj}$. Тому ми можемо описати розв'язок, підрахувавши кількість c_{kl} продуктів із $w_{kj} \cdot x_{kj} = 1$. Тоді локальне поле задається як:

$$h_k = \frac{1}{\sqrt{N}} \sum_{i=1}^{L} (C_{k,l} - c_{k,-l})$$

Спочатку вихід σ_k прихованої одиниці вибирається випадковим чином. Тому задане значення локального поля задається як $h_k = \sigma_k H$.

$$c_{kl} = \left\lfloor \frac{n_{kl}+1}{2} + \frac{1}{2l}\left(\sigma_k H\sqrt{N} - \sum_{j=l+1}^{l} j(2c_{kj} - n_{kj})\right) \right\rfloor \qquad (1.1)$$

щоб обчислити значення $c_{kl}, c_{kl-1}, \ldots, c_{k1}$. У кожному розрахунку одне з двох рівнянь вибирається випадковим чином з однаковою ймовірністю, щоб помилки округлення не впливали на середній результат. Крім того, необхідно враховувати, що $0 \leq c_{kl} \leq n_{kl}$. Тому ми встановлюємо c_{kl} до найближчого граничного значення, якщо (1.1) дають результат за межами цього діапазону.

Після цього формується вхідний вектор x_k. Входи, пов'язані з нульовою вагою, вибираються випадковим чином, оскільки вони не впливають на локальне поле. Інші вхідні біти x_{kj} поділяються на L груп за абсолютним значенням $l = |w_{kj}|$ їх відповідної ваги. У кожній групі входи c_{kl} вибираються випадковим чином і встановлюються на xk,j = sgn(wkj). Решта $n_{kl} - c_{kl}$ вхідних бітів встановлені в $x_{kj} = -sgn(w_{kj})$.

Для того, щоб забезпечити безпечний обмін ключами із запитами, партнери повинні вибрати параметр H таким чином, щоб вони швидко синхронізувалися, а зловмисник не досягав успіху. На щастя, це можливо для всіх відомих атак. Потім знову знаходять ті самі закони масштабування, за яким параметром H можна досягти більш високого рівня безпеки для протоколу обміну нейронними ключами без збільшення середнього часу синхронізації.

Єдина зміна в базовому алгоритмі обміну нейронними ключами полягає в тому, що публічні випадкові введення замінюються на запит, який базується на їх векторі ваги. Алгоритм наведено нижче:

1. Кожна сторона вибирає випадковий початковий вектор ваги W_k^A і W_k^B в момент часу $t = 0$.

2. На кожному кроці навчання вихід кожної прихованої одиниці обчислюється як:

$$\sigma_i^A = sign(w_i^A, x), \sigma_i^B = sign(w_i^B, x)$$

Вихідні біти прихованих одиниць об'єднані у вихідний біт o для обох мереж A та B [13].

$$O = \prod_{i=1}^{K} \sigma_i \quad (1.2)$$

3. Якщо вихідні біти відрізняються, $O^A \neq O^B$, нічого не змінюється.

4. Якщо $O^A = O^B$ навчаються лише приховані одиниці, які мають вихідний біт, ідентичний загальному виходу.

$$O^{A|B} = \sigma^{A|B}$$

5. Під час тренування ваги коригуються з використанням правила Хебба, як наведено нижче [14-15].

$$w_i^A(t+1) = w_i^A(t) + x_i(t) \quad (1.3)$$

Якщо будь-який компонент w_k виходить за межі інтервалу $[-L, L]$, він замінюється знаком $(w_k)L$. Використовуючи цей алгоритм, дві нейронні мережі синхронізуються із загальним секретним ключем, що залежить від часу $w^A(t) = w^B(t)$.

У вихідному протоколі обміну ключами структура мережі, вихід ДМП (1.2), правило адаптації (1.3) і загальні входи x_{kj}є відкритими. Єдині секрети, які стосуються, це різні початкові ваги w_{jk} двох сторін. Якби вони не були секретними, отримані ключі можна було б просто обчислити, оскільки всі подальші обчислення повністю детерміновані.

З використанням запитів рішення основного алгоритму поділяється на два етапи.

1 етап:

– у першому алгоритм обміну нейронними ключами із запитом застосовується до повної синхронізації двох мереж;

– вхідні дані (запит) видимі зловмиснику, але він не може передбачити, який буде запит ДМП (вхід), згенерований будь-якою стороною, оскільки він базується на вагових показниках, які ніколи не розкриваються.

2 етап:

– після синхронізації ідентичні вагові вектори використовуються як початковий для генератора псевдовипадкових чисел;

– запити не обмінюються, скоріше вхідні дані до нейронної мережі (ДМП) отримуються з псевдовипадкового генератора;

– ДМП тепер можна використовувати для створення секретного ключа, який використовується для шифрування/дешифрування звичайного тексту за допомогою розширеного стандарту шифрування.

Посів псевдовипадкового генератора не обмінюється по мережі, зловмисник не може передбачити вихід випадкових генераторів, хоча його алгоритм є відкритим. Отже, це ще один параметр невідомий для зловмисника.

1.7 Постановка завдання та аналіз вимог до розробки системи

Результатом виконання кваліфікаційної роботи магістра можна вважати розроблену систему для шифрованого обміну повідомленнями, яка може бути використана працівниками компанії, яка потребує такого рівня безпеки спілкування.

До вимог можна віднести наступне: надійність – а саме безвідмовність (виключення можливості відмови компонентів системи); можливість роботи максимально довгий період часу, виконуючи планові технічні роботи; придатність до усунення технічних проблем, виниклих під час користування системою.

Розроблювана система повинна виконувати наступні функції:

- надавати можливість користувачеві підключатися до серверу;
- надавати можливість шифрувати відправленні повідомлення;
- надавати можливість розшифровувати прийняті повідомлення;
- надати користувачам необхідний рівень захисту інформації.

Особливих вимог до користувацького інтерфейсу немає. Так як дана система планується до використання на підприємстві, де користувачам не важлива зовнішня складова системи, та надається перевага до надійності системи.

Система шифрування повідомлень повинна функціонувати на персональних комп’ютерах під керуванням операційної системи Windows 10. Мінімальними системними вимогами можна визначити рівними мінімальним системним вимогам операційної системи.

Також до системних вимог слід віднести необхідність встановлення пакету Python, що буде виконане системним адміністратором підприємства.

1.8 Вибір мови програмування для вирішення поставленої задачі

Написання коду для реалізації даної системи буде поділено на два етапи: етап реалізації клієнт-серверної частини та створення веб-додатку.

Так як для вирішення поставленої задачі треба використовувати ШНМ, то вибір мови програмування зупиниться на Python.

Він має величезну кількість бібліотек і фреймворків: мова Python постачається з багатьма бібліотеками та фреймворками, які полегшують кодування. Це також значно економить час.

Код Python є стислим і читабельним навіть для нових розробників, що корисно для проектів машинного та глибокого навчання. Завдяки простому синтаксису розробка додатків за допомогою Python швидка в порівнянні з багатьма мовами програмування. Крім того, це дозволяє розробнику тестувати алгоритми, не реалізуючи їх.

Python є мовою програмування з відкритим вихідним кодом і користується чудовою підтримкою з багатьох ресурсів і якісної документації по всьому світу. Він також має велику та активну спільноту розробників, які надають свою допомогу на будь-якому етапі розвитку.

Більшість вчених прийняли Python для проектів машинного навчання та глибокого навчання, що означає, що більшість найяскравіших розумів у всьому світі можна знайти в спільнотах Python.

У Python є легкий для розуміння та дружній синтаксис. Крім того, численні фреймворки та бібліотеки сприяють розробці програмного забезпечення. Використовуючи готові рішення, можна багато чого зробити за допомогою кількох рядків коду. Python добре підходить для розробки прототипів, що підвищує продуктивність [16-18].

Щодо розробки веб-додатку, то було застосовані HTML (HyperText Markup Language – «мова гіпертекстової розмітки») та JavaScript.

Використовуючи HTML буде розроблено графічний інтерфейс додатку, який надає можливість надсилати шифровані повідомлення.

HTML – це мова комп’ютера, з якої складається більшість веб-сторінок та онлайн-додатків. Гіпертекст – це текст, який використовується для посилань на інші фрагменти тексту, тоді як мова розмітки – це серія маркування, яка повідомляє веб-серверам стиль і структуру документа.

HTML не вважається мовою програмування, оскільки він не може створювати динамічні функції. Натомість за допомогою HTML веб-користувачі можуть створювати та структурувати розділи, абзаци та посилання за допомогою елементів, тегів та атрибутів.

Далі представлені деякі з найпоширеніших варіантів використання HTML:

– веб-розробка – розробники використовують HTML-код, щоб розробити, як браузер відображає елементи веб-сторінки, такі як текст, гіперпосилання та медіафайли;

– інтернет навігація – користувачі можуть легко переміщатися та вставляти посилання між пов’язаними сторінками та веб-сайтами, оскільки HTML активно використовується для вбудовування гіперпосилань;

– веб-документація – HTML дозволяє впорядковувати та форматувати документи, подібно до Microsoft Word.

Також варто зазначити, що HTML тепер вважається офіційним веб-стандартом. Консорціум World Wide Web Consortium (W3C) підтримує та розробляє специфікації HTML, а також надає регулярні оновлення [19-20].

За допомогою JavaScript були реалізовані функції шифрування та розшифрування повідомлень, та функції підключення до серверу.

JavaScript – це текстова мова програмування, яка використовується як на стороні клієнта, так і на стороні сервера, що дозволяє робити веб-сторінки інтерактивними. Якщо HTML є мовою, яка надає структуру та стиль веб-сторінкам, то JavaScript надає веб-сторінкам інтерактивні елементи, які залучають користувача.

Підключення JavaScript покращує роботу веб-сторінки, перетворюючи її зі статичної сторінки в інтерактивну. JavaScript додає поведінку веб-сторінкам [21].

1.9 Висновки за розділом

До підсумку з виконання першого розділу кваліфікаційної роботи можна зазначити про проведення аналітичного огляду на вирішення проблеми обраної тематики. Також було розкрито поняття ШНМ та варіації їх застосування. Далі було описано в який спосіб можна застосовувати нейронні мережі для забезпечення безпеки персональних даних. Та розкриті недоліки такого застосування нейронних мереж для забезпечення безпеки даних. Описані декілька існуючих протоколів обміну ключем шифрування та вибір оптимального протоколу для подальшої розробки та дослідження, а саме протокол на основі синхронізації двох ДМП.

РОЗДІЛ 2
ПРОЕКТУВАННЯ І РОЗРОБКА СИСТЕМИ ШИФРОВАНОГО ОБМІНУ ПОВІДОМЛЕННЯМИ

Наступним, після аналітичного огляду проблеми, етапом розробки є етап проектування майбутньої системи.

Та вже з готовими макетами приступити до наступного етапу – етапу програмної реалізації спроектованої системи.

Останнім етапом слід визначити етап тестування розробленої системи.

2.1 Процес проектування графічного інтерфейсу користувача

Графічний інтерфейс користувача – це система інтерактивних візуальних компонентів для комп’ютерного програмного забезпечення. Графічний інтерфейс відображає об’єкти, які передають інформацію та представляють дії, які може виконувати користувач. Об’єкти змінюють колір, розмір або видимість, коли користувач взаємодіє з ними.

Графічний інтерфейс містить об’єкти графічного інтерфейсу, такі як значки, курсори та кнопки. Ці графічні елементи іноді доповнюються звуками або візуальними ефектами, такими як прозорість і тіні. Використовуючи ці об’єкти, користувач може використовувати комп’ютер, не знаючи команд.

Графічний інтерфейс використовує вікна, значки та меню для виконання команд, таких як відкриття, видалення та переміщення файлів. Хоча навігація в операційній системі з графічним інтерфейсом здійснюється за допомогою миші, клавіатуру також можна використовувати за допомогою комбінацій клавіш або клавіш зі стрілками [22].

Інтерфейс серверної частини не вимагає графічного інтерфейсу, тому що він запускається та підтримується в робочому стані адміністратором системи у вигляді консольного додатку.

Також додаток для синхронізації двох ШНМ для формування ключа зв’язку не вимагає графічного інтерфейсу, тому що ключ генерується на персональній станції користувача адміністратором через консольний додаток.

Отже, до проектування графічного інтерфейсу користувача віднесемо проектування макету web-додатка. Так як система для обміну повідомленнями буде реалізована у вигляді чату у вікні браузера.

Це означає, що користувацький інтерфейс системи для обміну повідомленнями буде розроблений з використанням HTML.

Так як цільовою категорію користувачів будуть складати працівники компанії без просунутих навичок користування персональним комп’ютером, інтерфейс має бути простим у розумінні.

Після успішного підключення до серверу та подальшої синхронізації двох ШНМ для генерування ключа зв’язку, користувачеві відкривається вікно браузеру зі змістом представленим на рис. 2.1.

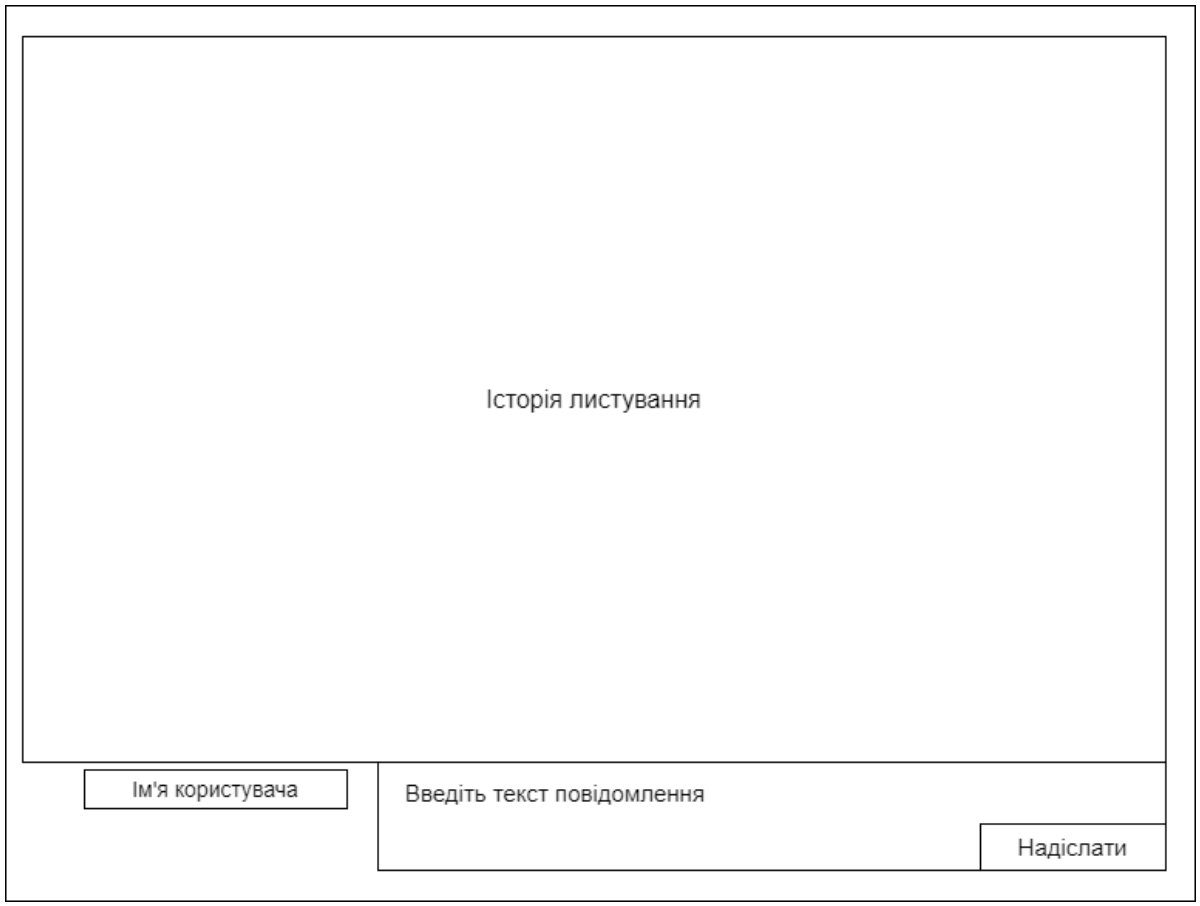


Рисунок 2.1 – Макет графічного інтерфейсу користувача

Використовуючи даний інтерфейс користувач отримує змогу, вказавши своє ім’я до поля «Ім’я користувача», надіслати зашифроване повідомлення з поля «Введіть текст повідомлення» іншим користувачам, шляхом натискання на кнопку «Надіслати», які в свою чергу розшифровують повідомлення і відображаються в полі «Історія листування».

2.2 Концептуальна модель системи

Концептуальна модель системи представляється за допомогою діаграми прецедентів, використовуючи UML (Уніфікованої мови моделювання).

В UML діаграми прецедентів моделюють поведінку системи та допомагають охопити вимоги системи.

Діаграми прецедентів описують функції високого рівня та область застосування системи. Ці діаграми також визначають взаємодії між системою та її акторами. Варіанти використання та дійові особи на діаграмах варіантів використання описують, що робить система і як учасники її використовують, але не те, як система працює всередині.

Діаграми прецедентів ілюструють і визначають контекст і вимоги або всієї системи, або важливих частин системи. Можна змоделювати складну систему за допомогою однієї діаграми прецедентів або створити багато діаграм прецедентів для моделювання компонентів системи. Зазвичай розробляються діаграми прецедентів на ранніх етапах проекту і звертаються до них протягом усього процесу розробки.

Діаграми варіантів використання корисні в наведених нижче ситуаціях:

– перед початком проекту потрібно створити діаграми прецедентів для моделювання бізнесу, щоб усі учасники проекту мали розуміння щодо працівників, клієнтів та діяльності підприємства;

– збираючи вимоги, можна створювати діаграми прецедентів, щоб охопити системні вимоги та представити іншим, що має робити система;

– на етапах аналізу та проектування можна використовувати варіанти використання та акторів із ваших діаграм прецедентів, щоб визначити класи, які потрібні системі;

– на етапі тестування можна використовувати діаграми варіантів використання для визначення тестів для системи [23].

Використовуючи спроектовану діаграму прецедентів (див. рис. 2.2) можна визначити певну послідовність дій, результатом виконання яких буде виконання визначених функцій.

Актор в особі користувача системи отримує можливість генерації секретного ключа зв’язку, можливість підключення до співрозмовника та обмінюватись повідомленнями в шифрованому вигляді.

Актор в особі адміністратору системи має можливість оновлення ПЗ, налагодження каналів зв’язку та формування звітності.

Рисунок 2.2 – Діаграма прецедентів проектованої системи

2.3 Поведінкова модель системи

Очікувану поведінку проектованої системи можна представити за допомогою діаграми станів.

Діаграма стану – це графічне представлення стану системи. Діаграми станів показують поведінкову модель, що складається із станів, переходів станів і дій.

Діаграми станів UML засновані на концепції діаграм станів Девіда Харела. Діаграми станів зображують дозволені стани та переходи, а також події, які впливають на ці переходи.

Діаграми станів зазвичай використовуються в області вбудованих систем. Діаграми станів допомагають візуалізувати весь життєвий цикл об'єктів і, таким чином, допомагають краще зрозуміти системи, засновані на стані [24].

