МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Інформаційна система нейрокриптографічного шифрування повідомлень»

Завідувач
випускаючої кафедри **Довбиш А. С.**

Керівник роботи **Москаленко В. В.**

Студента групи КБ-71 **Кузьменка А.В.**

СУМИ 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп’ютерних наук

Затверджую___________

Зав. кафедрою Довбиш А.С.

“_____” ____________2021 г.

ЗАВДАННЯ

до випускної роботи

Студента п’ятого курсу, групи КБ-71 спеціальності “Кібербезпека” денної форми навчання Кузьменка Андрія Володимировича.

Тема: “ **Інформаційна система нейрокриптографічного шифрування повідомлень** ”

Затверджена наказом по СумДУ

№________ от ___________2021 г.

Зміст пояснювальної записки: 1) аналіз проблеми та постановка задачі; 2) інформаційна система криптозахисту повідомлень; 3) реалізація інформаційної системи.

Дата видачі завдання “_____”____________2021 г.

Керівник випускної роботи __________ Москаленко В.В.

Завдання приняв до виконання ____________ Кузьменко А.В.

РЕФЕРАТ

Записка: 47 стор., 21 рис., 1 додаток, 7 джерел.

Об'єкт дослідження — інформаційна система нейрокриптографічного шифрування повідомлень.

Мета роботи — розробити моделі та методи інформаційної технології нейрокриптографічного шифрування повідомлень. Модель повинна вміти шифрувати та дешифрувати повідомлення, а також бути криптостійкою до атак.

Результати — проаналізовано та розроблено модель для нейрокриптографічного шифрування повідомлень. Для шифрування використовується текст та ключ для перетворення у шифротекст, для перетворення у тексти використовується шифротекст та ключ. Було розроблено програмну реалізацію, яка виконує нейрокриптографічне шифрування.

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ, МАШИННЕ НАВЧАННЯ, ШИФРУВАННЯ, МОДЕЛІ МАШИННОГО НАВЧАННЯ, RSA, DIFFIE-HELLMAN, PYTHON, TENSORFLOW.

ЗМІСТ

ВСТУП 5

1. АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧІ 6

1.1. Сучасний стан та тенденції розвитку систем криптографічного захисту інформації 6

1.2. Нейромережеві методи в кібербезпеці 13

1.3. Постановка задачі 20

2. ІНФОРМАЦІЙНА СИСТЕМА КРИПТОЗАХИСТУ ПОВІДОМЛЕНЬ 21

2.1. Модель нейромережі для шифрування та дешифрування повідомлень 21

2.2. Метод навчання нейромережі 27

3. РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ 31

3.1. Короткий опис програмного забезпечення 31

3.2. Результати навчання 35

ВИСНОВКИ 39

СПИСОК ЛІТЕРАТУРИ 40

ДОДАТКИ 42

Додаток А 42

ВСТУП

Сучасні алгоритми криптографії базуються на теорії чисел. Майже всі методи створення ключів достатньої довжини для безпечного шифрування звертаються або до генерації псевдовипадкових послідовностей, або до розширення простих чисел. Зокрема, прості числа в більшості випадків використовуються найбільш популярним методом шифрування відкритим ключем RSA. На сьогоднішній день цей алгоритм криптозахищений, не зазнає взлому, незважаючи на збільшення швидкості обчислювального обладнання та появу нових класів атак, що використовують апаратні особливості комп’ютерів, що генерують ключі.

Однак головна небезпека для сучасної криптографії не виходить із сфери нових методів криптографічних атак, а з іншої сфери, яка зараз має експериментальний характер, що незабаром може стати причиною нової науково-технічної революції. Це квантові обчислення. Звичайно, ми не говоримо про квантовий комп'ютер, але повідомлення про створення перших запрограмованих комп'ютерів такого типу є в Колорадо (Colin, 2009) та в IBM Research, (2012). Дані квантові обчислення є проблемою для шифрування, оскільки вони можуть зруйнувати математичне підгрунтя криптографічних методів. Тому з’являється необхідність в нових методах шифрування, котрі не піддаються новому виду атак.

Головним завданням цієї роботи є аналіз і існуючих методів нейрокриптографічного шифрування, а також розробка алгоритму нейромережевого криптографічного шифрування.

1. АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Сучасний стан та тенденції розвитку систем криптографічного захисту інформації

Потреба в захищеності повідомлень сягає глибокої давнини. Щоразу ми спостерігаємо за п’ять тисяч-річними друкарськими циліндрами Месопотамії, котрі були використані для аутентифікації або шифру Цезаря, який використовувався для захисту військових повідомлень, ми розуміємо – інформація повинна бути захищеною[1].

На зараз є дві окремі парадигми забезпечення захищеності інформації. Одна з них – це симетричне шифрування, або шифрування з приватним ключем, котре може бути розглянуте, як перші спроби шифрування. Будь яка система з таким шифруванням, передбачає що якщо дві особи хочуть безпечно спілкуватись, вони повинні мати секретний ключ, котрий буде використовуватися для шифрування та дешифрування. Хоча багато таких систем еволюціонували з часом, саме ці умови та складності передачі та розповсюдження ключа та зберігання його в таємниці призводить до їхньої вразливості та необхідності нової парадигми. На початку 1970-х років почав формуватися значно новий і інший погляд на безпеку, що сприяло появі криптографії з відкритими ключами.

Криптосистеми з публічними ключами поділяються на дві категорії: протокол Діффі-Хелмана, опублікований Вайтфілдом Діффі (Whitfield Diffie) та Мартіном Хелманом (Martin Hellman) в 1976р., та RSA протокол, публічно описаний Роном Рівестом, Аді Шаміром та Леонардо Адлеманом в 1978р. Перш ніж зосередитись на структурі цих різних підходів, варто розглянути ширший вплив методів з публічним ключем.

Інтернет не завжди був таким розповсюдженим, тому лише в 90-х роках були запроваджені необхідні протоколи безпеки, які дозволяли користувачам безпечно передавати данні в мережі інтернет. Наприклад, онлайн-революція у фінансовому секторі, електронна комерція та медичні записи - все

покладається на прорив у галузі безпеки, який забезпечує криптографія з відкритим ключем. Багато в чому ми знаходимося в подібній еволюційній точці із вбудованими системами, бездротовими сенсорними мережами та технологіями радіочастотної ідентифікації (RFID). Рішення, в яких було створено Інтернет-революцію, не підходять для цих малоресурсних середовищ; необхідно створити нове покоління інфраструктури із відкритим ключем. Основна тема, яка бере початок із ранньої цивілізації, залишилась: інформація повинна бути захищена.

При схематичному перегляді на високому рівні всі способи шифрування із закритим ключем набувають наступної форми. Два користувачі, Аліса та Боб, мають один і той же секретний ключ, k, який використовується як для шифрування, так і для дешифрування (звідси термін симетричний). Коли Аліса хоче безпечно спілкуватися з Бобом, вона вводить своє повідомлення (яке часто називають відкритим текстом) разом зі своїм ключем k у свій протокол шифрування, який потім виводить повідомлення у зашифрованому вигляді, що називається зашифрованим текстом (рис. 1.1.1).

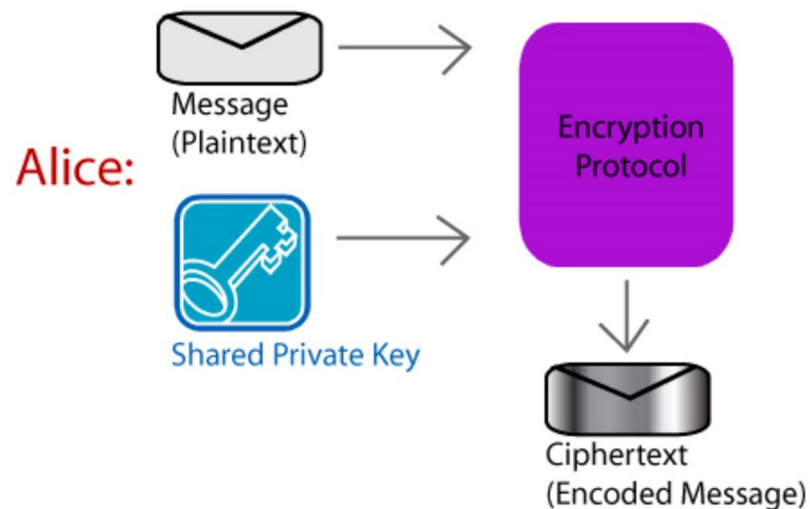

Рисунок 1.1.1 – Аліса створює зашифроване повідомлення з приватним ключем

Отримавши зашифрований текст від Аліси, Боб може отримати своє оригінальне повідомлення, ввівши їх спільний ключ k разом із зашифрованим текстом у свій протокол розшифрування, як показано на рисунку 1.1.2.

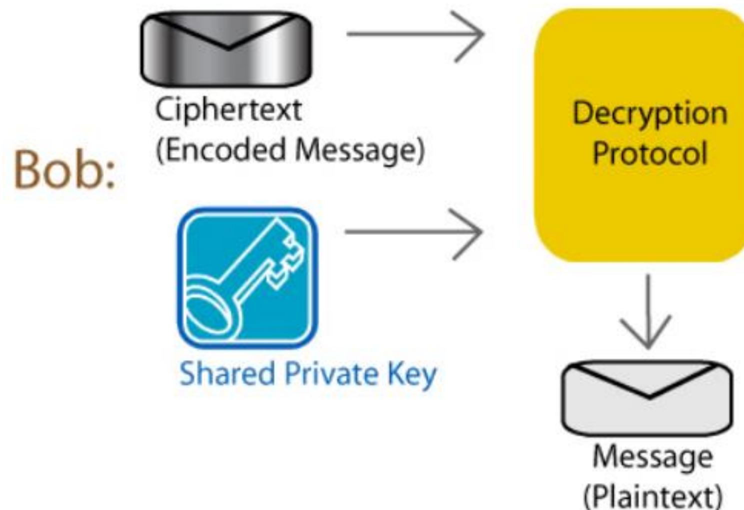

Рисунок 1.1.2 – Боб розшифровує повідомлення від Аліси в текст, використовуючи приватний ключ