Діаграма стану представлена на рис. 2.3.

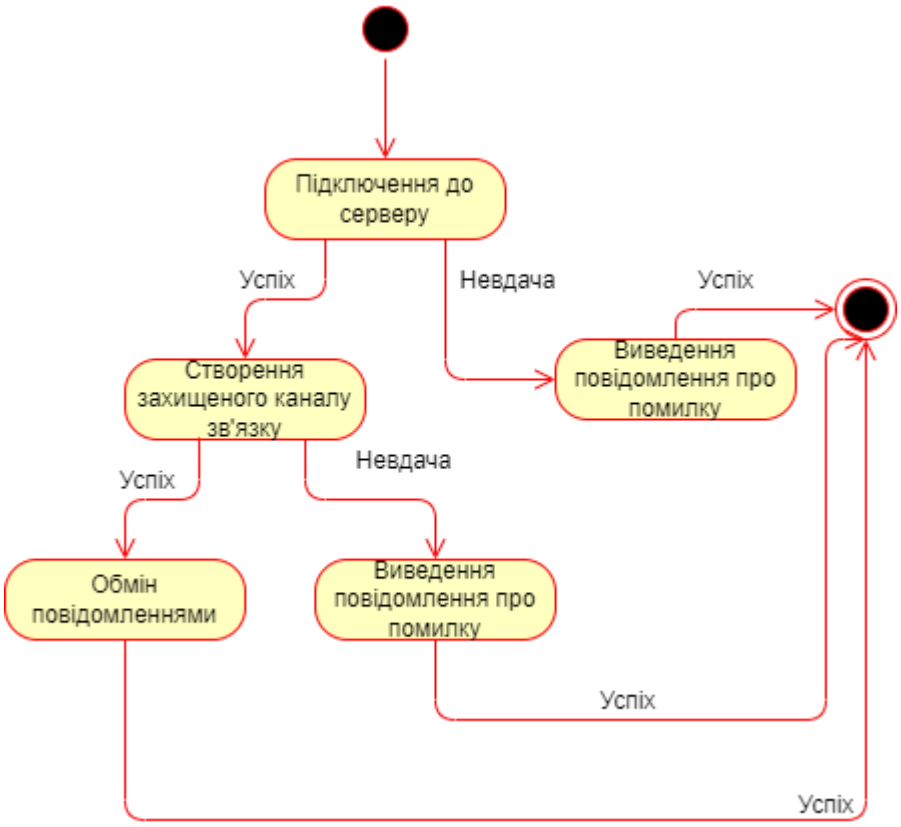


Рисунок 2.3 – Діаграма стану

Спочатку виконується спроба підключення до серверу, якщо підключення було виконане вдало, то запуск додатку є вдалим, інакше – виведення повідомлення про помилку.

Далі виконується спроба створення захищеного каналу зв’язку, шляхом генерування секретного ключа, при вдалій синхронізації надається можливість на обмін шифрованими повідомленнями, інакше – виведення повідомлення про помилку.

2.4 Архітектурна модель системи

В результаті визначення архітектури проектованої системи на ранніх етапах розробки дозволить полегшити подальшу програмну реалізацію.

Діаграма компонентів описує організацію та підключення фізичних компонентів у системі.

Діаграми компонентів часто малюються, щоб допомогти з деталями реалізації моделі та перевірити, чи кожен аспект необхідних функцій системи охоплений запланованою розробкою.

У першій версії UML компоненти, включені в ці діаграми, були фізичними: документи, таблиця бази даних, файли та виконувані файли, усі фізичні елементи з місцем розташування.

У UML 2 ці компоненти є менш фізичними і більш концептуальними окремими елементами дизайну, такими як бізнес-процес, який забезпечує або вимагає інтерфейсів для взаємодії з іншими конструкціями в системі. Фізичні елементи, описані в UML 1, такі як файли та документи, тепер називаються артефактами.

Компонент UML 2 може містити кілька фізичних артефактів, якщо вони разом [25].

Діаграма компонентів представлена на рис. 2.4.

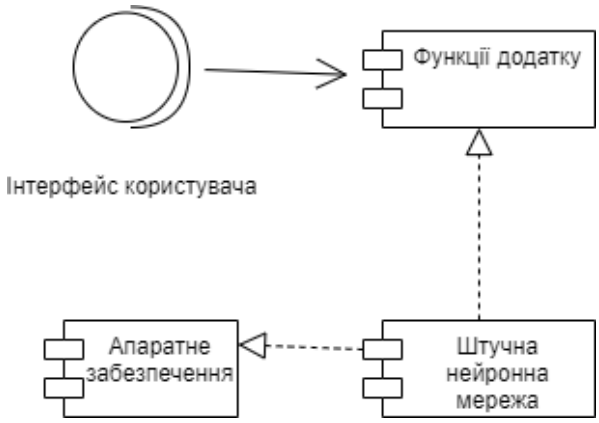


Рисунок 2.4 – Діаграма компонентів системи

Інтерфейс користувача потрібен для взаємодії між користувачем та ПЗ.

До функцій додатку відносяться процес шифрування, розшифрування та надсилання і отримання повідомлень.

Апаратне забезпечення складається з комп’ютерних комплектуючих системи користувача.

ШНМ являє собою головну функцію системи, а саме генерація ключа шифрування.

2.5 Представлення алгоритму синхронізації на основі використання ДМП

Протокол синхронізації ДМП є більш безпечним аналогом алгоритму Діффі-Хеллмана [26]. Протокол являє собою специфічний алгоритм обміну криптографічними ключами. Алгоритм обміну ключами дозволяє двом користувачам обмінюватися загальним секретним ключем через незахищений канал зв'язку без будь-яких попередніх секретів між собою.

Що стосується шифрування секретних повідомлень, то цей спільний ключ можна використовувати для шифрування наступних комунікацій за допомогою шифру симетричного ключа. У цьому протоколі два користувача (тобто Аліса і Боб) хочуть спілкуватися один з одним, але не хочуть, щоб підслуховувач (тобто Єва) знав їхнє повідомлення. Для цього вони домовляться та встановлять випадковий секретний ключ для своєї системи

приватних ключів, використовуючи відкритий, але автентифікований канал. Алгоритм обміну ключами виглядає наступним чином:

1. Аліса і Боб узгоджують і оприлюднюють два числа g і p, де g називають генератором, а p – простим числом. Слід відзначити, що будь-хто має публічний доступ до цих номерів.

2. Аліса вибирає випадкове число a і обчислює u за модулем p і надсилає його Бобу.

$$u = g^a mod\ p$$

3. Боб вибирає випадкове число b і обчислює v за модулем p і надсилає його Алісі.

$$v = g^b mod\ p$$

4. Аліса обчислює секретний ключ s_a за тим самим модулем p.

$$s_a = v^a mod\ p = (g^b)^a mod\ p$$

5. Боб обчислює секретний ключ s_b за тим же модулем p.

$$s_b = u^b mod\ p = (g^a)^b mod\ p$$

6. Тепер і Аліса, і Боб мають однаковий загальний секретний ключ, а саме s.

$$s = s_a = s_b = g^{ab} mod\ p$$

7. І Аліса, і Боб зберігають цей спільний секретний ключ як приватний ключ, і він використовуватиметься для шифрування повідомлень один одному.

На рис. 2.5 показаний алгоритм обміну ключами. Можна побачити, що Аліса генерує випадкове число a і обчислює u за модулем арифметики з генератором g і простим числом p.

Аліса надсилає обчислені значення u Бобу як відкритий ключ. З отриманого відкритого ключа Аліси Боб обчислює секретний ключ s_b за тією ж арифметикою за модулем p. Аналогічно, Аліса обчислює секретний ключ s_a за відкритим ключем Боба, і цей секретний ключ такий самий, як і секретний ключ Боба для спільного доступу.

Якщо зловмисник Єва хоче обчислити секретний ключ s, їй знадобиться або випадкове число a, або випадкове число b. Незважаючи на те, що Єва помічає множину $\{g, u, v\}$, яка зараз є загальнодоступною інформацією, нелегко отримати a чи b із множини $\{g, u, v\}$, оскільки їй потрібно вирішити проблему дискретного логарифма.

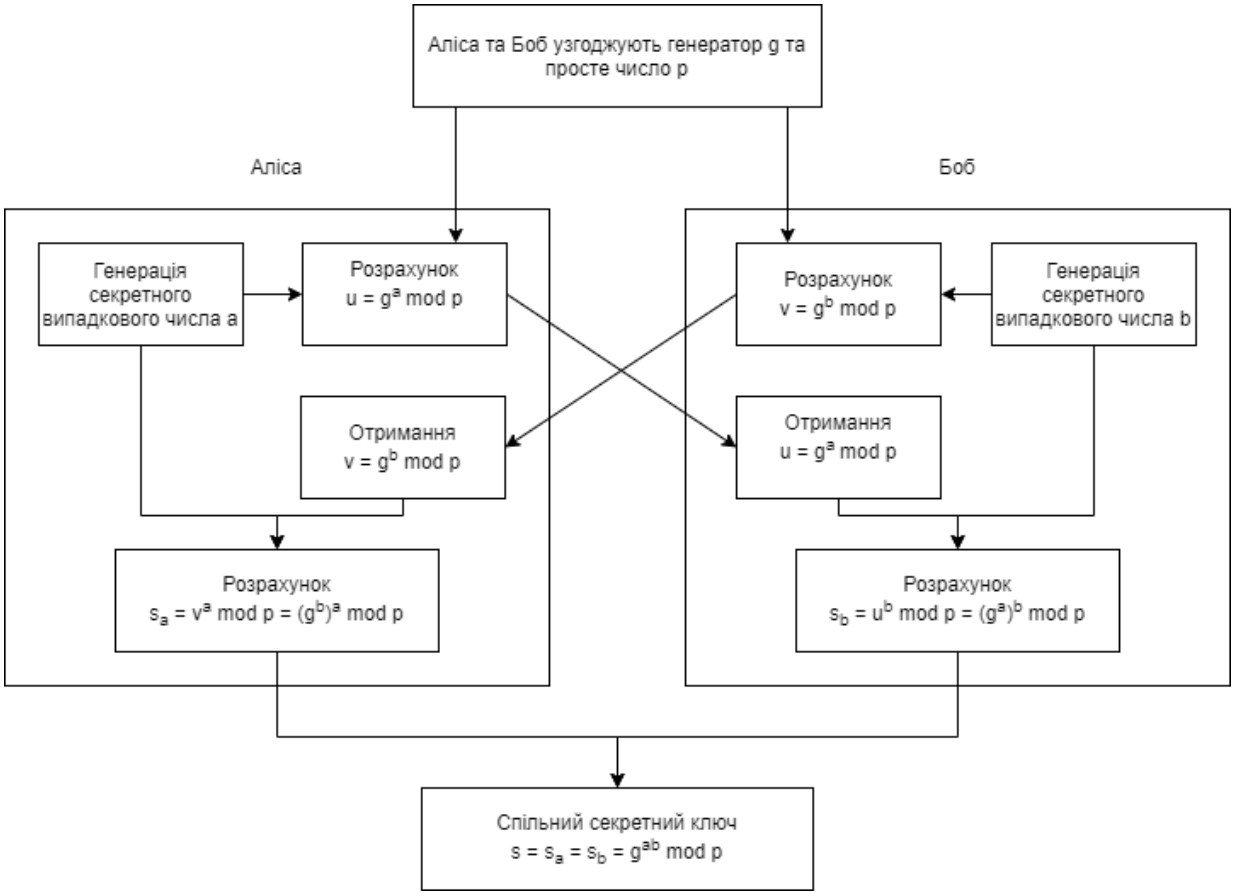


Рисунок 2.5 – Алгоритм обміном ключем шифрування

Немає відомого алгоритму для вирішення цієї проблеми за розумний проміжок часу. Якщо просте число p досить велике, для вирішення цієї проблеми дискретного логарифма потрібно досить багато часу, а обчислення рішення за допомогою грубої атаки неефективно та практично.

Наступним кроком є застосування алгоритму генерації секретного ключа шифрування для процесу шифрованого обміну повідомленнями.

На рис. 2.6 наведений алгоритм проектованої системи.

Рисунок 2.6 – Блок-схема синхронізації двох ШНМ

2.6 Перелік використаних бібліотек

2.6.1 Перелік бібліотек, використаних для розробки клієнт-серверного додатку

Під час програмної реалізації спроектованої системи для шифрованого обміну повідомленнями були застосовані бібліотеки мови Python, перелік яких наведено в табл. 2.1.

Таблиця 2.1 – Перелік застосованих бібліотек

Назва бібліотеки	Опис застосування
Flask	невеликий та легкий веб-фреймворк для створення веб-застосунків за допомогою Python
Flask_Socketio	надає програмам Flask доступ до двонаправленого зв'язку з низькою затримкою між клієнтами та сервером
Pickle	реалізує потужний алгоритм серіалізації та десеріалізації об'єктів Python
Sys	забезпечує доступ до деяких змінних та функцій, що взаємодіють з інтерпретатором Python
SocketIO	реалізуються клієнти та сервери, які можуть працювати окремо або інтегровано з різноманітними веб-фреймворками Python
NumPy	додає підтримку великих багатовимірних масивів і матриць
Math	модуль надає великий функціонал до роботи з числами
Random	надає функції для створення випадкових чисел, літер, випадкового вибору елементів послідовності.
Webbrowser	забезпечує інтерфейс високого рівня, що дозволяє користувачам переглядати веб-сторінки
Cryptography	надає користувачам різні способи криптографії; одна з них – просте шифрування та дешифрування даних

Бібліотека Flask використовується для створення веб-додатків використовуючи мову програмування Python [27].

В процесі програмної реалізації системи були застосовані модулі, перелік яких наведено в табл. 2.2.

Таблиця 2.2 – Перелік застосованих модулів бібліотеки Flask

Модуль	Опис застосування
Flask	надає можливість використовувати його в будь-яких функціях уявлень
Request	в модулі зберігаються всі вхідні дані запиту, включаючи тип MIME, джерело, IP-адресу, необроблені дані, метод HTTP, заголовки тощо
Render_template	ця функція приймає ім'я шаблону та список змінних аргументів шаблону, а повертає готовий шаблон із заміненими аргументами

Бібліотека Flask_Socketio застосовується для створення каналу зв’язку між клієнтом та сервером [28].

В процесі програмної реалізації системи були застосовані модулі, перелік яких наведено в табл. 2.3.

Таблиця 2.3 – Перелік застосованих модулів бібліотеки Flask_Socketio

Модуль	Опис застосування
SocketIO	використаний для відкриття каналу зв’язку
join_room	використаний для приєднання до серверу
leave_room	використаний для від’єднання до серверу
send	використаний для відправки повідомлень
emit	використаний для отримання повідомлень

Бібліотека Pickle реалізує процес перетворення об'єкта Python на потік байтів. Так як потік байтів легко можна записати у файл, модуль pickle широко застосовується для збереження та завантаження складних об'єктів у Python [29].

Бібліотека sys дозволяє програмісту отримувати інформацію про інтерпретатора Python і операційну систему, працювати з введенням і висновком, змінювати параметри модуля і обробляти помилки [30].

Бібліотека SocketIO – це транспортний протокол, який забезпечує двоспрямований зв’язок на основі подій у реальному часі між клієнтами (як правило, але не завжди веб-браузерами) і сервером [31].

Бібліотека NumPy мови Python, що додає підтримку великих багатовимірних масивів і матриць, разом із великою бібліотекою високорівневих (і дуже швидких) математичних функцій для операцій із цими масивами [32].

Бібліотека Math у Python забезпечує доступ до деяких популярних математичних функцій і постійних, які можна використовувати в кодах для більш складних математичних розрахунків [33].

Бібліотека random реалізує генератор псевдовипадкових чисел для різних розподілів, включаючи цілі та матеріальні числа з плаваючою зап’ятою. Модуль randrange повертає випадково вибране число із послідовності [34].

2.6.2 Перелік бібліотек, використаних для розробки веб-додатку

Браузерна частина системи буде реалізована за допомогою HTML, з використанням бібліотек JavaScript.

Перелік застосованих бібліотек для реалізації браузерної частини системи наведений в табл. 2.4.

jQuery – це швидка, невелика і багатофункціональна бібліотека JavaScript. При її застосуванні значно спрощуються такі речі, як обхід і маніпуляції з документами HTML, обробка подій, анімація та Ajax, завдяки простому у використанні API, який працює в багатьох браузерах.

Таблиця 2.4 – Перелік застосованих бібліотек для розробки веб-додатку

Назва бібліотеки	Застосування
jQuery	необхідний для плагінів JavaScript
Socket.io	необхідний для роботи клієнт-серверного додатку
CryptoJS	необхідна для шифрування та розшифрування повідомлень

Завдяки поєднанню універсальності та розширюваності jQuery змінив спосіб написання JavaScript мільйонами людей [35].

Socket.io забезпечує зв'язок на основі подій у реальному часі між одним або кількома клієнтами та сервером. Він працює на будь-якій платформі, браузері чи пристрої та є швидким та надійним. Socket.io складається з двох частин: бібліотека на стороні клієнта, яка працює у браузері, і бібліотека на стороні сервера для Node.js. Обидва компоненти мають ідентичний API [36].

CryptoJS – це зростаюча колекція стандартних і безпечних криптографічних алгоритмів, реалізованих у JavaScript з використанням найкращих практик і шаблонів. Вони швидкі та мають послідовний і простий інтерфейс [37].

2.7 Програмна реалізація спроектованого програмного продукту

Процес реалізації спроектованої системи можна поділити на два етапи: етап реалізації серверної частини та реалізації алгоритму синхронізації двох ШНМ для отримання ключу шифрування; етап реалізації веб-додатку.

2.7.1 Опис програмної реалізації серверної частини та реалізація алгоритму синхронізації двох ШНМ

Для написання програмного коду клієнт-серверної частини була використана мова програмування Python. Програмна реалізація серверної та клієнтської частини можна поділити на декілька етапів.

До першого етапу слід віднести реалізацію серверної частини. Спочатку налаштовується можливість до підключення клієнтами до серверу, шляхом відкриття портів для з’єднання. Далі було реалізований функції, які дозволяють отримувати та відправляти повідомлення між сервером та клієнтом.

Наступним кроком стала розробка функціоналу, який отримує ідентифікатор користувача, який намагається підключитися до серверу, що дозволяє виключити спробу підключення третіх осіб. Також була реалізована функція, яка отримає вагові коефіцієнти користувачів, для того, щоб можна було відстежувати процес синхронізації двох ШНМ. Та однією з найважливіших функцій серверної частини є підтвердження синхронізації ШНМ клієнтів, а саме підтвердження однакового значення вихідних нейронів у кожного клієнта.

Другим етапом розробки стала реалізація клієнтської частини. Ця частина відповідає за реалізацію алгоритму синхронізації двох ДМП, а саме – синхронізацію ШНМ двох користувачів та отримання ключа шифрування. Першим кроком стала реалізація функціоналу, який надає можливість підключатися до серверу.