Настільки ж інтуїтивно зрозумілим як такий метод з закритим ключем, є і вразливості. Безпека системи покладається на те, що секретний ключ k постійно залишається не скомпрометованим. Оскільки існує безліч криптографічних методів пошуку секретного ключа, ключ повинен залишатися захищеним найдовше. З використанням рішення із закритим ключем, якщо захищеність ключа одного користувача порушено, то це вплине на всіх користувачів, які використовують цей ключ[1].

По мірі зростання кількості користувачів системи приватних ключів (у випадку з RFID - на мільярди!), Необхідність регулярного конфіденційного розповсюдження нових ключів призводить до поступового нагромадження управління ключами. Хоча загалом вірно, що системи з приватними ключами мають швидкий час роботи, що увічнює їх використання у багатьох комерційних системах сьогодні, їх внутрішні слабкості є розповсюдженими.

Криптографія з відкритим ключем (або асиметрична) звільняється від більшості вищезгаданих проблем, присвоюючи кожному користувачеві

певний приватний ключ, який відомий лише користувачеві, та відкритий математичний ключ, який можна зробити відкритим та вільно розповсюджувати. Є два основні методи з використанням криптографії з відкритим ключем, RSA і Діффі-Хеллмана. В обох випадках системи спроектовані таким чином, що відкриті ключі не повинні розподілятися між усіма сторонами до спілкування, а безпека системи не порушується публікацією відкритих ключів на початку сеансу.

У випадку з системою типу RSA, якщо Аліса хоче надіслати Бобу повідомлення, вона використовує відкритий ключ Боба для шифрування її відкритого тексту (рис. 1.1.3).

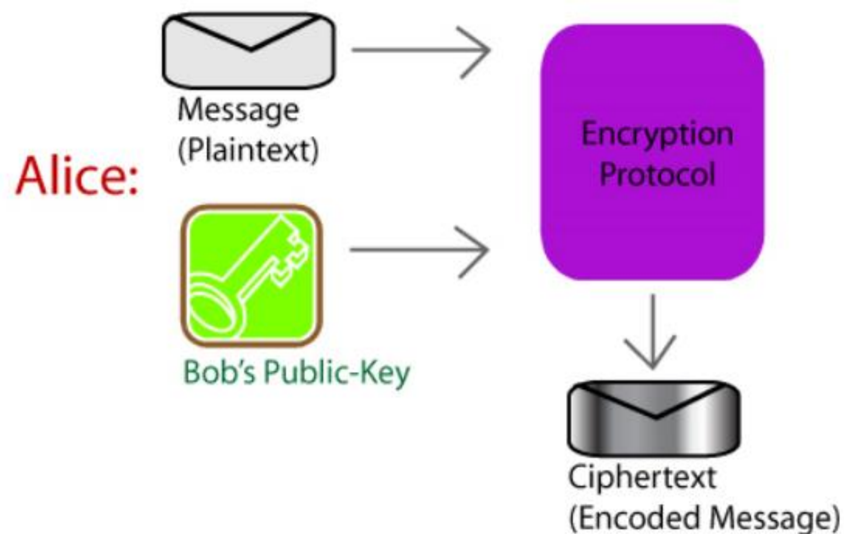

Рисунок 1.1.3 – Система типу RSA: Аліса використовує відкритий ключ Боба для шифрування свого повідомлення Бобу

Отримавши зашифрований текст, Боб може отримати оригінальне повідомлення Аліси, ввівши свій приватний ключ к разом із зашифрованим текстом у свій протокол розшифрування (рис. 1.1.4).

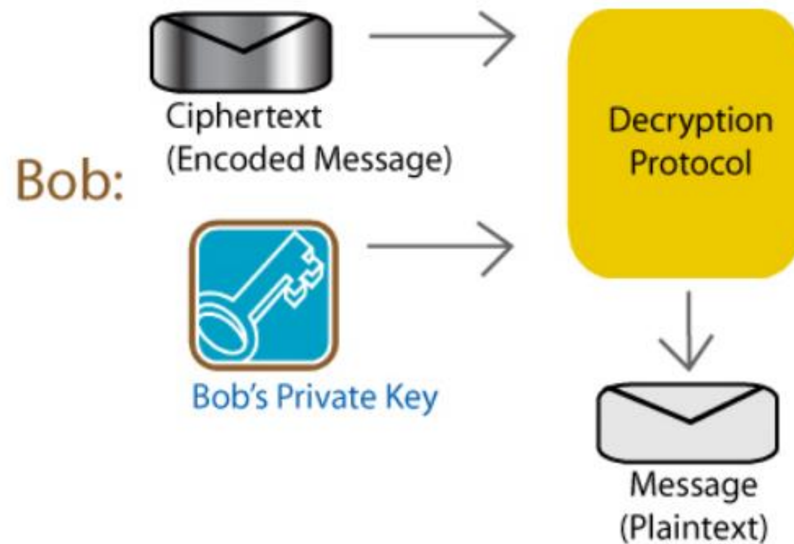

Рисунок 1.1.4 – Система типу RSA: Боб використовує власний приватний ключ для дешифрування закодованого повідомлення

У системі типу RSA лише Боб має копію свого приватного ключа, тому кожен, хто перехоплює зашифроване повідомлення від Аліси, не зможе його розшифрувати.

У випадку з системою типу Діффі-Хеллмана Аліса та Боб використовують відкриті ключі один одного разом із власними відповідними приватними ключами, щоб встановити загальний секретний ключ, який можна використовувати для шифрування та дешифрування їх повідомлення. Лише Аліса та Боб, які беруть участь у цій сесії, матимуть доступ до необхідного ключа для шифрування або дешифрування повідомлення[2].

Системи типу Діффі-Хеллмана структурно відрізняються (рис. 1.1.5).

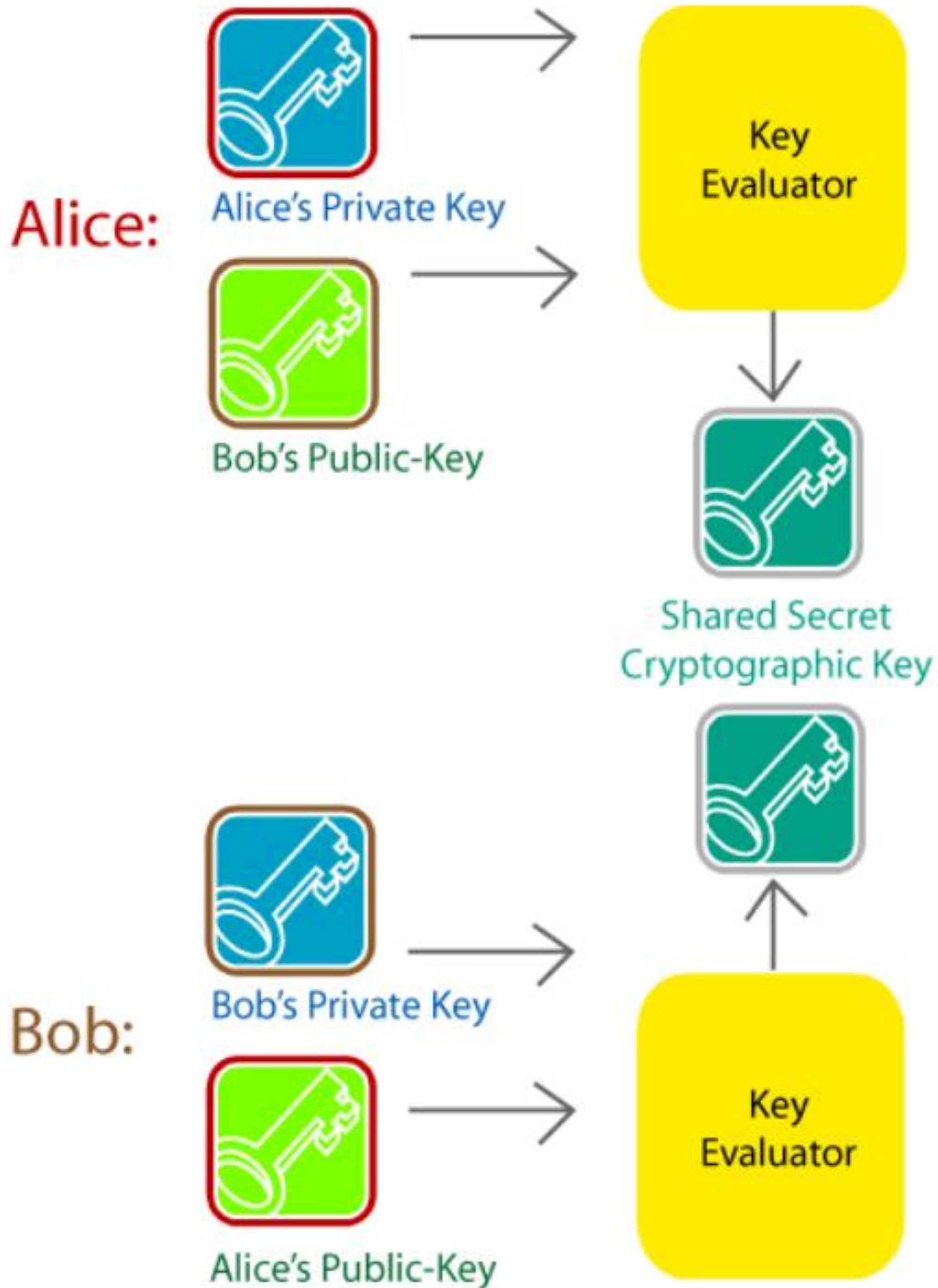

Рисунок 1.1.5 – Система типу Діффі-Хеллмана

Результатом роботи системи типу Діффі-Хеллмана є спільний секретний криптографічний ключ, значення якого однакове як для Аліси, так і для Боба. Потім цей спільний ключ можна використовувати для шифрування та дешифрування переданого між ними повідомлення, подібно до системи приватних ключів. Для порівняння результатом роботи системи типу RSA є