Далі надсилається ідентифікатор користувача на сервер та отримується ідентифікатор співрозмовника. Після цього було реалізований функціонал, який відповідає за синхронізацію ШНМ користувача зі ШНМ співрозмовника, іншими словами реалізація алгоритму синхронізації ДМП. Після успішного процесу синхронізації двох ШНМ користувачів на кожній станції генерується ключ шифрування, та зберігається локально, що надає додаткової безпеки від можливого отримання ключа третіми особами. Виконавши отримання ключа шифрування, користувачеві відкривається веб-додаток, за допомогою якого він отримує можливість шифрованим способом обмінюватися повідомленнями зі співрозмовником.

2.7.2 Опис програмної реалізації веб-додатку

Процес реалізації веб-додатку можна поділити на два етапи: налаштування користувацького інтерфейсу, використовуючи HTML, та реалізація функцій роботи системи шифрування.

До першого етапу можна віднести процес налаштування інтерфейсу користувача. HTML дозволяє налаштовувати елементи інтерфейсу на розсуд розробника. Були визначені розташування історії обміну повідомленнями, розташування елементу, який відповідає за введення користувачем свого повідомлення, розташування елементу, який виконує функцію відправки повідомлення на сервер та розташування елементу, який отримує секретний ключ шифрування для подальшої шифровки та дешифровки повідомлень.

Другим етапом є реалізація функцій за допомогою JavaScript. До цих функцій відноситься: процес підключення до серверу; функція надсилання повідомлень на сервер, які в свою чергу надсилаються з серверу іншому користувачеві; функцію шифрування та дешифрування повідомлень; функцію завантаження в веб-додаток секретного ключа шифрування.

2.8 Процес тестування розробленої системи

В даному розділі буде описаний загальний процес роботи розробленої системи та виконане тестування готового програмного продукту різними методами.

2.8.1 Процес налаштування системи

Процес налаштування повинен виконуватися працівником з просунутими навичками володіння персональним комп’ютером, рекомендується виконувати ці налаштування системним адміністратором підприємства.

Перш за все необхідно встановити пакет Python на робочі станції користувачів і робочу станцію, яка буде виконувати зв’язуючу роль – роль сервера, шляхом завантаження інсталятору з офіційного сайту Python.

Наступним кроком є встановлення використаних під час розробки системи бібліотек. Це можна зробити, використовуючи функції мови Python, перелічивши ці бібліотеки у текстовому файлі та виконавши команду, яка представлена на рис. 2.7.

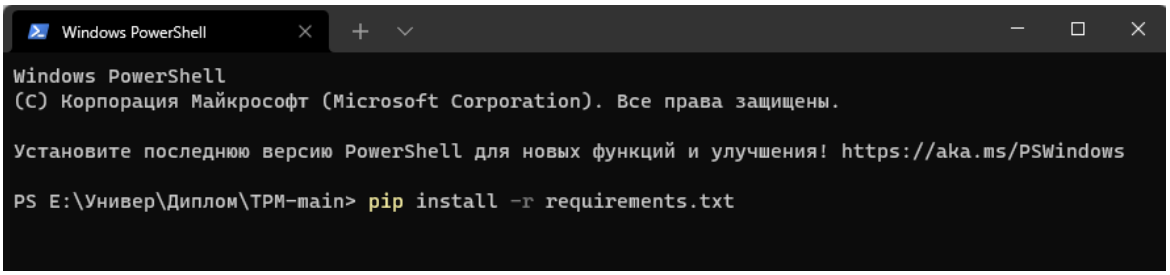


Рисунок 2.7 – Приклад встановлення використаних бібліотек

Приклад заповнення текстового файлу зі змістом бібліотек наведений на рис. 2.8.

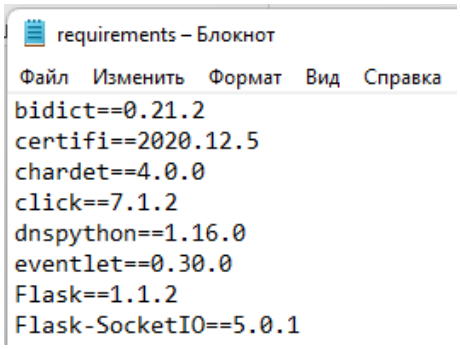


Рисунок 2.8 – Приклад заповнення файлу з переліком бібліотек

Наступним кроком системного адміністратора підприємства є налаштування серверу, виконавши команду, представлену на рис. 2.9.

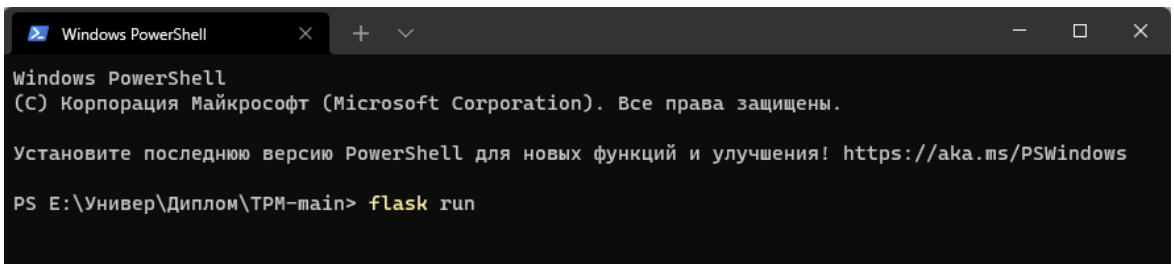


Рисунок 2.9 – Команда для налаштування серверної частини

Результат роботи команди для налаштування серверу представлений на рис. 2.10.

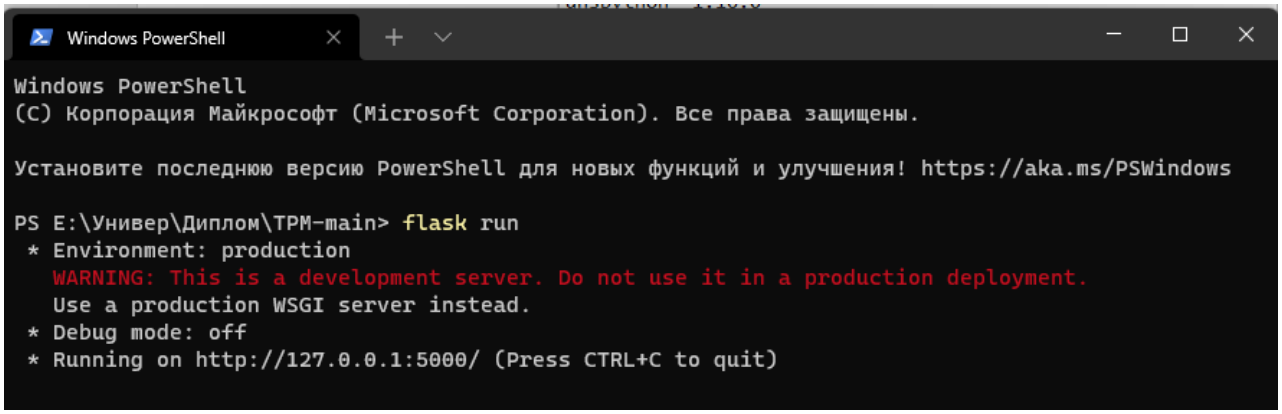


Рисунок 2.10 – Результат виконання команди для налаштування серверу

В результаті отримаємо робочу серверну частину, до якої можуть бути підключені користувачі.

2.8.2 Загальний опис роботи розробленої системи

Виконавши етап налаштування серверної частини, який представлений на рис. 2.9-2.10, треба виконати етап, який являє собою процес отримання секретного ключа шифрування для співрозмовників.

Спочатку одним із користувачів необхідно запустити додаток, шляхом виконання команди, представленої на рис. 2.11.

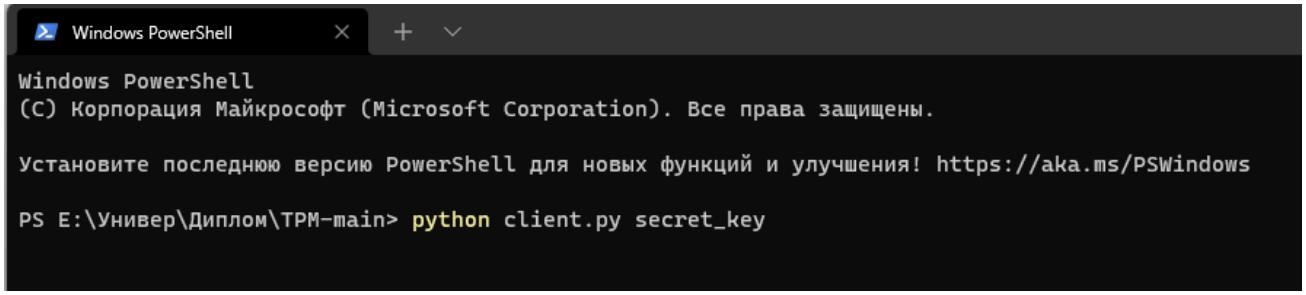


Рисунок 2.11 – Запуск додатку для генерації ключа шифрування

В результаті користувачеві буде відображення повідомлення про те, що підключення до серверу вдале. Демонстрація підключення наведена на рис. 2.12.

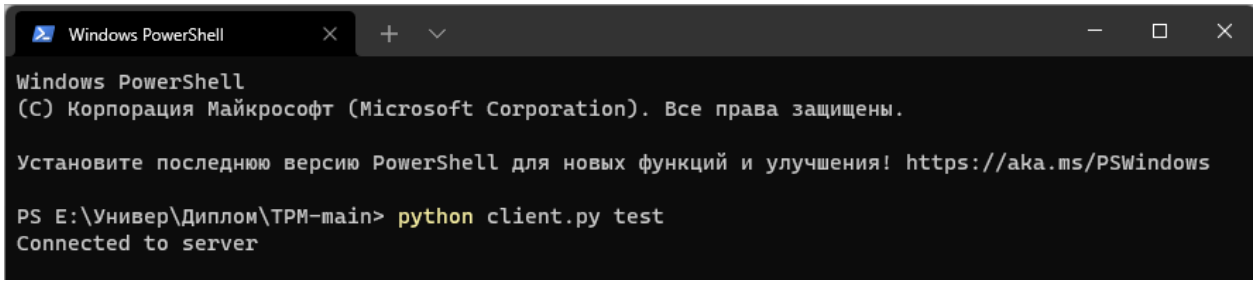


Рисунок 2.12 – Результат вдалого підключення до серверу

Системний адміністратор з боку серверу може отримати повідомлення про спробу та результат підключення клієнту до серверу, ідентифікатор підключеного користувача. Приклад повідомлення можна побачити на рис. 2.13.

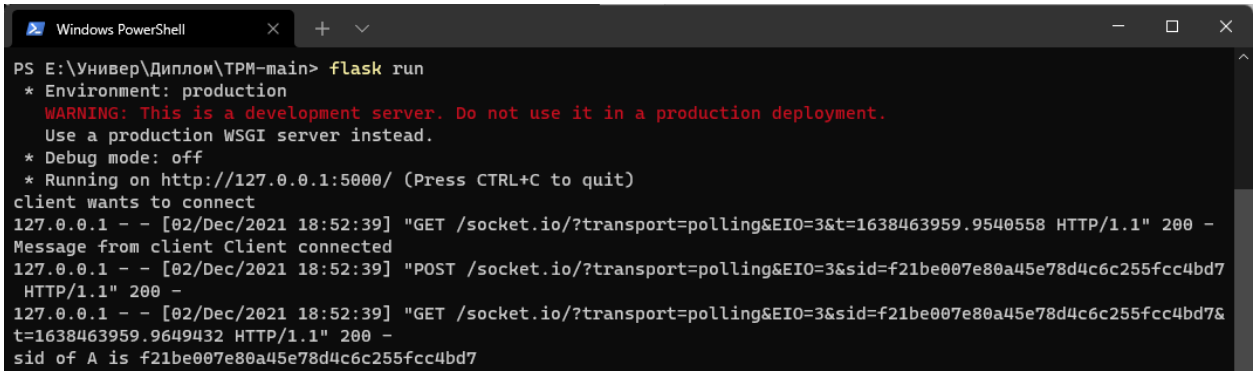


Рисунок 2.13 – Підтвердження підключення клієнта до серверу

Наступним кроком є підключення іншого користувача до серверу. Процес аналогічний представленому на рис. 2.11-2.12.

Після вдалого підключення другого користувача починається процес синхронізації двох ШНМ. Процес синхронізації від зору користувачів представлений на рис. 2.14-2.15.

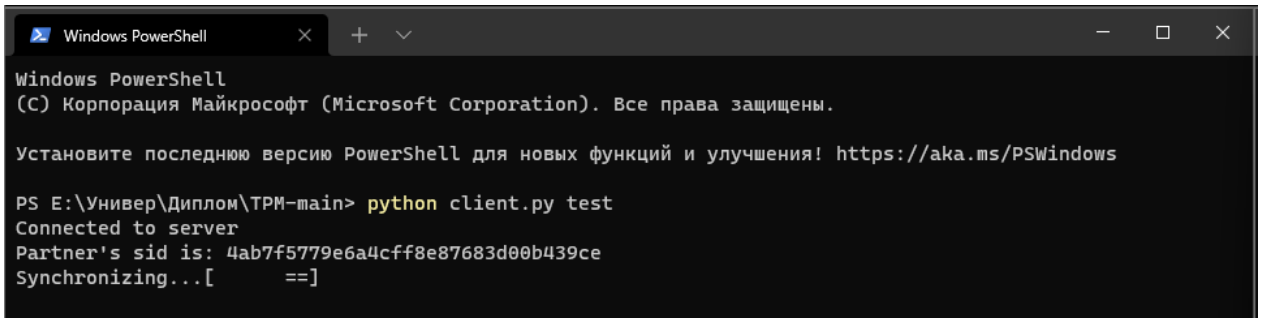


Рисунок 2.14 – Процес синхронізація від зору абонента А

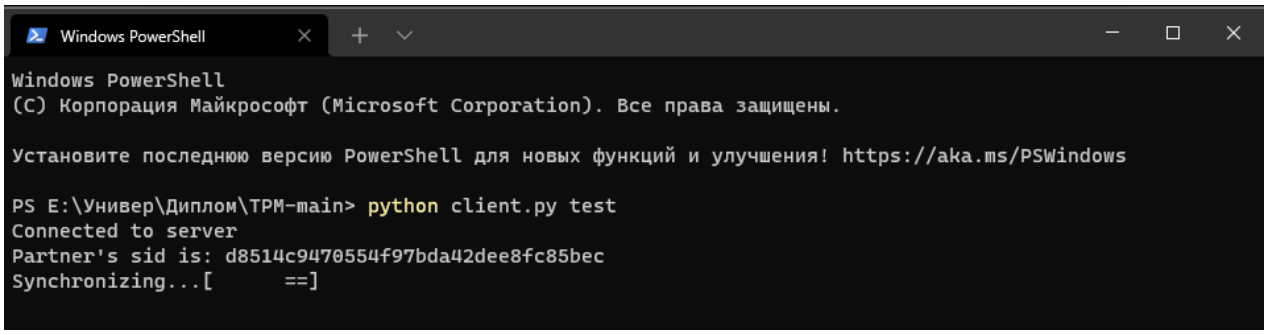


Рисунок 2.15 – Процес синхронізація від зору абонента В

Можна побачити, що користувачам відображаються ідентифікатори співрозмовників, які також може спостерігати адміністратор системи з боку серверу, це можна побачити на рис. 2.16.

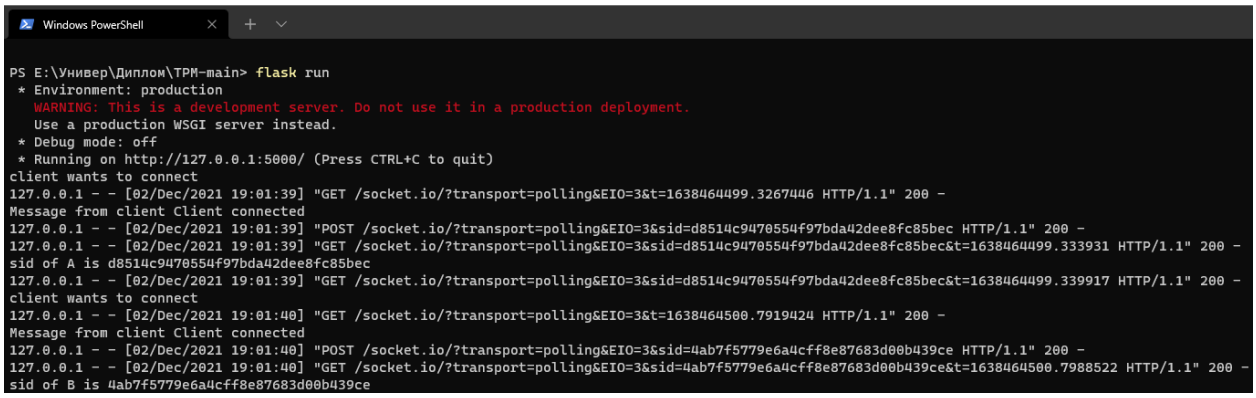


Рисунок 2.16 – Результат успішного підключення двох клієнтів до серверу

Можна побачити, що ідентифікатори співпадають представленим на рис. 2.14-2.15.

Далі спрацьовує функція для синхронізації двох ШНМ. А саме виконується відправка вагових коефіцієнтів та значення вихідного вектору на сервер, який в свою чергу надсилає ці дані протилежним клієнтам. Даний процес з боку сервера можна побачити на рис. 2.17.

Рисунок 2.17 – Процес обміну ваговими коефіцієнтами

В результаті процесу, представленому на рис. 2.17, ШНМ отримують дані для оновлення вагових коефіцієнтів. Даний процес виконується до тих пір поки значення вихідного вектору двох ШНМ не будуть однаковими. Приклад різних значень вихідного вектору наведений на рис. 2.18.

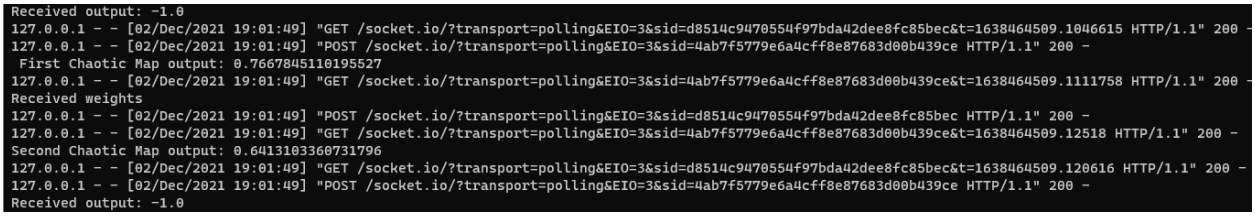


Рисунок 2.18 – Приклад значення вихідного вектору

Процес синхронізації продовжується до тих пір, поки значення вихідних векторів обох ШНМ не будуть однаковими, як це представлено на рис. 2.19.

Рисунок 2.19 – Приклад значень вихідних векторів синхронізованих ШНМ

В результаті успішної синхронізації ШНМ користувачів, вони отримають повідомлення про вдалу синхронізацію зі співрозмовником (див. рис. 2.20).