сам зашифрований текст. В обох цих асиметричних системах приватний ключ кожного користувача відрізняється від приватного ключа іншого користувача. Стислий вигляд різних систем відкритого та закритого ключів є у дереві концепцій (рис. 1.1.6).

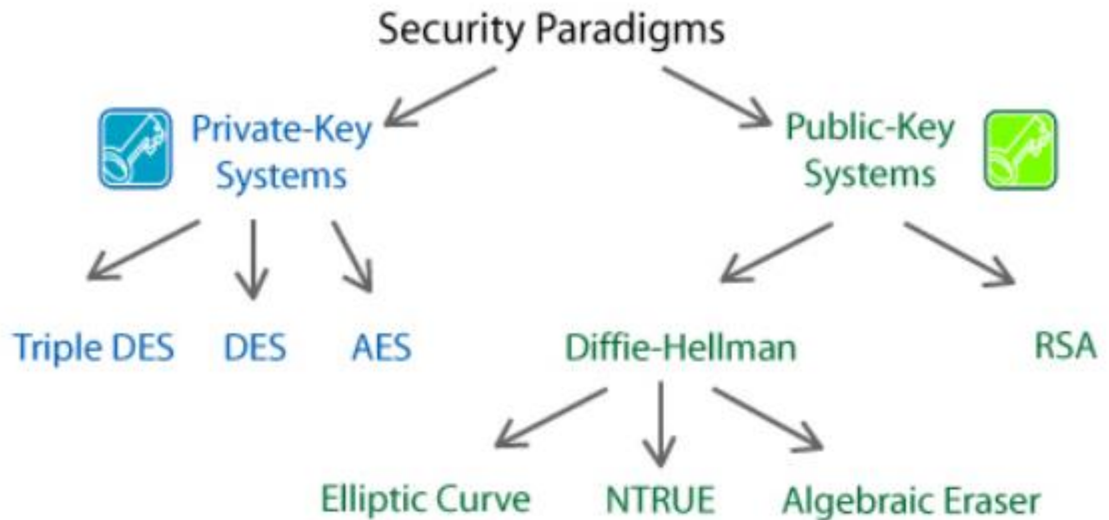

Рисунок 1.1.7 – Категоризація парадигм безпеки закритого та відкритого ключів

У ситуації із закритим ключем безпека системи покладається на те, що секретний ключ k постійно залишається у таємниці. На відміну від цього, із системою відкритих ключів, якщо приватний ключ будь-якого користувача стає відомим, інші користувачі залишаються захищеними від компрометації їх комунікацій. Це пов’язано з тим, що приватний ключ не є загальним у всій системі, як у симетричній системі, а навпаки, приватний ключ унікальний для кожної людини. У сценарії відкритого ключа математичний взаємозв'язок між приватним та відкритим ключем користувача розроблений таким чином, що неможливо отримати приватний ключ із відкритого ключа (такий зв'язок називається односторонньою функцією). Для того, щоб мати можливість ефективно здійснити зворотне отримання відкритого ключа, необхідно вирішити математично нерозв’язну проблему. Тобто математичні задачі, які можуть бути вирішуваними, але настільки складними або потребувати стільки

часу для завершення, що рішення не дасть жодної вигоди після отримання. Безпека цих систем полягає в цій нерозв'язності.

Системи відкритих ключів мають проблеми, які потрібно подолати. Забезпечення автентичності розподілених відкритих ключів має важливе значення для безпеки (відоме як атака "людина посередині"). Потрібно також запровадити протоколи для запобігання зловмисному використанню публічних даних (запобігання атакам повторного відтворення). Незважаючи на технічну складність, ці перешкоди можна пом'якшити.

Більш глибока проблема із загальновпровадженими сьогодні протоколами RSA та Diffie-Hellman типу відкритого ключа, включаючи NTRU та криптосистеми на еліптичних кривих (ECC), стосується обчислювального сліду, який вони створюють. Незважаючи на те, що використання пам’яті та енергії не є основною проблемою для більшості середовищ, що вимагають криптографічної безпеки, ці проблеми, поряд із часом роботи, лежать в основі будь-якої дискусії щодо безпеки невеликих обчислювальних пристроїв. Кожна з цих криптографічних систем у своїй основі використовує множення великих чисел. В результаті обчислювальні ресурси, необхідні для досягнення безпеки, швидко зростають із збільшенням рівня безпеки [2].

1.2. Нейромережеві методи в кібербезпеці

Одним з небагатьох постквантових підходів у криптографії з перспективою набуття широкого поширення є використання нейронних мереж. Раніше вони використовувались для створення нових типів атак на існуючі коди, засновані на ідеї, що будь-яка функція може бути представлена як нейронна мережа, яка може досліджувати простір рішення. Це дозволяє вирішити багато проблем, пов'язаних з криптографією, зокрема, хешування та генерації псевдовипадкових масивів. Запропонована в 1995 році Себастьеном Дурленсом нейронна криптографія заснована на застосуванні нейронних мереж для вирішення таких проблем. Цей французький математик

використовував нейронні мережі для перестановок в DES-алгоритмі, демонструючи таким чином застосовність фундаментальної нейронної мережі в області шифрування (Sébastien Dourlens Neuro-applied Cryptography, 1995-1996) [3]. З тих пір було проведено більше досліджень, включаючи протокол криптографії з відкритим ключем, розроблений Халілом (2012). У цій роботі зроблена спроба схрестити звичну систему створення відкритих та приватних ключів з алгоритмами нейронної мережі. Генерація приватного ключа за допомогою цього підходу була зроблена звичайними методами, а відкритий ключ представив нейронну мережу, яка отримує можливість розшифрувати повідомлення навчаючись шляхом зворотного розповсюдження. Однак цей метод має недолік - для великих обсягів даних час навчання нейронних мереж різко збільшується.

Це не єдині приклади застосування нейронних мереж у криптографії: існує багато робіт, включаючи оригінал - наприклад, роботи тайванських вчених (Tai-Wen and Suchen, 2001) пропонують використовувати нейромережу Хопфілда для шифрування. Однак, на жаль, вони в основному є теоретичними, і все ж мають обмежене практичне використання.

Криптографічний протокол, заснований на нейронних мережах: щоб дати уявлення про те, наскільки корисними є нейронні мережі, розглянемо приклад, описаний Оскаром та Карл Хайнцом, 2010). Вони описують цікаву зміну алгоритму Діффі-Хеллмана, який використовується для обміну двома ключами. Модель Рейеса-Циммермана складається з двох деревовидних машин парності (ДМП), які синхронізуються незалежно, і таким чином вони отримують ключ. Механізм синхронізації побудований подібно до синхронізації двох хаотичних осциляторів (це математичний опис дуже складний і виходить за межі даного дослідження). Отже, припустимо, що існує якась ДМП, який містить три рівні, як показано на рис. 1.2.1.

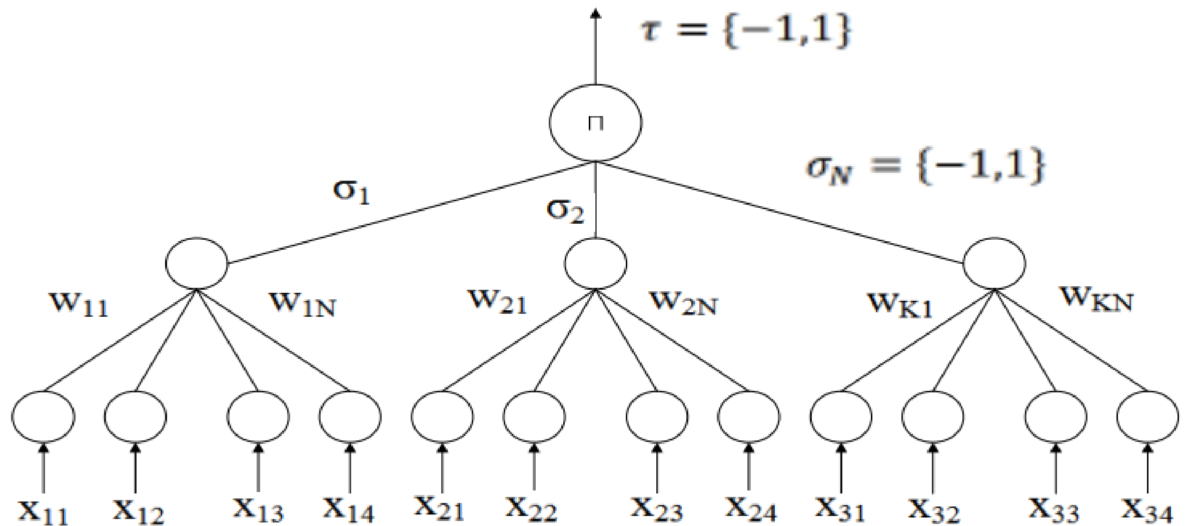

Рисунок 1.2.1 – Схема деревовидної машини парності

Як видно з діаграми, це нейронна мережа прямого розподілу і вона містить три рівні. Вхідний рівень складається з *K*N* двійкових нейронів, які описуються формулою:

$$x_{ij} \in \{-1, 1\}$$

Між вихідним нейроном і масивом вхідних нейронів є так званий "прихований" рівень, який не є двійковим і має таку вагу:

$$w_{ij} = \{-L, \dots, 0, \dots, +L\}$$

Значення кожного із прихованих нейронів обчислюється як сума вхідних значень і ваг. Зверніть увагу, що знак обчислюється окремо, оскільки нуль прирівнюється до від'ємного.