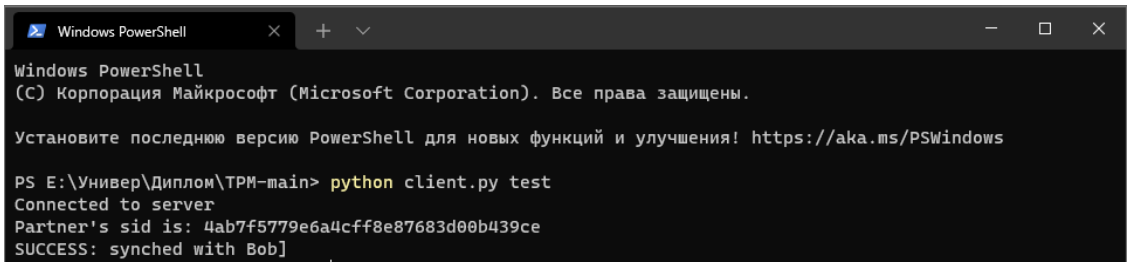


Рисунок 2.20 – Повідомлення про успішну синхронізацію клієнта A

Аналогічне повідомлення отримує другий клієнт (див. рис. 2.21).

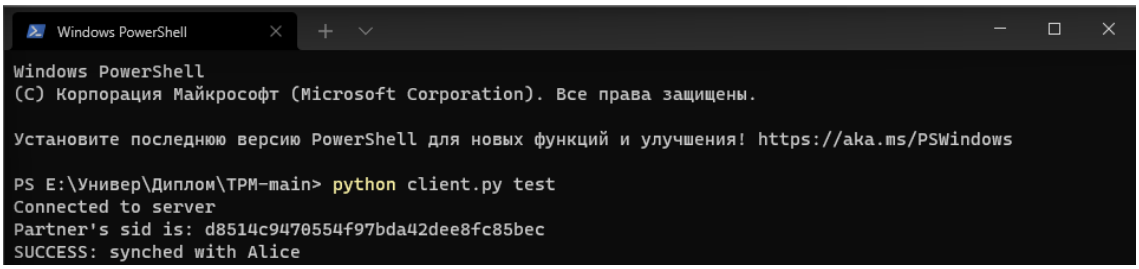


Рисунок 2.21 – Повідомлення про успішну синхронізацію клієнта B

В результаті успішної синхронізації двох клієнтів буде сформований секретний ключ шифрування. Приклад сформованого ключа представлений на рис. 2.22.

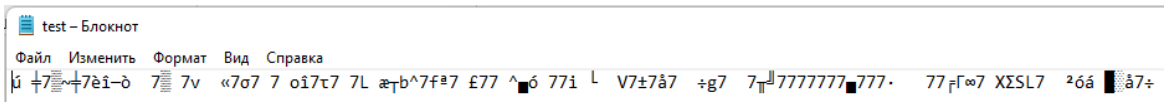


Рисунок 2.22 – Приклад сформованого ключа шифрування

Також, користувачам відкривається веб-додаток, який дозволяє використовуючи сформований ключ шифрування, обмінюватися повідомленнями без можливості заволодіння персональними даними третіми особами.

Представлення інтерфейсу веб-додатка наведений на рис. 2.23.

Рисунок 2.23 – Інтерфейс веб-додатку

Для успішної роботи з веб-додатком, користувачеві слід вказати шлях, де розташований сформований ключ шифрування, шляхом натискання на відповідний перемикач, який в свою чергу відкриває вікно «Провідника» операційної системи. Приклад обрання файлу, який містить ключ шифрування наведений на рис. 2.24.

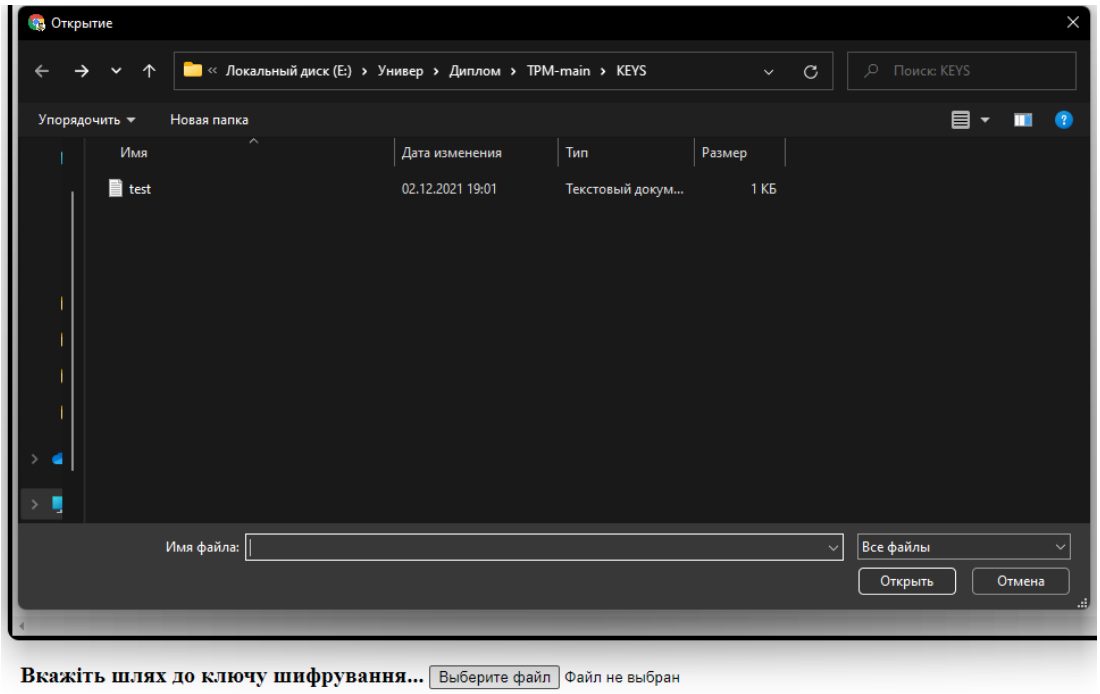


Рисунок 2.24 – Приклад роботи додатка для вибору ключа

Слід зазначити, що подібну процедуру, представлену на рис. 2.24, потрібно виконати обом співрозмовникам, так як без ключа шифрування вони не зможуть ні отримати повідомлення ні надіслати.

Після того, як користувачі надали шлях до ключа шифрування, вони можуть обмінюватися шифрованими повідомленнями. Спочатку їм слід Вказати своє ім’я та надрукувати текст повідомлення у відповідних полях. Та після цього натиснути кнопку «Надіслати».

Приклад процесу обміну повідомленнями наведений на рис. 2.25.

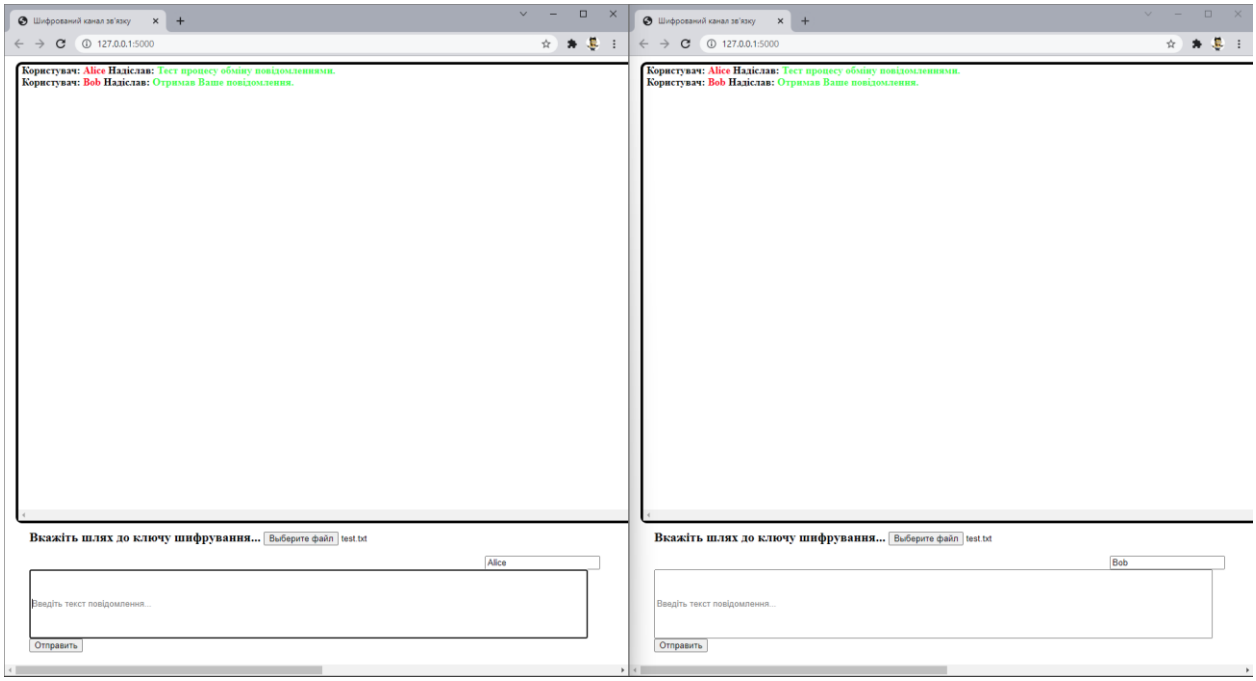


Рисунок 2.25 – Процес обміну повідомленнями

Тепер слід перевірити процес шифрування повідомлень. На серверній частині відображаються отримані та надіслані повідомлення. Але вони відображаються в шифрованому вигляді, що можна побачити на рис. 2.26.

Рисунок 2.26 – Представлення повідомлень з боку серверу

Можна зробити висновок, що система працює, і від клієнтів надсилається вже зашифроване повідомлення, а отримати ключ для розшифрування повідомлення третіми особами не є можливим, тому що він генерується локально і не пересилається між співрозмовниками.

2.8.3 Результати тестування розробленої системи

Тестування ПЗ – це метод визначення того, чи відповідає фактичний ПП очікуваним вимогам, і гарантує, що ПП не має дефектів. Це передбачає запуск програмних/системних компонентів за допомогою ручних або автоматизованих інструментів для оцінки однієї або кількох властивостей, що цікавлять.

Мета тестування ПЗ – знайти помилки, прогалини або відсутні вимоги в порівнянні з реальними вимогами.

Коли триває проект розробки ПЗ, ви повинні знати, що помилки можуть з’явитися на будь-якій фазі життєвого циклу.

Відомо, що деякі з них є невідкритими. Таким чином, не можна ігнорувати важливість забезпечення якості.

Є висока ймовірність того, що остаточний код містить помилки функціональності та дизайну. Для виявлення проблем до появи у критичному середовищі необхідною умовою є проведення тестування ПЗ.

Тестування ПЗ важливе, оскільки якщо в ПЗ є помилки або помилки, їх можна виявити завчасно та виправити до того, як програмний продукт буде доставлено. Правильно перевірений ПП забезпечує надійність, безпеку та високу продуктивність, що призводить до економії часу, економічної ефективності та задоволеності клієнтів.

Це невід’ємна частина процесу. Однак це передбачає величезний відрізок від кишені.

Тим не менш, потрібно мати на увазі, що ціна через збій ПЗ може бути дійсно високою [38-39].

Першим кроком слід виконати перевірку розмітки HTML на наявність помилок написання коду. Даний тест можна провести за допомогою зовнішнього ресурсу Schema.org.

Schema.org – це спільна громадська діяльність, місією якої є створення, підтримка та просування схем для структурованих даних в Інтернеті. Окрім людей із компаній-засновників (Google, Microsoft, Yahoo та Yandex), значну участь бере ширша веб-спільнота через загальнодоступні списки розсилки, такі як public-vocabs@w3.org та через GitHub.

З квітня 2015 року група спільноти W3C Schema.org є основним форумом для співпраці зі схемами та надає список розсилки public-schemaorg@w3.org для обговорень. Проблеми Schema.org відстежуються на GitHub.

Повсякденною роботою Schema.org, включаючи рішення щодо схеми, займається керівна група, до складу якої входять представники компаній-засновників, представник W3C і невелика кількість осіб, які зробили значний внесок у Schema.org . Обговорення керівної групи є публічними [40].

Загальний вид ресурсу наведений на рис. 2.27.

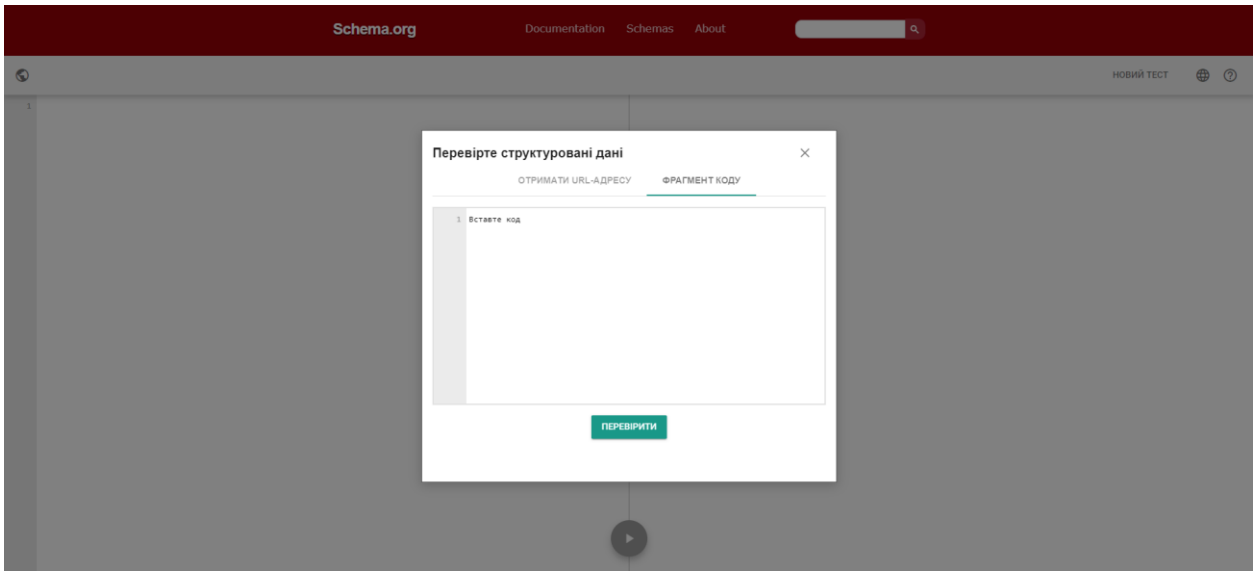


Рисунок 2.27 – Головна сторінка сервісу для перевірки розмітки HTML Schema.org

Перевірку коду веб-додатку виконаємо шляхом копіювання тексту програми до відповідного поля ресурсу Schema.org, як це представлено на рис. 2.28.

Далі, особа що проводить тест, натискає на кнопку «Перевірити», що тягне за собою початок процесу перевірки коду розмітки на наявність помилок.

Як можна побачити на рис. 2.29 представлені результати про відсутність помилок у написані коду веб-додатку. Отже тестування можна вважати вдалим.

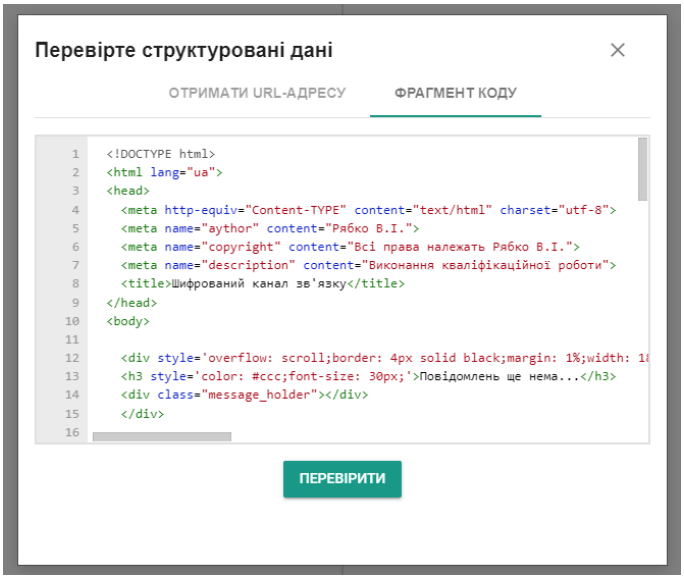


Рисунок 2.28 – Приклад застосування сервісу Schema.org

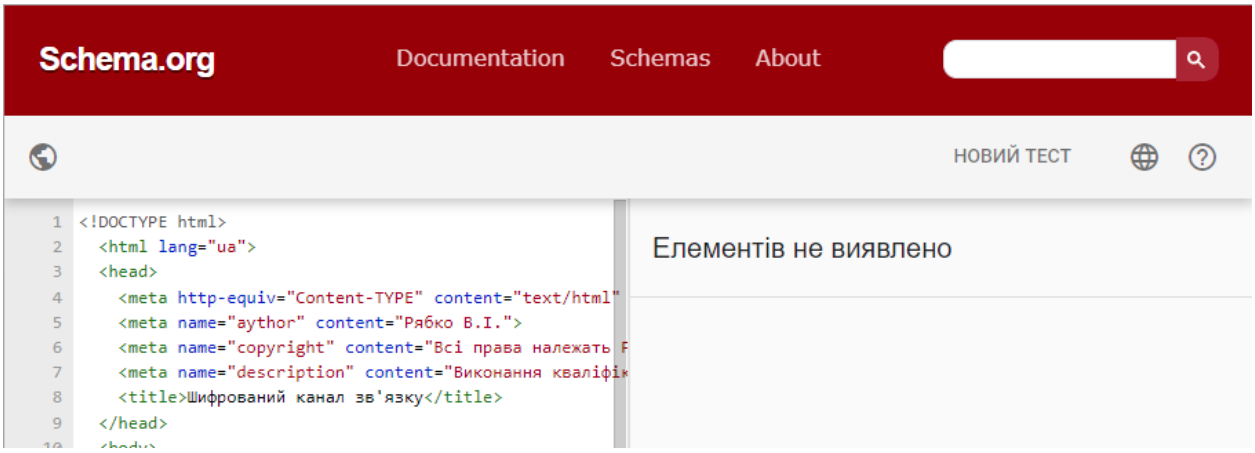


Рисунок 2.29 – Результат перевірки на наявність помилок коду розмітки HTML

До наступного етапу тестування можна віднести тестування методом чорного ящика.

Тестування методом чорного ящика – це метод тестування програмного забезпечення, за допомогою якого функціональні можливості програмних додатків перевіряються без знання внутрішньої структури коду, деталей реалізації та внутрішніх шляхів. Тестування методом чорного ящика в основному зосереджується на введенні та виводі програмних додатків і повністю базується на вимогах і специфікаціях програмного забезпечення. Він також відомий як поведінковий тест [41].

Чому саме «чорний ящик»? Як випливає з назви, спеціалісти з контролю якості працюють з програмним забезпеченням, яке схоже на чорну непрозору скриньку, вміст якої ніхто не бачить. Мета – виявлення помилок в інтерфейсі, поведінці системи; виявити неправильно реалізовані або відсутні функції; помилки в структурах даних, або неправильна організація доступу до зовнішніх баз даних [42].

Результати тестування наведені в табл. 2.5.

Таблиця 2.5 – Процес тестування методом чорного ящика

Сутність тесту	Очікування	Результати
1	2	3
1. Налаштування серверу		
Адміністратор запускає виконання команди для налаштування серверу	Сервер запускається і очікує на підключення користувачів	пройдено
2. Підключення двох ДМП до серверу		
Адміністратор запускає виконання команди для підключення до серверу обох співрозмовників	Користувачі приєднуються до серверу; сервер відображає повідомлення	пройдено

Продовження таблиці 2.5

1	2	3
3. Синхронізація ДМП		
Після підключення до серверу обох клієнтів починається процес синхронізації	Успішна синхронізація ДМП; отримання секретного ключа шифрування	пройдено
4. Запуск веб-додатку		
Після успішної синхронізації виконується перехід до веб-додатку	Користувачам відкривається сторінка в браузері	пройдено
5. Завантаження до веб-додатку ключа шифрування		
Користувачі завантажує ключ шифрування, який був сформований в процесі синхронізації ДМП	Завантаження секретного ключа шифрування; надання можливості для обміну повідомленнями	пройдено
6. Відправка повідомлення		
Абонент «А» вказує своє ім’я та вводить текст повідомлення до відповід-ного поля; натиснення кнопки «Відправити»	Відправлення шифрованого повідомлення від абонента «А» абоненту «В»	пройдено
7. Помилка у вказанні ім’я користувача		
Абонент «В» не вказує своє ім’я; вводиться текст повідомлення; натиснення кнопки «Відправити»	Повідомлення не було відправлене	пройдено

Продовження таблиці 2.5

1	2	3
8. Помилка завантаження ключа шифрування		
Абонент «А» не завантажує ключ шифрування; абонент «В» завантажує ключ шифрування; натиснення кнопки «Відправити»	Абонент «А» не отримує повідомлення	пройдено
9. Відправка порожнього повідомлення		
Абонент вказує ім’я; поле з текстом повідомлення порожнє; натиснення кнопки «Відправити»	Повідомлення відправляється; користувачам відображується порожнє повідомлення з ім’ям відправника	пройдено

2.9 Висновки за розділом

Результатами даного розділу можна представити виконані процеси з проектування та розробки системи шифрованого обміну повідомленнями з використанням ДМП для синхронізації ШНМ співрозмовників. На етапі проектування були визначені макети графічного інтерфейсу, список необхідних для програмної реалізації бібліотек, представлено концептуальну, поведінкову та архітектурну модель проектованої системи. Продемонстрована схема роботи алгоритму для синхронізації двох ДМП для отримання ключа шифрування. Після етапу проектування було проведена описова характеристика процесу програмної реалізації системи. Наступним етапом став процес тестування розробленої системи, результатом якого були виключені можливі технічні помилки під час написання коду системи.

РОЗДІЛ 3
ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ РОЗРОБЛЕНОЇ СИСТЕМИ

3.1 Процес атаки на систему синхронізації

Для перевірки безпеки процедури синхронізації ДМП було використано три методи атаки на систему. Перший – це атака грубою силою, другий – генетичний алгоритм для отримання ваг від кінцевого значення вихідного нейрону та третій – це атака за допомогою власної ДМП. Необхідна умова для того, щоб значення атаки K, N, L були відомі та якими були фактично секретні дані.

3.1.1 Атака методом грубої сили

Атака грубою силою – це метод проб і помилок, який використовується для отримання інформації, тобто атака грубої сили намагається використовувати всі можливі комбінації, які можуть виникнути в отриманих вагових показниках, для досягнення кінцевого значення.

Щоб здійснити атаку грубою силою, зловмисник повинен перевірити всі можливі ключі (усі можливі значення ваг w_{kj}). За K прихованих нейронів, $K \cdot N$ вхідних нейронів та межі ваг L, це дає $(2L+1)^{KN}$ можливостей. Наприклад, конфігурація $K = 3,\ L = 100$ і $N = 3$ дає нам $(201)^9$ ключових можливостей, що унеможливлює атаку при сучасних потужностях комп'ютера [43].

3.1.2 Атака, використовуючи генетичні алгоритми

Генетичні алгоритми – це пошукові алгоритми, засновані на механікі природного відбору та природної генетики. Вони поєднують виживання найбільш пристосованих серед рядкових структур зі структурованим, але рандомізованим обміном інформацією, щоб сформувати алгоритм пошуку з деякими інноваційними властивостями людського пошуку [44].

У кожному поколінні створюється новий набір штучних створінь, використовуючи частки найсильніших із старих; час від часу випробовуючи нову частину. Хоча генетичні алгоритми рандомізовані, вони не є простим випадковим блуканням. Вони ефективно використовують історичну інформацію для спекуляцій щодо нових точок пошуку з очікуваним покращенням ефективності.

Блок-схема генетичного алгоритму показана на рис. 3.1 [45].

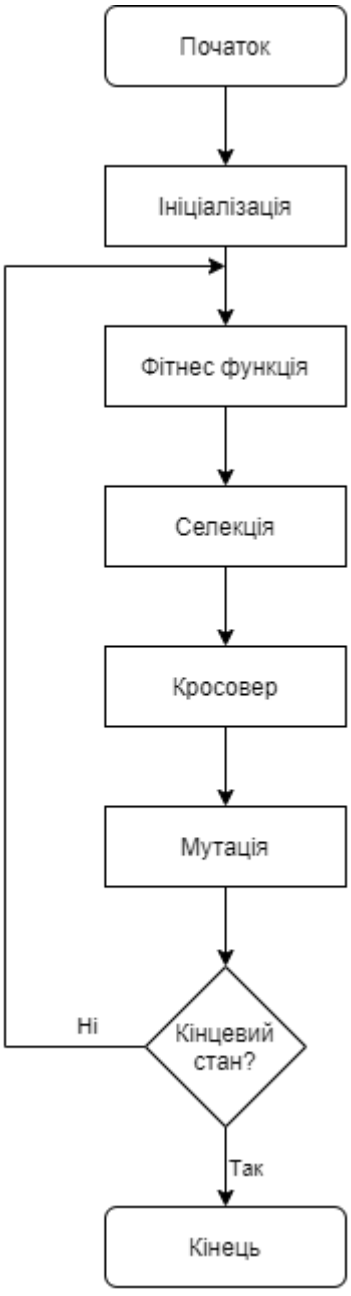


Рисунок 3.1 – Блок-схема використаного генетичного алгоритму

Модифікація стандартного генетичного алгоритму складається з хромосомного представлення застосування та модифікації стандартних генетичних операторів відбору, кросинговеру та мутації.

Хромосома являє собою рядок параметрів нейронної мережі (показаний на рис. 3.1) і її постійна довжина становить 12 цілих чисел. В якості методів відбору використовувалася як рулетка, так і турнірний відбір з елітарністю. Цю опцію можна встановити в програмі перед початком оптимізації. Оператори кросовера – це стандартний одноточковий кросовер або рівномірний кросовер з випадковою маскою. Мутація змінює значення генів за випадковим числом у заданому діапазоні. Цей діапазон можна адаптувати, якщо конвергенція населення надто повільна.

Критерієм придатності для генетичного алгоритму є значення математичного виразу, показаного на рис. 3.1 [46-47]. Придатність для кожного генотипу оцінюється під час оптимізації. Діяльність генетичного алгоритму може бути закінчена після реалізації визначеної кількості поколінь або після досягнення необхідного часу для виконання програми. Генетичний алгоритм має такі налаштування:

- частота кросинговеру = 0,7;
- частота мутацій = 0,01;
- розмір популяції = 1000;
- довжина хромосоми = 48 біт;
- довжина гена = 4 біти;
- максимально допустима кількість поколінь = 200 000.

Було проведено 3 види експериментів, але ніякий не приніс успішний результат. Перший експеримент був для пошуку всіх 12 параметрів вагових коефіцієнтів. Не було жодного успіху навіть після 200 000 поколінь. Тест було проведено багато разів, але прийнятного результату не було знайдено. Другий експеримент був для пошуку лише першого та другого рядків з матриці, що означає лише 8 параметрів вагових коефіцієнтів. Знову ж таки, успіху не було навіть після 200 000 поколінь. Третій експеримент передбачав пошук лише в

одному рядку з матриці. Отже, необхідно було знайти лише 4 параметри вагових коефіцієнтів. Цей експеримент також невдалий. Прийнятний результат може бути знайдений через значну кількість поколінь, а час, необхідний для отримання результату, буде не в рамках розумного.

3.1.3 Атака з використанням власної ДМП

Одна з основних атак може бути здійснена зловмисником, який володіє тією ж ДМП, що й сторони A і B. Він хоче синхронізувати свою ДМП з цими двома сторонами. На кожному кроці можливі три ситуації:

– вихідне значення (A) ≠ Вихідне значення (B): жодна зі сторін не оновлює свої ваги;

– вихідне значення (A) = Вихідне значення (B) = Вихідне значення (E): всі три сторони оновлюють ваги в своїх ДМП.

Вихідне значення (A) = Вихідне значення (B) ≠ Вихідне значення (E): Сторони A і B оновлюють свої ДМП, але зловмисник не може цього зробити. Через цю ситуацію його навчання відбувається повільніше, ніж синхронізація сторін A і B.

Доведено, що синхронізація двох сторін швидше, ніж навчання зловмисника. Його можна покращити шляхом збільшення синаптичної глибини L нейронної мережі. Це забезпечує цьому протоколу достатню безпеку, і зловмисник може знайти ключ лише з малою ймовірністю. Зміна цього параметра збільшує вартість успішної атаки в геометричній прогресії, тоді як зусилля для користувачів зростають поліноміально. Отже, порушення безпеки обміну нейронними ключами відноситься до класу складності високого рівня [48].

Слід зазначити, що даний вид атаки є найбільш перспективним. Для його проведення було реалізовані дві ДМП, які синхронізуються між собою, та ДМП зловмисника, ціль якої – синхронізуватися разом з ДМП абонентів A і B, використовуючи вектори та дані, отримані від ДМП абонентів A і B. Але ДМП зловмисника не має можливості впливати на процес синхронізації.

Шляхом збільшення розміру ДМП, шляхом збільшення кількості прихованих нейронів були реалізовані спроби атаки, використовуючи метод паралельної синхронізації третьої ДМП зловмисника.

Сутність перевірки даним видом атаки передбачає продовження процесу синхронізації до моменту досягання 5000 ітерацій. Тому що це допоможе з'ясувати скільки ітерацій після синхронізації абонентів A і B потребує зловмисник для синхронізації своєю ДМП.

Результатом даного виду атаки можна вважати:

– ДМП зловмисника синхронізувалася паралельно або раніше ДМП абонентів;

– ДМП зловмисника не була синхронізована на протязі 5000 ітерацій;

– ДМП зловмисника синхронізувалася пізніше ДМП абонентів.

Перший та третій варіант можливого результату повідомляє про можливе втручання зловмисника в роботу шифрувальної системи. А другий варіант результату виключає можливість втручання зловмисника.

Для проведення експерименту, використовуючи дану методику визначимо архітектуру ШНМ: кількість прихованих нейронів $K = 3$; кількість вхідних нейронів $K \cdot 3$; діапазон значень, які приймають ваги вхідних та прихованих нейронів $L = 1$.

Архітектура системи змінювалася кожні 5000 ітерацій, витрачених на синхронізацію, а саме збільшувалося значення кількості прихованих нейронів. Результатом зміни архітектури ШНМ, шляхом збільшення кількості прихованих нейронів, представлено на рис. 3.2.

На рис. 3.2 представлений графік відношення кількості прихованих нейронів до необхідної кількості ітерацій для синхронізації ШНМ. Проаналізувавши представлений графік можна визначити, що при збільшенні кількості прихованих нейронів збільшується кількість необхідних ітерацій для синхронізації. Але слід зазначити, що збільшення кількості ітерацій не є різким, що дозволяє значно збільшити кількість прихованих нейронів без значних втрат на час синхронізації.

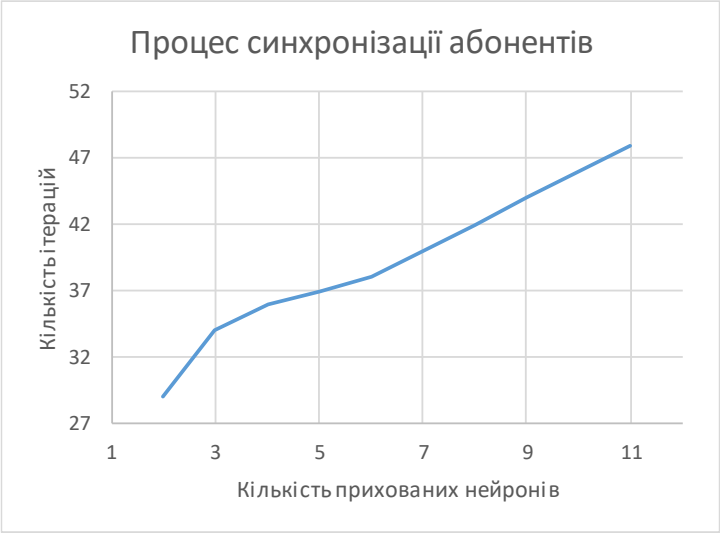


Рисунок 3.2 – Кількість ітерацій, потрібних для синхронізації ДМП

Після збільшення кількості прихованих нейронів було проаналізовано графік (див. рис. 3.3) відношення невдалих спроб на синхронізацію ДМП зловмисника до кількості прихованих нейронів.

Приходячи до висновків, що при збільшенні кількості прихований нейронів до 6 можна спостерігати різкий спад вдалих спроб на синхронізацію ДМП зловмисника, але так як ще спостерігається наявність вдалих процесів синхронізації, кількість прихованих нейронів збільшується до 9.

Недоліком такої архітектури можна вважати те, що на кожен прихований нейрон в даній архітектурі ШНМ приходиться лише 3 вхідних нейрона. Архітектура такого типу не використовується в реальних умовах.

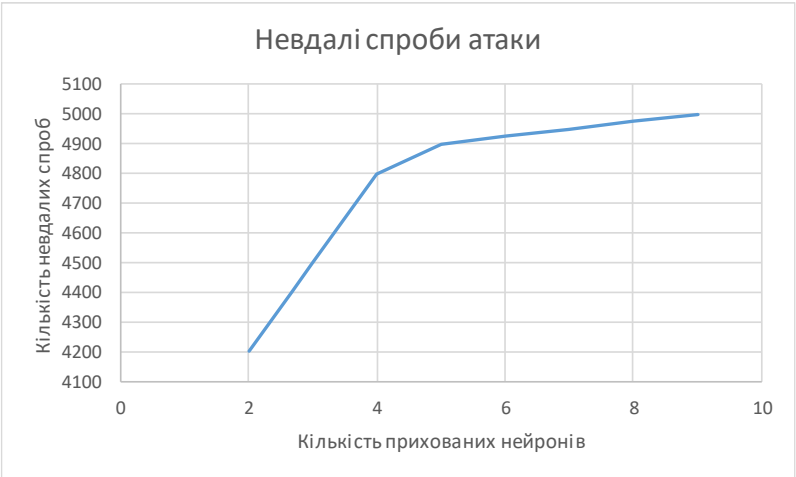


Рисунок 3.3 – Кількість невдалих спроб синхронізації ДМП зловмисника

Далі розглянемо процес атаки до та після синхронізації абонентів А і В.

На рис. 3.4 представлений графік відношення вдалих спроб синхронізації ДМП зловмисника до кількості прихованих нейронів до завершення синхронізації абонентів А і В.

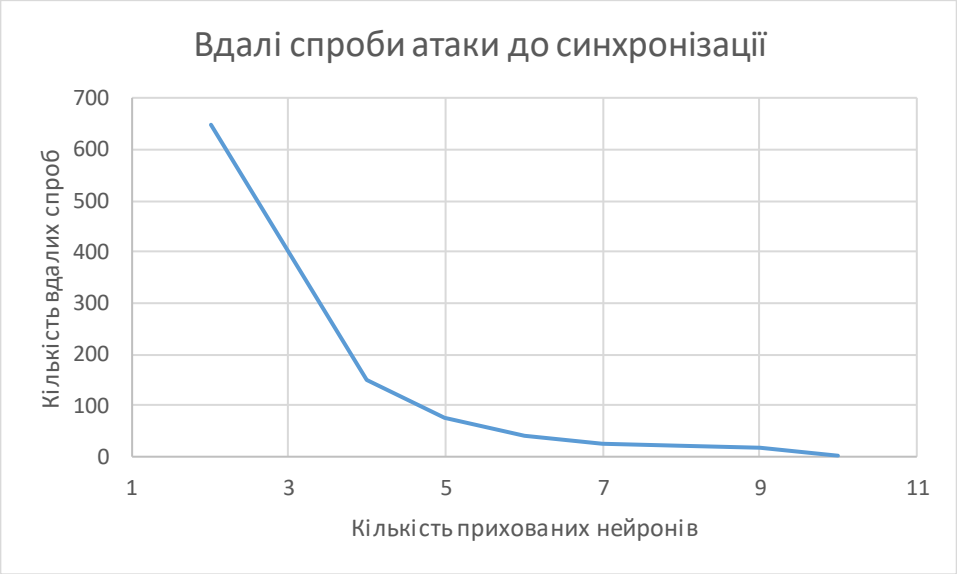


Рисунок 3.4 – Кількість вдалих спроб синхронізації ДМП до синхронізації абонентів А і В

Можна спостерігати, що при збільшенні кількості прихованих нейронів до 6 кількість вдалих спроб синхронізації ДМП зловмисника зменшується до 1/100 від всієї кількості спроб.

При збільшенні кількості прихованих нейронів до 10 спостерігається повне виключення можливості на вдалу синхронізацію ДМП зловмисника.

На рис. 3.5 представлений графік відношення вдалих спроб синхронізації ДМП зловмисника до кількості прихованих нейронів після завершення синхронізації абонентів А і В.

В порівнянні від атаки до завершення синхронізації кількість вдалих спроб на синхронізацію ДМП зловмисника вже при збільшенні прихованих нейронів до 4 не спостерігається. Це зумовлюється тим, що після завершення синхронізації між ДМП абонентів А і В вони перестають оновлювати свої ваги і ДМП зловмисника не має даних для синхронізації.

Рисунок 3.5 – Кількість вдалих спроб синхронізації ДМП після синхронізації абонентів A і B

Проаналізувавши всі результати атак можна зробити висновки, що навіть при збільшенні кількості прихованих нейронів та кількості вхідних нейронів до 4 можна спостерігати значне зменшення кількості вдалих спроб на втручання ДМП зловмисника.

3.2 Вплив на час, витрачений на синхронізацію ДМП абонентів

Значний вплив на час виконує архітектура заданої ШНМ: кількість прихованих нейронів, кількість вхідний нейронів та діапазон значень, які виступають в ролі синоптичних ваг [49]. Також загальна кількість та діапазон ваг має значний вплив на розмір та складність сформованого криптографічного ключа.

До вимог в бік архітектури ДМП, які впливатимуть на час синхронізації можна віднести наступне: кількість вхідних нейронів повинно бути мінімальним, но бути достатнім для досягання задовільного часу синхронізації; кількість прихованих нейронів повинно бути максимальним, но не перевищувати граничне значення для досягання задовільного часу синхронізації; значення, яке відповідає за діапазон значень повинен бути

максимальним, але не перевищувати граничне значення для досягання задовільного часу синхронізації [50].