$$\sigma_i = sign(\sum_{j=1}^{N} w_{ij} x_{ij})$$

$$sign(x) = \begin{cases} -1, x =< 0 \\ 1, x > 0 \end{cases}$$

Вихідний нейрон також має цілочисельний формат і є множенням усіх прихованих нейронів. Це двійковий:

$$\tau = \prod_{i=1}^{K} \sigma_i$$

А тепер уявіть, що кожна з двох сторін A і B має ДМП. Для їх синхронізації використовується двонаправлене навчання:

- ініціалізація вагових коефіцієнтів випадковими значеннями;
- формування випадкового вектора введення;
- обчислення вихідних значення нейронів прихованого шару;
- порівняння виходів ДМП: якщо виходи різні, то повторення ітерації.

З кроку 2. Якщо вони однакові, ми зберігаємо розраховану вагу.

Складемо блок-схему такого алгоритму навчання (враховуючи, що максимальна вага дорівнює L) (Рис. 1.2.2).

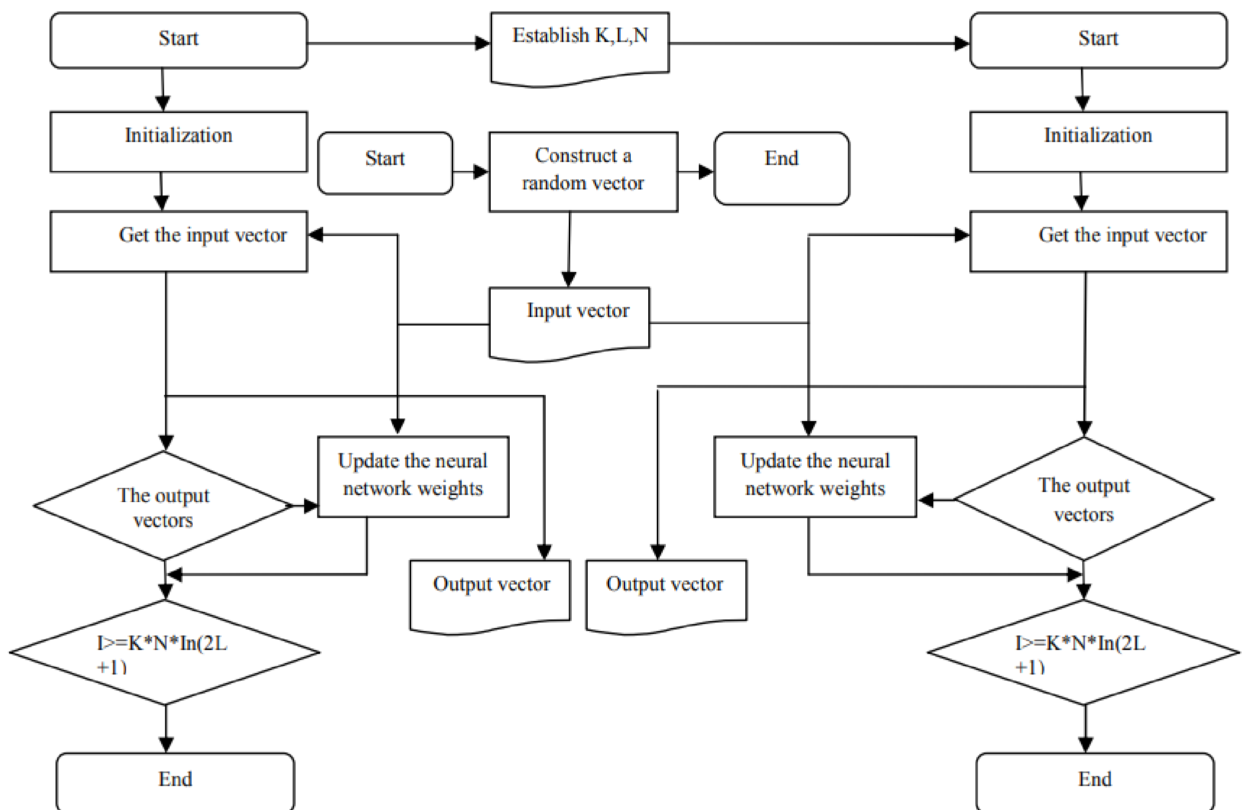

Рисунок 1.2.2 – Блок схема алгоритму навчання

Якщо досягнуто повної синхронізації, тобто ваги прихованих нейронів в обох ДМП ідентичні на відміну від обидвох частин що можуть використовувати ці ваги як ключ. Щоб оновити ваги в процесі або періодичну зміну ключа, ви можете скористатися одним із трьох варіантів [4]:

1. Правило навчання Геббіана

$$w_i^+ = w_i + \sigma_i x_i \Theta(\sigma_i \tau)\Theta(\tau^A \tau^B)$$

2. Правило навчання анти-Геббіана

$$w_i^+ = w_i - \sigma_i x_i \Theta(\sigma_i \tau)\Theta(\tau^A \tau^B)$$

3. Випадкове блукання

$$w_i^+ = w_i + x_i \Theta(\sigma_i \tau)\Theta(\tau^A \tau^B)$$

Таким чином, ми маємо відносно простий нейронний криптографічний протокол.