Для дослідження факторів, які впливають на час, витрачений на синхронізацію ДМП, було проведено декілька експериментальних досліджень. Для їх проведення було реалізовано дві ДМП, архітектура якої змінювалася.

Першим експериментом стало дослідження впливу на час синхронізації кількості вхідних нейронів. Сутність даного експерименту полягає в зміненні тільки кількості вхідних нейронів та правила модифікації вагових коефіцієнтів.

Результати дослідження наведені в табл. 3.1.

Таблиця 3.1 – Результати першого експерименту

№ експерименту	Архітектура ШНМ				Середнє значення кількості ітерацій
	Правило модифікації ваг	Кількість прихованих нейронів	Кількість вхідних нейронів	Діапазон значень синоптичних ваг	
1	2	3	4	5	6
1	Хебба	10	3	1.00	48
2	Хебба	10	4	1.00	50
3	Хебба	10	5	1.00	55
4	Хебба	10	10	1.00	57
5	Хебба	10	50	1.00	60
6	Хебба	10	100	1.00	63
7	Анті-Хебба	10	3	1.00	44
8	Анті-Хебба	10	4	1.00	48
9	Анті-Хебба	10	5	1.00	50
10	Анті-Хебба	10	10	1.00	53

Продовження таблиці 3.1

1	2	3	4	5	6
11	Анті-Хебба	10	50	1.00	61
12	Анті-Хебба	10	100	1.00	70
13	Випадкове блукання	10	3	1.00	71
14	Випадкове блукання	10	4	1.00	96
15	Випадкове блукання	10	5	1.00	125
16	Випадкове блукання	10	10	1.00	189
17	Випадкове блукання	10	50	1.00	269
18	Випадкове блукання	10	100	1.00	341

Виходячи з результатів проведеного дослідження можна зробити висновки, що при збільшенні кількості вхідних нейронів спостерігається незначне збільшення кількості ітерацій для досягнення синхронізації ДМП. Це дозволяє використовувати велику кількість вхідних нейронів, що тягне за собою збільшення кількості використаних синоптичних ваг, які використовуються для формування ключа шифрування.

Слід відзначити, що правило Хебба та Анти-Хебба демонструють схожі результати, але використання правила Хебба все ж таки має перевагу зі свого боку.

Також слід відзначити, що при збільшенні кількості вхідних нейронів збільшується кількість інформації, що передається мережею, та збільшення часу на синхронізацію ДМП, що дає зловмиснику більше часу на можливу спробу атаки.

Отже, для вирішення цих недоліків доречніше буде змінювати кількість прихованих нейронів, а кількість вхідних залишити незмінним. Це стане сутністю другого експерименту, результати якого представлені в табл. 3.2.

Таблиця 3.2 – Результати другого експерименту

№ експерименту	Архітектура ШНМ				Середнє значення кількості ітерацій
	Правило модифікації ваг	Кількість прихованих нейронів	Кількість вхідних нейронів	Діапазон значень синоптичних ваг	
1	2	3	4	5	6
1	Хебба	3	10	1.00	41
2	Хебба	4	10	1.00	43
3	Хебба	5	10	1.00	44
4	Хебба	10	10	1.00	51
5	Хебба	50	10	1.00	79
6	Хебба	100	10	1.00	74
7	Анті-Хебба	3	10	1.00	37
8	Анті-Хебба	4	10	1.00	47
9	Анті-Хебба	5	10	1.00	49
10	Анті-Хебба	10	10	1.00	55
11	Анті-Хебба	50	10	1.00	69
12	Анті-Хебба	100	10	1.00	82
13	Випадкове блукання	3	10	1.00	121
14	Випадкове блукання	4	10	1.00	135
15	Випадкове блукання	5	10	1.00	145

Продовження таблиці 3.2

1	2	3	4	5	6
16	Випадкове блукання	10	10	1.00	196
17	Випадкове блукання	50	10	1.00	293
18	Випадкове блукання	100	10	1.00	351

В результаті проведення другого експерименту можна відзначити аналогічну перевагу використання правил Хебба та Анті-Хебба над правилом випадкового блукання. Найкращим підходом є використання правила Хебба.

Можна відзначити, що при збільшенні кількості прихованих нейронів спостерігається незначний зріст кількості необхідних ітерацій.

Наступним експериментом стане перевірка впливу на час синхронізації значення L, яке визначає діапазон синоптичних ваг. Для цього було проведено третій експеримент, результати якого представлені в табл. 3.3.

Таблиця 3.3 – Результати третього експерименту

№ експерименту	Архітектура ШНМ				Середнє значення кількості ітерацій
	Правило модифікації ваг	Кількість прихованих нейронів	Кількість вхідних нейронів	Діапазон значень синоптичних ваг	
1	2	3	4	5	6
1	Хебба	10	10	0.50	24
2	Хебба	10	10	0.50	28
3	Хебба	10	10	0.50	59

Продовження таблиці 3.3

1	2	3	4	5	6
4	Хебба	10	10	1.00	145
5	Хебба	10	10	1.00	156
6	Хебба	10	10	1.00	196
7	Анті-Хебба	10	10	1.50	22
8	Анті-Хебба	10	10	1.50	55
9	Анті-Хебба	10	10	1.50	99
10	Анті-Хебба	10	10	2.00	121
11	Анті-Хебба	10	10	2.00	251
12	Анті-Хебба	10	10	2.00	293
13	Випадкове блукання	10	10	2.50	39
14	Випадкове блукання	10	10	2.50	135
15	Випадкове блукання	10	10	2.50	1325
16	Випадкове блукання	10	10	3.00	1244
17	Випадкове блукання	10	10	3.00	7855
18	Випадкове блукання	10	10	3.00	99999

Висновками проведення третього експерименту можна визначити те, що збільшення значення L збільшує час, який витрачається на синхронізацію.

3.3 Висновки за розділом

До результатів виконання за даним розділом можна віднести проведення досліджень з наступних напрямів: проведення експериментальних

досліджень, спрямованих на визначення криптостійкості системи, шляхом проведення атак за методами грубої сили, використовуючи генетичні алгоритми та з використання ДМП зловмисником; та проведено дослідження факторів, які впливають на час синхронізації двох ДМП, а саме на змінення кількості вхідних нейронів, змінення кількості прихованих нейронів та змінення значення діапазону синоптичних ваг ШНМ. В результаті проведених досліджень було визначено, що система має відповідний рівень криптостійкості від атак, та було використана оптимальна архітектура системи для збереження часу, який витрачається на процес синхронізації.

ВИСНОВКИ

В результаті виконання випускної кваліфікаційної роботи магістра було виконані наступні задачі:

– аналіз проблемної області;

– постановка задачі на розробку;

– проектування системи шифрування повідомлень;

– програмна реалізація спроектованої системи;

– проведення тестування розробленої системи;

– проведення досліджень на перевірку криптостійкості реалізованої системи;

– проведення дослідження на виявлення факторів, які впливають на час синхронізації ДМП.

Результатом виконання кваліфікаційної роботи – є розроблена система шифрованого обміну повідомленнями на основі деревовидних машин парності, яка може бути впроваджена на підприємстві.

В процесі розробки системи були вдосконалені навички розробки великих систем та проведено знайомство з алгоритмами, які виконують функції, які вимагає система.

Провівши тестування та дослідження розробленої системи, як розробник, можу виділити наступні недоліки:

– відсутність можливості підключення трьох та більше співрозмовників до каналу зв’язку;

– відсутність можливості пересилання файлів між користувачами;

– несучасний інтерфейс.

А до переваг можна визначити наступне:

– виключення необхідності пересилки ключа шифрування через незахищений канал зв’язку;

– високий рівень криптостійкості системи;

– безкоштовне впровадження на підприємстві.

Під час подальшої технічної підтримки системи можна додати функціонал, який відповідає за пересилання файлів між користувачем, функціонал, який виконує роль вибору співрозмовника.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Tell Me About It: How Do Messaging Apps Respond to Privacy? [Електронний ресурс]. – режим доступу: https://www.commonsense.org/education/articles/tell-me-about-it-how-do-messaging-apps-respond-to-privacy

2. Data Security Using Neural Networks Can Provide Additional Security Layers [Електронний ресурс]. – режим доступу: https://www.copado.com/devops-hub/blog/data-security-using-neural-networks-can-provide-additional-security-layers

3. Diffie–Hellman (DH) Algorithm [Електронний ресурс]. – режим доступу: https://www.hypr.com/diffie-hellman-algorithim/

4. [ElGamal, 1984] El Gamal T. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms // Proceedings of CRYPTO 84 on Advances in Cryptology. – Santa Barbara, California, USA: Springer-Verlag New York, Inc., 1985. – с. 10-18. – ISBN 0-387-15658-5.

5. [ElGamal, 1985] El Gamal T. A public key cryptosystem and a signature scheme based on discrete logarithms // IEEE Transactions on Information Theory. – 1985. – июль. – т. 31, № 4. – с. 469-472. – DOI: 10.1109/TIT.1985.1057074.

6. I. Kanter, and W. Kinzel, “The Theory of Neural Networks and Cryptography.” Quantum Computers and Computing. V. 5, No1, pp. 130-140 (2005).

7. A. Ruttor, W. Kinzel, and I. Kanter, “Dynamics of Neural Cryptography.” Physical review. E 75, 056104, Statistical, nonlinear, and soft matter physics [1539-3755] (2007).

8. S. Santhanalakshmi, K. Sangeeta, and G. K. Patra, “Design o Stream Cipher for Text Encryption using Soft Computing based Techniques.” IJCSNS International Journal of Computer Science and Network Security, VOL.12 No.12, pp. 149-152 (2012).

9. W. Kinzel, and I. Kante, “Neural Cryptography.” 9th International Conference on Neural Information Processing, Singapore, pp. 1351-1354 vol.3 (2002).

10. A. Ruttor, W. Kinzel, R. Nach, and I. Kanter, “Genetic Attack on Neural Cryptography.” Phys. Rev. E 73.036121 (2006)

11. P. S. Revankar, W. Z. Gandhare, D. T. Rathod, “Neural synchronization with queries,” ICSAP 10, in press.

12. Andreas Ruttor, Wolfgang Kinzel and Ido Kanter, “Neural cryptography with queries”, Journal of Statistical Mechanics: Theory and Experiment, doi:1088/1742-5468/2005/01/P01009, Jan. 2005.

13. Kanter I. Analytical Study of Time Series Generation by Feed-Forward Networks / Kanter I., Kessler D.A., Priel A., Eisenstein E. // Phys. Rev. Lett. – 1995. – Vol.75. – P. 2614-2617.

14. Червяков Н. Применение искусственных нейронных сетей и системы остаточных классов в криптографии / Н. Червяков, А. Евдокимов // М.: ФИЗМАЛИТ. – 2012. – С. 280 – ISBN 978-5-9221-1386-1

15. E. Klein, R. Mislovaty, I. Kanter, A. Ruttor, and W. Kinzel. Synchronization of neural networks by mutual learning and its application to cryptography. / E. Klein, R. Mislovaty, I. Kanter, A. Ruttor, and W. Kinzel. L. K. Saul, Y. Weiss, and L. Bottou, editors. // Advances in Neural Information Processing Systems. MIT Press, Cambridge, MA – 2005. – Vol 17. – P. 689–696.

16. Python 3.10.0 documentation [Електронний ресурс]. – режим доступу: https://docs.python.org/3/

17. The Hitchhiker’s Guide to Python! [Електронний ресурс]. – режим доступу: https://docs.python-guide.org/writing/documentation/

18. Documenting Python Code: A Complete Guide [Електронний ресурс]. – режим доступу: https://realpython.com/documenting-python-code/

19. HTML Introduction Guide [Електронний ресурс]. – режим доступу: https://www.w3schools.com/html/html_intro.asp

20. What Is HTML? Hypertext Markup Language Basics Explained [Електронний ресурс]. – режим доступу: https://www.hostinger.com/tutorials/what-is-html

21. What is JavaScript used for? [Електронний ресурс]. – режим доступу: https://www.hackreactor.com/blog/what-is-javascript-used-for

22. GUI [Електронний ресурс]. – режим доступу: https://www.computerhope.com/jargon/g/gui.htm

23. Use-case diagrams [Електронний ресурс]. – режим доступу: https://www.ibm.com/docs/en/rational-soft-arch/9.6.1?topic=diagrams-use-case

24. State diagrams [Електронний ресурс]. – режим доступу: https://www.microtool.de/en/knowledge-base/what-is-a-state-diagram/

25. Component Diagram [Електронний ресурс]. – режим доступу: https://www.smartdraw.com/component-diagram/

26. W. Diffie and M. Hellman, “New directions in cryptography,” IEEE Trans. on Info. Theory 22, 644-654 (1976).

27. Welcome to Flask’s documentation [Електронний ресурс]. – режим доступу: https://flask.palletsprojects.com/en/2.0.x/quickstart/

28. Flask-SocketIO [Електронний ресурс]. – режим доступу: https://flask-socketio.readthedocs.io/en/latest/index.html

29. Модуль pickle [Електронний ресурс]. – режим доступу: https://pythonworld.ru/moduli/modul-pickle.html

30. Библиотека sys [Електронний ресурс]. – режим доступу: https://all-python.ru/osnovy/sys.html

31. python-socketio [Електронний ресурс]. – режим доступу: https://python-socketio.readthedocs.io/en/latest/

32. NumPy [Електронний ресурс]. – режим доступу: https://pythonworld.ru/numpy/1.html

33. Модуль Math [Електронний ресурс]. – режим доступу: https://python-scripts.com/math

34. Модуль random [Електронний ресурс]. – режим доступу: https://pythonworld.ru/moduli/modul-random.html

35. What is jQuery? [Електронний ресурс]. – режим доступу: https://jquery.com/

36. Getting started with Socket.io [Електронний ресурс]. – режим доступу: https://www.scaleway.com/en/docs/tutorials/socket-io/

37. CryptoJS [Електронний ресурс]. – режим доступу: https://cryptojs.gitbook.io/docs/

38. 7 Reasons Why Software Testing is Important [Електронний ресурс]. – режим доступу: https://www.indiumsoftware.com/blog/why-software-testing/

39. What is software testing? [Електронний ресурс]. – режим доступу: https://www.ibm.com/topics/software-testing

40. Schema.org [Електронний ресурс]. – режим доступу: https://schema.org/docs/about.html

41. What is BLACK Box Testing? Techniques, Example & Types [Електронний ресурс]. – режим доступу: https://www.guru99.com/black-box-testing.html

42. Black-Box Testing: Algorithm & Methods [Електронний ресурс]. – режим доступу: https://qatestlab.com/resources/knowledge-center/black-box-testing/

43. Andreas Ruttor, Wolfgang Kinzel, Rivka Naeh, and Ido Kanter. Genetic attack on neural cryptography // Physical Review E: journal. – 2006.

44. M. Turčaník, and M. Líška, “Simulácia koevolúcie umelých neurónových sietí pomocou genetických algoritmov.” MATLAB 2001: Sborník příspěvků 9. ročníku konference, 11. 10. 2001, Praha. – Praha: VŠCHT, 2001. – ISBN 80-7080-446-7, pp. 419-423 (2001).

45. M. Turčaník, and M. Líška, “Genetic Algorithm Optimization of the Dataflow Processor.” Proceedings of the 2nd Euro-International Symposium on Computational Intelligence: E-ISCI – 2002, June 16 – June 19, 2002, Košice. – Amsterdam: OIS Press (2002).

46. S.C. Chu and H.L. Fang, – Genetic Algorithms vs. Tabu search in Timetable scheduling‖, Third International Conference on Knowledge-Based Intelligent Engineering System, Australia, pp. 492-95, 1999

47. Генетические алгоритмы [Електронний ресурс]. – режим доступу: https://docplayer.ru/71742640-Tema-5-geneticheskie-algoritmy.html

48. Allam A. Improved Security of Neural Cryptography Using Don't-Trust-My-Partner and Error Prediction / A.Allam, H.Abbas // International Joint Conference on Neural Networks (IJCNN09) – 2009. – P. 121-127

49. Volkmer M. Entity Authentication and Authenticated Key Exchange with Tree Parity Machines / M. Volkmer, S. Wallner // – 2012. – Vol.112. – P. 1–6.

50. Малік О.Р. Емпіричне дослідження функції розподілу часу синхронізації нейронних мереж в протоколі обміну ключа / О.Р. Малік // Технологический аудит и резервы производства – 2014. – № 4/1(18). – С. 26-31.

Додаток А

Зауваження нормоконтролера

Таблиця А.1 – Зауваження нормоконтролера

Сторінка	Позначення	Зміст зауваження

Додаток Б

Лістинг основних компонентів додатку

Б.1 Лістинг файлу «server.py»

```
from flask import Flask, request, render_template
from flask_socketio import SocketIO, join_room, leave_room, send, emit, disconnect
import pickle


pickle.dump({}, open("users.p", "wb"))
pickle.dump({}, open("channels.p", "wb"))

app = Flask(__name__)
socketio = SocketIO(app)

@app.route('/')
def index(methods=['GET', 'POST']):
    if request.method == 'GET':
        return render_template('session.html')
    else:
        return 'POST'


def messageReceived(methods=['GET', 'POST']):
    print('message was received!!!')


@socketio.on('my event')
def handle_my_custom_event(json, methods=['GET', 'POST']):
    print('received my event: ' + str(json))
    socketio.emit('my response', json, callback=messageReceived)

K = 16
L = 100
N = 16

@socketio.on("connect")
def connect():
    print("client wants to connect")
    emit("status", { "message": "Connected. Hello!" })

@socketio.on('join')
def on_join(data):
    channels = pickle.load(open("channels.p", "rb"))
    users = pickle.load(open("users.p", "rb"))

    if data['channel'] in channels.keys():
```

```
        if channels[data['channel']].num_users == 2:
            emit("status", { "message": "Channel is full, connect to other channel" })
            disconnect(request.sid)
        else:
            users[request.sid] = data['channel']
            join_room(data['channel'])
            channels[data['channel']].num_users += 1
            channels[data['channel']].BSid = request.sid
            pickle.dump(channels, open("channels.p", "wb"))
            pickle.dump(users, open("users.p", "wb"))
            print('sid of B is ' + request.sid)
            emit("status",
                {
                "message": "Connected to channel as Bob, room is full",
                'assign Bob': 'B',
                'channel': data['channel'],
                'start' : 'yes',
                'ASid': channels[data['channel']].ASid,
                'BSid': channels[data['channel']].BSid
                },
                room = data['channel'],
            )

    else:
        users[request.sid] = data['channel']
        join_room(data['channel'])
        channels[data['channel']] = TPMSync(L, K, N)
        channels[data['channel']].num_users += 1
        channels[data['channel']].ASid = request.sid
        print('sid of A is ' + request.sid)
        pickle.dump(channels, open("channels.p", "wb"))
        pickle.dump(users, open("users.p", "wb"))
        emit("status",
            {
            "message": "Connected to channel as Alice",
            'assign Alice': 'A'
            }
        )

@socketio.on('my message')
def handle_message(data):
    print("Message from client" + data)

@socketio.on('weights')
def handle_message(data):
    msg = {
        'vector': data['vector'],
        'output': data['output']
        }
    emit('get_weights', msg, room = data['sid'])
    print("Received weights")
```

```
@socketio.on('send_output')
def handle_message(data):
    emit('output_received', data, room = data['sid'])
    print("Received output: " + str(data['output']))


@socketio.on('send_chaos_output')
def handle_message(data):
    emit('receive_chaos_output', data, room = data['sid'])
    print(" First Chaotic Map output: " + str(data['output']))

@socketio.on('confirm_chaos_output')
def handle_message(data):
    emit('receive_chaos_output2', data, room = data['sid'])
    print("Second Chaotic Map output: " + str(data['output']))

class TPMSync:

    def __init__(self, l_, k_, n_):
        self.num_users = 0.0
        self.ASid = None
        self.BSid = None

        self.L = l_
        self.K = k_
        self.N = n_


if __name__ == '__main__':
    socketio.run(app, debug=True)
```

Б.2 Лістинг файлу «client.py»

```
import sys
import socketio
import numpy as np
import math
from random import randrange
import webbrowser
from cryptography.fernet import Fernet


sio = socketio.Client()


@sio.on('receive_chaos_output2')
def on_message(msg):
    if msg['output'] == tpmclient.tpm.chaosmap():
        tpmclient.IsSync = True
        print('SUCCESS: synched with Bob')
        webbrowser.open ('http://127.0.0.1:5000/', new=2)
        sio.disconnect()
```

```
        tpmclient.save_key()

        # save key to KEYS

@sio.on('receive_chaos_output')
def on_message(msg):
    sio.emit('confirm_chaos_output',
        {
            'msg':'sending chaos output',
            'output': tpmclient.tpm.chaosmap(),
            'sid': tpmclient.partner_sid
        }
    )

    if msg['output'] == tpmclient.tpm.chaosmap():
        tpmclient.IsSync = True
        print('SUCCESS: synched with Alice')
        webbrowser.open ('http://127.0.0.1:5000/', new=2)
        sio.disconnect()
        tpmclient.save_key()

        # save key to KEYS


@sio.on('output_received')
def on_message(msg):
    if tpmclient.tpm.out == msg['output']:
        tpmclient.tpm.update_weights(msg['output'])

    if tpmclient.n >= 200:
        if tpmclient.n  % 10 == 0:
            tpmclient.send_chaos_output()

    if not tpmclient.IsSync:
        tpmclient.send_vector_and_output()
        # print("output received", + msg['output'])
        animation.update()

@sio.on('get_weights')
def on_message(msg):
    try:
        vector = msg['vector']
        list_vec = [np.array(vector[x:x+16]) for x in range(0, len(vector), 16)]
        tpmclient.receive_vector(list_vec)
        tpmclient.send_output()
        if tpmclient.tpm.out == msg['output']:
            tpmclient.tpm.update_weights(msg['output'])
        # print("output received", + msg['output'])
        animation.update()
    except socketio.exceptions.BadNamespaceError:
        pass
```

```
@sio.on('status')
def on_message(msg):
    if 'assign Alice' in msg:
        tpmclient.user = msg['assign Alice']
    if 'assign Bob' in msg and not tpmclient.user:
        tpmclient.user = msg['assign Bob']
    if 'start' in msg:
        if tpmclient.user == 'A':
            tpmclient.partner_sid = msg["BSid"]
            # print(' B sid is ' + tpmclient.partner_sid)
            tpmclient.send_vector_and_output()
        else:
            tpmclient.partner_sid = msg["ASid"]
            # print(' A sid is ' + tpmclient.partner_sid)

        print("Partner's sid is: " + tpmclient.partner_sid)

    # print('message from server: ' + msg['message'])

@sio.event
def connect():
    sio.emit('my message', " Client connected", namespace='/')
    sio.emit('join', {'channel': CHANNEL}, namespace='/')
    print("Connected to server")


class TPMClient:
    def __init__(self):
        self.user = None
        self.tpm = TPM(16, 16, 100)
        self.n = 0
        self.IsSync = False

    def send_vector_and_output(self):
        vector = self.rand_vec()
        list_vec = [np.array(vector[x:x+16]) for x in range(0, len(vector), 16)]
        self.tpm.get_output(list_vec)

        sio.emit('weights',
            {
                'msg':'sending random vector and output',
                'vector': vector,
                'output': self.tpm.out,
                'sid': tpmclient.partner_sid
            }
        )
        self.n += 1

    def receive_vector(self, vector_):
        vec = [np.array(x) for x in vector_]
        self.tpm.get_output(vec)
```

```
    def rand_vec(self):
        l = []
        for i in range(256):
            l.append(randrange(-100, 100))
        return l

    def send_output(self):
        sio.emit('send_output',
            {
                'msg':'sending output',
                'output': self.tpm.out,
                'sid': tpmclient.partner_sid
            }
        )

    def send_chaos_output(self):
        sio.emit('send_chaos_output',
            {
                'msg':'sending chaos output',
                'output': self.tpm.chaosmap(),
                'sid': tpmclient.partner_sid
            }
        )

    def save_key(self):
        re = [abs(item+155) for sublist in self.tpm.weights for item in sublist]
        key = bytes(abs(x) for x in re).decode('cp437')
        with open("KEYS/{}.txt".format(CHANNEL), "w", encoding="utf-8") as text_file:
            print(key, file=text_file)
        key1 = Fernet.generate_key()
        with open("secret.key", "wb") as key_file:
            key_file.write(key1)


class TPM:
    def __init__(self, k_, n_, l_):
        self.k = k_
        self.n = n_
        self.l = l_
        self.weights = self.initialize_w()
        self.inputs = None
        self.H = np.zeros(self.k)
        self.out = None
        self.X = None

    def get_output(self, input_):
        self.X = input_
        self.out = 1

        for i in range(self.k):
            self.H[i] = self.signum(np.dot(input_[i], self.weights[i]))
            self.out *= self.signum(np.dot(input_[i], self.weights[i]))
```

```
        def initialize_w(self):
            p = []
            for i in range(self.k):
                p.append(np.random.randint(-self.l, self.l, size=self.n))
            return p

        def signum(self, x):
            return math.copysign(1, x)

        def update_weights(self, outputB):
            for i in range(self.k):
                for j in range(self.n):
                    self.weights[i][j] += self.X[i][j] * self.out * self.isequal(self.out, self.H[i]) *
self.isequal(self.out, outputB)
                    self.weights[i][j] = self.g(self.weights[i][j])

        def isequal(self, A, B):
            if A==B:
                return 1.0
            else:
                return 0.0

        def g(self, w):
            if w > self.l:
                return self.l
            if w < -self.l:
                return -self.l
            else:
                return w

        def chaosmap(self):
            r = sum(list(np.hstack(self.weights)))

            rr = sum([abs(x) for x in (list(np.hstack(self.weights)))])

            t = float(abs(r)) / float(rr)
            x = t
            for i in range(rr):
                x = (3.6 + t/2)* x *(1 - x)
            return x


    class Animation:
        def __init__(self):
            self.animation = [
                        "Synchronizing  [       ]",
                        "Synchronizing  [=      ]",
                        "Synchronizing  [===    ]",
                        "Synchronizing  [====   ]",
                        "Synchronizing. [=====  ]",
                        "Synchronizing. [====== ]",
```

```
                    "Synchronizing.  [======= ]",
                    "Synchronizing.  [========]",
                    "Synchronizing.. [ =======]",
                    "Synchronizing.. [  ======]",
                    "Synchronizing.. [   =====]",
                    "Synchronizing.. [    ====]",
                    "Synchronizing...[     ===]",
                    "Synchronizing...[      ==]",
                    "Synchronizing...[       =]",
                    "Synchronizing...[        ]",
                    "Synchronizing   [        ]"]
        self.i = 0
        self.timer = None

    def update(self):
        print(self.animation[self.i % len(self.animation)], end='\r')
        self.i += 1

if __name__ == "__main__":
    CHANNEL = sys.argv[1]
    tpmclient = TPMClient()
    animation = Animation()
    # sio.connect('https://tpmserver.herokuapp.com/')
    sio.connect('http://127.0.0.1:5000')
```

Б.3 Лістинг файлу «session.html»

```
<!DOCTYPE html>
<html lang="ua">
<head>
        <meta http-equiv="Content-TYPE" content="text/html" charset="utf-8">
        <meta name="aythor" content="Рябко В.І.">
        <meta name="copyright" content="Всі права належать Рябко В.І.">
        <meta name="description" content="Виконання кваліфікаційної роботи">
    <title>Шифрований канал зв'язку</title>
</head>
<body>
    <div style='overflow: scroll;border: 4px solid black;margin: 1%;width: 1850px;height: 700px;padding-left: 5px; border-radius: 10px;'>
        <h3 style='color: #ccc;font-size: 30px;'>Повідомлень ще нема...</h3>
        <div class="message_holder"></div>
        </div>

    <form style='padding-left: 30px;' action="" method="POST">
          <b style='font-size: 20px;'>Вкажіть шлях до ключу шифрування... </b><input type="file" onchange="loadFile(this.files[0])">
        <p><div style='float: left;padding-right: 10px;padding-left: 700px;'><input type="text" class="username" placeholder="Ім'я користувача..."/></div>
        <div style='float: left;padding-right: 10px;'><input style='width: 850px; height: 100px;' type="text" class="message" placeholder="Введіть текст повідомлення..."/></div>
        <input type="submit"/></p>
    </form>
```

```
        <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
        <script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
        <script
src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/1.7.3/socket.io.min.js"></script>
            <script src="https://cdnjs.cloudflare.com/ajax/libs/crypto-js/3.1.9-1/crypto-
js.js"></script>
            <script src="https://cdnjs.cloudflare.com/ajax/libs/crypto-
js/3.1.2/rollups/aes.js"></script>
        <script type="text/javascript"></script>
            <script>
         var socket = io.connect('http://' + document.domain + ':' + location.port);
              var key

              async function loadFile(file) {
          let text = await file.text()
                      key = text
         }
         socket.on( 'connect', function() {
          socket.emit( 'my event', {
           data: 'User Connected'
          } )
          var form = $( 'form' ).on( 'submit', function( e ) {
           e.preventDefault()
           let user_name = $( 'input.username' ).val()
           let user_input = $( 'input.message' ).val()
                       var encryptedAES = CryptoJS.AES.encrypt(user_input, key);
                       encryptedAES = encryptedAES.toString();
           socket.emit( 'my event', {
            user_name : user_name,
            message : encryptedAES
           } )
           $( 'input.message' ).val( '' ).focus()
          } )
         } )
         socket.on( 'my response', function( msg ) {
                      console.log( msg )
                      let encryptedmsg=msg.message
          console.log( key );
          if( typeof msg.user_name !== 'undefined' ) {
           $( 'h3' ).remove()
           $( 'div.message_holder' ).append( '<div><b style="color: #000">Користувач:
</b><b style="color: #FF0000">'+msg.user_name+'</b><b style="color: #000"> Надіслав:
</b><b style="color: #00FF00">'+CryptoJS.AES.decrypt(encryptedmsg,
key).toString(CryptoJS.enc.Utf8)+'</b></div>' )
          }
         })
        </script>
       </body>
       </html>
```

Додаток В

Роздатковий матеріал

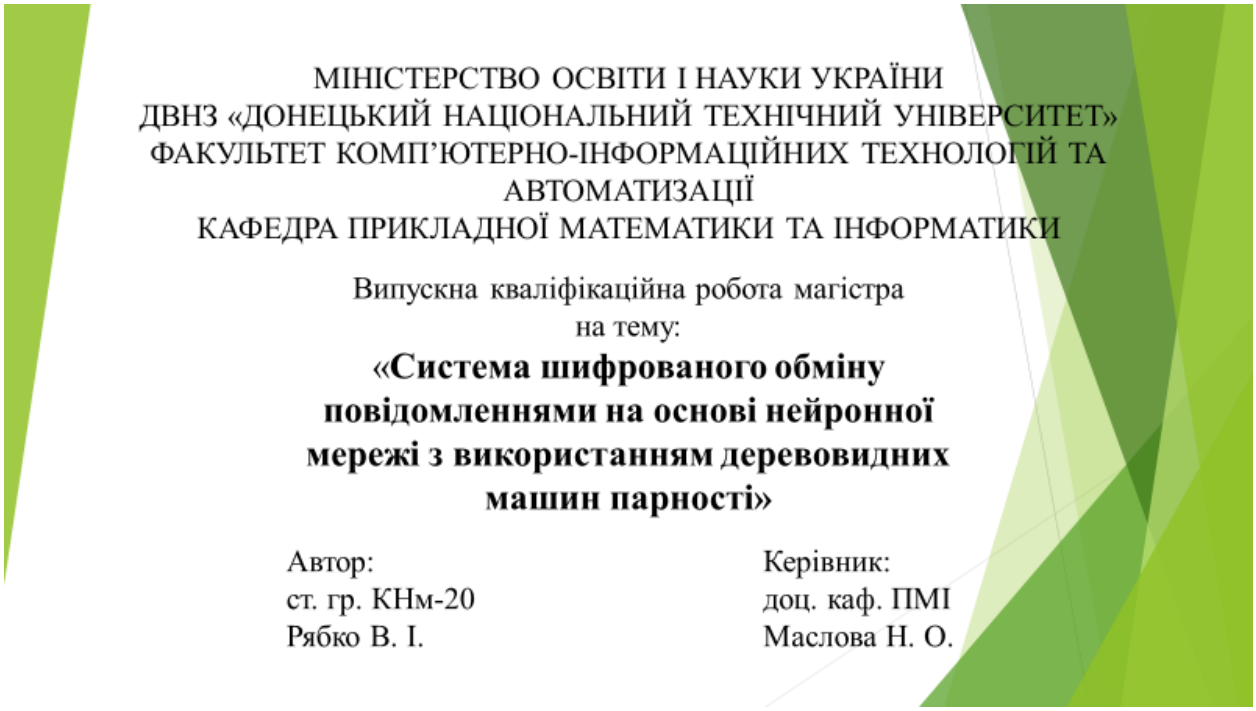


Рисунок В.1 – Перший аркуш презентації

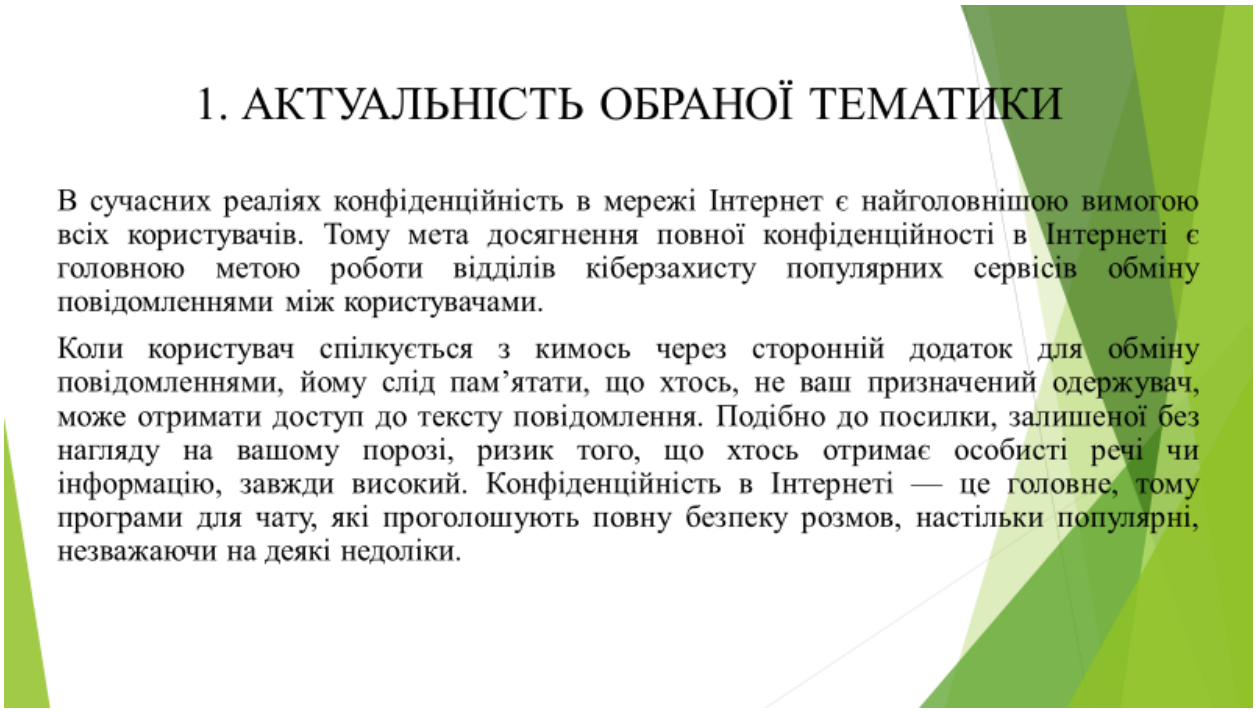


Рисунок В.2 – Другий аркуш презентації

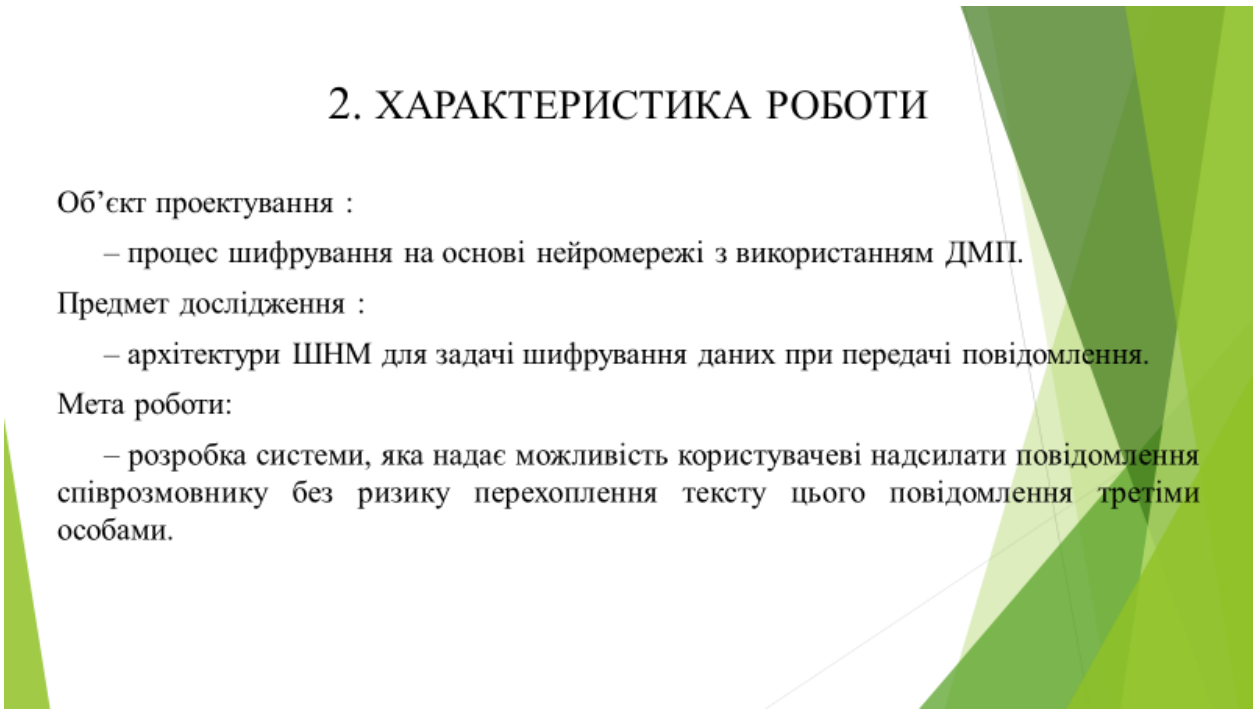


Рисунок В.3 – Третій аркуш презентації

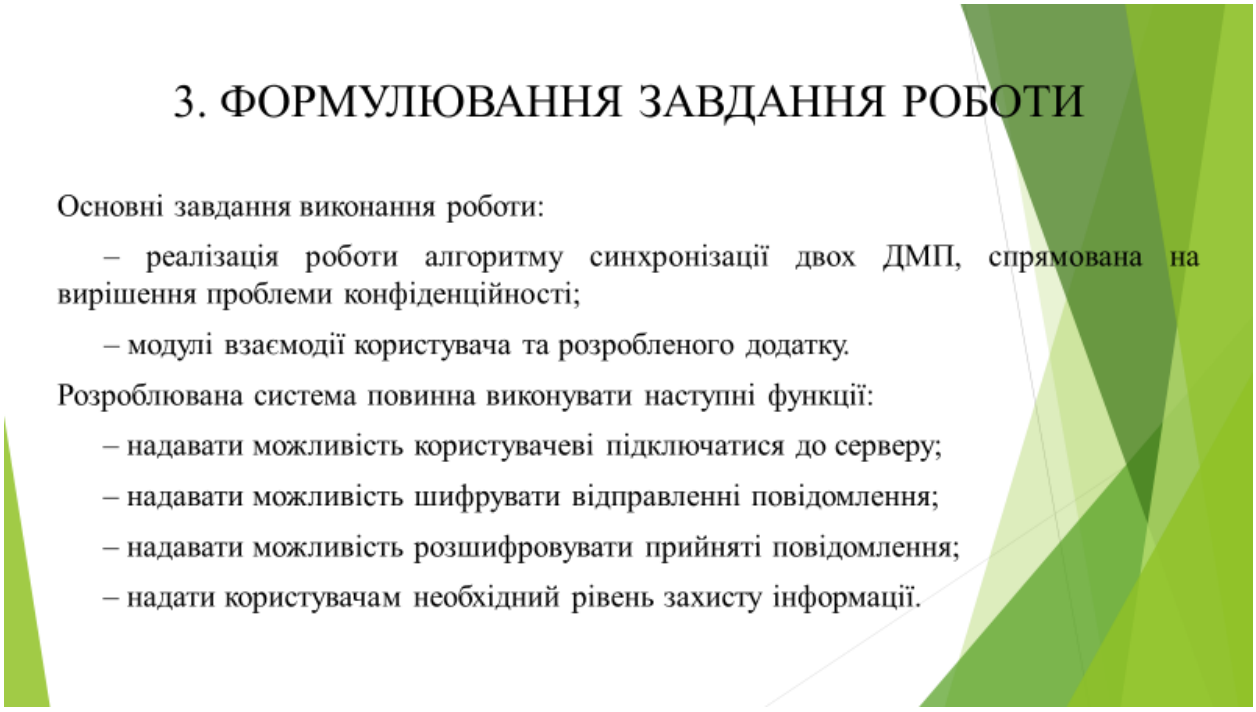


Рисунок В.4 – Четвертий аркуш презентації

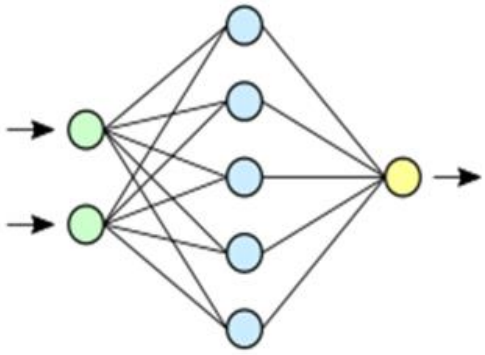
4. АНАЛІЗ ПРОБЛЕМИ КОНФІДЕНЦІЙНОСТІ В МЕРЕЖІ ІНТЕРНЕТ

Розуміння того, що деякі повідомлення можуть бути прочитані, обмежує, хто може використовувати деякі продукти, якщо їх конфіденційність та безпека є слабкими. Тому шифрування надзвичайно важливе. Шифрування або навіть припущення про те, що шифрування існує, дозволяє мовленню вільно передаватись, не турбуючись про те, що ці повідомлення будуть перехоплені, прочитані та використані не в тих цілях, які передбачав відправник.

Користувачі також повинні знати, що якщо вміст їхніх повідомлень захищено наскрізним шифруванням і заблокований від перехоплення компанією для обміну повідомленнями або правоохоронними органами, це не означає, що ризиків немає. Інформація про наскрізний зашифрований зв'язок може все ще описувати метадані зв'язку, такі як платформа, яка використовується для надсилання повідомлення, час надсилання та отримання повідомлення, ідентифікація відправника та одержувача, а також кількість надісланих та отриманих даних повідомлення.

Рисунок В.5 – П'ятий аркуш презентації

5. ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ ДАНИХ ЗА ДОПОМОГОЮ ШНМ (Ч.1)



Модель штучного нейрону

Рисунок В.6 – Шостий аркуш презентації

5. ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ ДАНИХ ЗА ДОПОМОГОЮ ШНМ (Ч.2)

Традиційно системи виявлення та запобігання вторгненням використовували алгоритми машинного навчання або виявлення на основі сигнатур для моніторингу активності мережі та запобігання вторгненням.

Деякі нові рішення СВВ та СЗВ почали використовувати технологію нейронних мереж, включаючи глибоке навчання, згорткові нейронні мережі та рекурентні нейронні мережі, щоб аналізувати мережевий трафік з більшою точністю та усувати інші обмеження традиційних систем. ШНМ краще розпізнають закономірності та ідентифікують порушення, навіть якщо вони не дотримуються встановлених векторів атаки, а ШНМ можуть автоматично вирішувати багато проблем без взаємодії з людиною.

Рисунок В.7 – Сьомий аркуш презентації

6. НЕДОЛІКИ ВИКОРИСТАННЯ НЕЙРОННИХ МЕРЕЖ ДЛЯ ЗАХИСТУ ДАНИХ

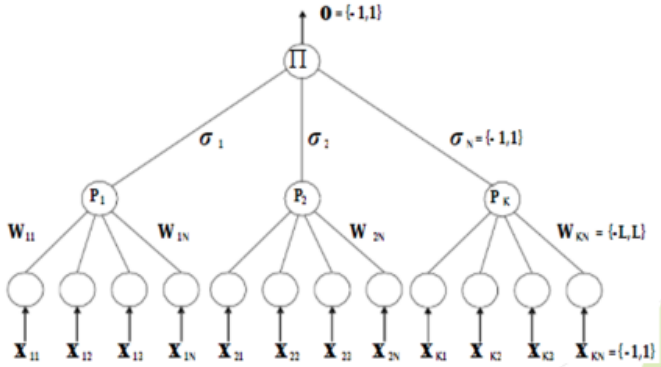
Треба бути дуже обережними під час навчання нейронної мережі, щоб уникнути зміщення моделі. По суті, за допомогою моделювання та зворотного зв’язку, який надається ШНМ на етапі навчання, можна ненавмисно навчити її отримувати результати, які ви хочете отримати, а не точні результати.

Інша проблема з нейронними мережами називається проблемою «чорного ящика». В основному, як тільки ваша ШНМ починає давати результати, може бути дуже важко, якщо взагалі неможливо, визначити, як вона досягла цих результатів. Алгоритми, що використовуються в розширеному машинному навчанні, стають настільки складними, що навіть інженери, які їх створюють, не повністю розуміють, як вони працюють або чому поводяться певним чином.

Рисунок В.8 – Восьмий аркуш презентації

7. ПРОЦЕС ШИФРУВАННЯ НА ОСНОВІ ДЕРЕВОВИДНИХ МАШИН ПАРНОСТІ (Ч.1)

Одним з найпоширеніших криптографічних методів, пов'язаних зі штучним інтелектом у нейронних мережах, є ДМП.



Архітектура ДМП

Рисунок В.9 – Дев'ятий аркуш презентації

7. ПРОЦЕС ШИФРУВАННЯ НА ОСНОВІ ДЕРЕВОВИДНИХ МАШИН ПАРНОСТІ (Ч.2)

ДМП складається з вхідних бітів x, вагових векторів w (w – значення синаптичної ваги) та вихідного біта o. Крім того, ДМП складається з K прихованих нейронів і N входів для кожного прихованого нейрона. Значення синаптичних ваг w генеруються як випадкові дискретні значення L. Отже, w_{kj} може брати числа з $-L, -L+1, \ldots, L$ [10]. Передбачається, що K, N, L і w є секретними для зловмисника, який намагається перехопити зашифроване повідомлення з загальнодоступного каналу. Наступне припущення полягає в тому, що K, N, L мають бути однаковими для обох ДМП, які беруть участь у процесі синхронізації.

Рисунок В.10 – Десятий аркуш презентації

8. ПРОЦЕС АТАКИ НА СИСТЕМУ СИНХРОНІЗАЦІЇ (Ч.1)

Для перевірки безпеки процедури синхронізації ДМП було використано три методи атаки на систему. Перший – це атака грубою силою, другий – генетичний алгоритм для отримання ваг від кінцевого значення вихідного нейрону та третій – це атака за допомогою власної ДМП.

Атака методом грубої сили.

Щоб здійснити атаку грубою силою, зловмисник повинен перевірити всі можливі ключі (усі можливі значення ваг w_{kj}). За K прихованих нейронів, $K \cdot N$ вхідних нейронів та межі ваг L, це дає $(2L+1)^{KN}$ можливостей. Наприклад, конфігурація $K = 3$, $L = 100$ і $N = 3$ дає нам $(201)^9$ ключових можливостей, що унеможливлює атаку при сучасних потужностях комп'ютера.

Рисунок В.11 – Одинадцятий аркуш презентації

8. ПРОЦЕС АТАКИ НА СИСТЕМУ СИНХРОНІЗАЦІЇ (Ч.2)

Атака, використовуючи генетичні алгоритми.

Було проведено 3 види експериментів, але ніякий не приніс успішний результат. Перший експеримент був для пошуку всіх 12 параметрів вагових коефіцієнтів. Не було жодного успіху навіть після 200 000 поколінь. Тест було проведено багато разів, але прийнятного результату не було знайдено. Другий експеримент був для пошуку лише першого та другого рядків з матриці, що означає лише 8 параметрів вагових коефіцієнтів. Знову ж таки, успіху не було навіть після 200 000 поколінь. Третій експеримент передбачав пошук лише в одному рядку з матриці. Отже, необхідно було знайти лише 4 параметри вагових коефіцієнтів. Цей експеримент також невдалий. Прийнятний результат може бути знайдений через значну кількість поколінь, а час, необхідний для отримання результату, буде не в рамках розумного.

Рисунок В.12 – Дванадцятий аркуш презентації

8. ПРОЦЕС АТАКИ НА СИСТЕМУ СИНХРОНІЗАЦІЇ (Ч.3)

Атака з використанням власної ДМП.

Після збільшення кількості прихованих нейронів було проаналізовано графік відношення невдалих спроб на синхронізацію ДМП зловмисника до кількості прихованих нейронів.

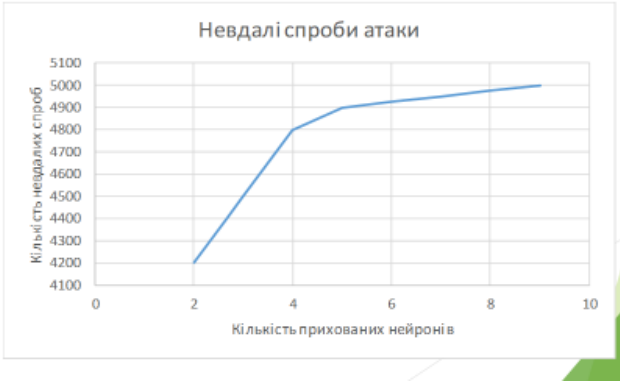


Рисунок В.13 – Тринадцятий аркуш презентації

8. ПРОЦЕС АТАКИ НА СИСТЕМУ СИНХРОНІЗАЦІЇ (Ч.4)

Далі розглянемо процес атаки до та після синхронізації абонентів A і B.

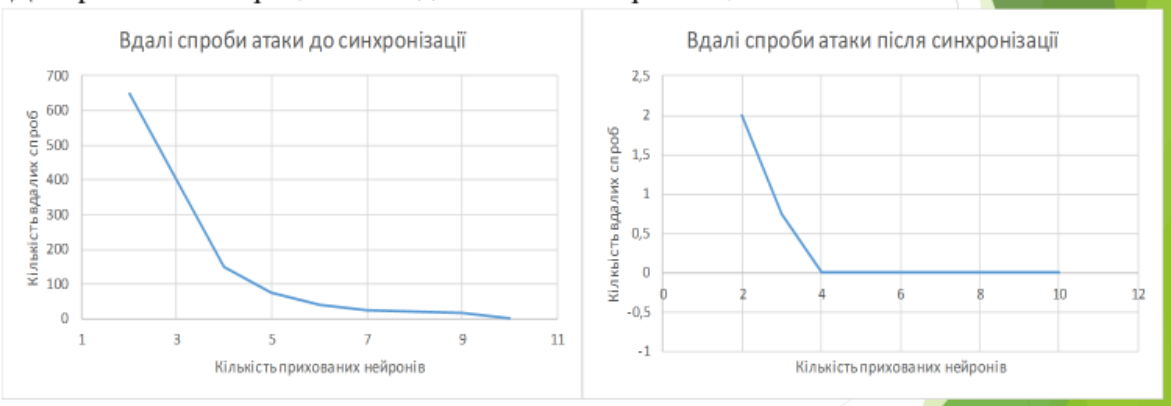


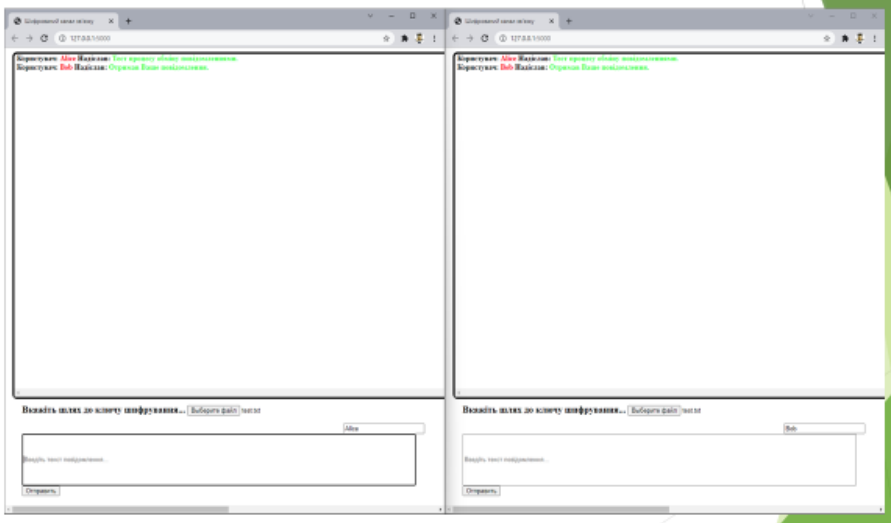
Рисунок В.14 – Чотирнадцятий аркуш презентації

8. ПРОЦЕС АТАКИ НА СИСТЕМУ СИНХРОНІЗАЦІЇ (Ч.5)

Проаналізувавши всі результати атак можна зробити висновки, що навіть при збільшенні кількості прихованих нейронів та кількості вхідних нейронів до 4 можна спостерігати значне зменшення кількості вдалих спроб на втручання ДМП зловмисника.

Рисунок В.15 – П’ятнадцятий аркуш презентації

9. ЕКРАННІ ФОРМИ



Процес обміну повідомленнями

Рисунок В.16 – Шістнадцятий аркуш презентації

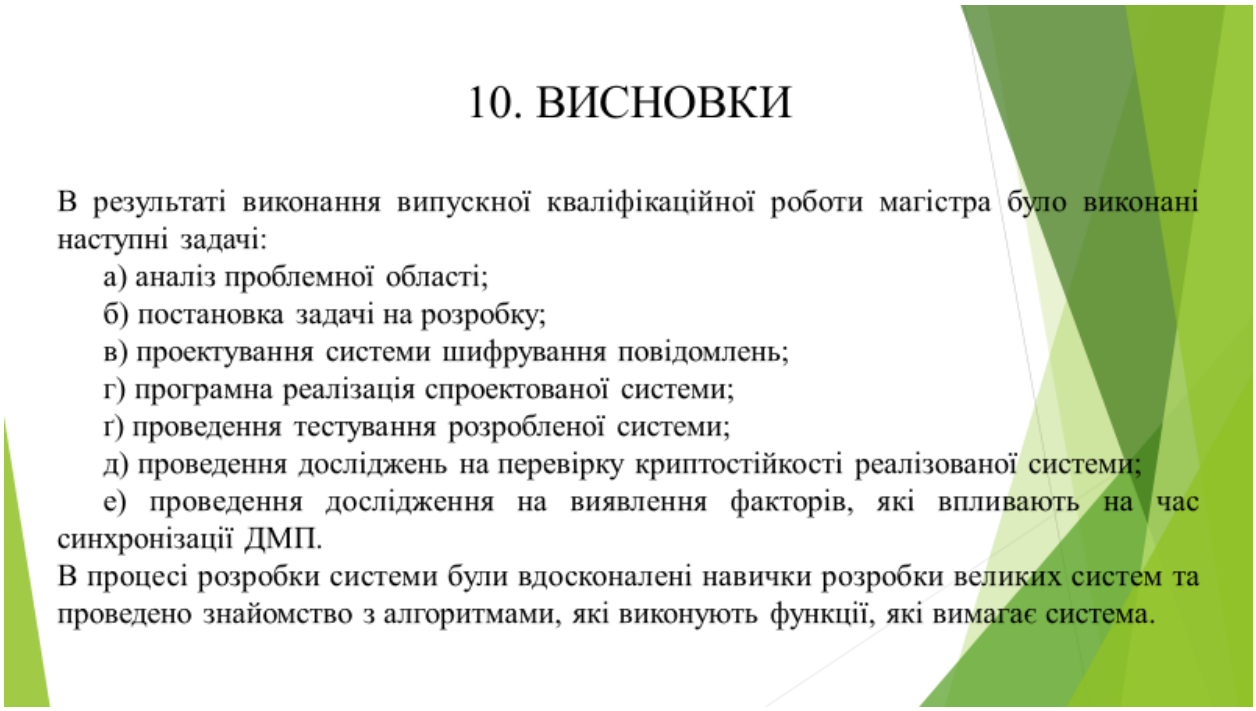


Рисунок В.17 – Сімнадцятий аркуш презентації

Рисунок В.18 – Вісімнадцятий аркуш презентації