Атаки на нейрокриптографічний протокол: природно, що нейрокриптографічна система, як і будь-який кодовий протокол, піддається атакам. Давайте розглянемо деякі з них, припустивши, що певний криптоаналітик має доступ до повідомлень між співрозмовниками, але не може їх змінити. Припустимо також, що криптоаналітику повідомляється про те, що він має справу з нейрокриптографічним протоколом і навіть має у своєму розпорядженні ДМП [4].

«Фронтальна атака»: Для аналогічної атаки слід перевірити всі можливі комбінації клавіш. Кількість ключів для K прихованих нейронів, *K*N* вхідних нейронів та максимальна вага L становить:

$$S = (2 \cdot L + 1)^{K \cdot N}$$

Давайте обчислимо цю швидкість для нейрокриптографічної системи на рис. 1. Припустимо, що K = 3, L = 3, N = 10:

$$S = (2 \cdot 3 + 1)^{3 \cdot 10} = 7^{30} = 22539340290692258087863249$$

Як ми бачимо, кількість пошуку величезна навіть для найменших нейрокриптографічних систем. І якщо нейронна мережа складається щонайменше з 100 вхідних елементів, фронтальна атака на нейрокриптограму вимагатиме величезного комп’ютерного часу.

Тренування власних ДМП: Припустимо, що криптоаналітик має подібні ДМП, як обидва співрозмовники, і цей криптоаналітик читає їх листування.

Використовуючи шаблони диспетчеризації як початкові вектори, він може тренувати свою власну ДМП за допомогою алгоритму, який описаний у попередньому розділі, і постійно порівнюючи вихід зі своєї ДМП з даними співрозмовників. У цьому випадку виникають три ситуації (криптоаналітик відзначений буквою Е):

- A <> B. Ніхто не відновлює значення
- A = B = E. Усі три поновлюють значення
- A = B <> E. A і B поновлюють значення, тоді як E не може цього зробити

Залежно від частоти поновлення значущості (тобто періодичної зміни ключів) може бути емпірично показано, що криптоаналітик завжди поновлюватиме значення повільніше, ніж замовники. Тому він може ідентифікувати лише частину ключа, але не повністю. Ця атака може бути ефективною лише у випадку великої кількості проаналізованих повідомлень або коли співрозмовники взагалі не поновлюють значення свого ДМП.

Генетична атака: цей метод запропоновано Клімовим та співавт. (2002) і ефективний лише для малих нейронних мереж (N, K <= 3), оскільки для великих (K> 6) його ускладнення та інтенсивність використання збільшуються в геометричній прогресії. Однак це єдиний метод атаки, який може бути застосований до квантових розрахунків і стати серйозною небезпекою для нейрокриптографії.

Насправді це модифікація попереднього методу атаки, який базується на тренуванні ДМП його криптоаналітиком та поєднується з генетичними алгоритмами. Створюється певна різноманітність (яку можна порівняти з кількістю аналізованих повідомлень) нейронних мереж зі структурою ДМП. Після ітерації кожен із них випадково обмінюється своїм значенням із сусідами. Після цього ті TPM, які вийшли з ладу, скасовуються. Таким чином, у процесі криптоаналізу завжди будуть обрані лише ті TPM, які в достатній мірі представляють ключі запитувачів [5].

Давайте розглянемо найпростіший випадок, описаний у роботі Andreas та співавт. (2006):

- Фаза ініціалізації: у різновиді є лише одна ДМП із випадковим початковим вектором. Також встановлюється певна цифра M, яка є “межею населення” і також фіксована.
- Фаза криптоаналізу: вона поділяється на три ситуації, так як і у попередньому методі атаки:
    - Вихідні значення A і B не однакові, відновлення значень не відбувається, сукупність не змінюється
    - A = B, таким чином кількість ДМП не перевищує цифру M. Всі ДМП замінюються на нові, кожен з яких стає значущою заміною одного із прихованих нейронів на протилежну швидкість. Після цього навчання відбувається згідно з правилом Гебба
    - Якщо A = B і обмеження кількості перевищує цифру M, тоді всі ДМП, які не дорівнюють отриманому виходу з системи для A і B, анулюються із сукупності

У роботі Тай-Вена та Сучена (2001) показано, що подібний підхід має сенс лише для малих нейрокриптографічних систем, і, маючи значення N> 100, K> 100, L> 10, він стає нісенітницею, оскільки інтенсивність ресурсів буде зростати в геометричній прогресії. У той же час цей дефект можна усунути на квантовому комп'ютері (це лише емпіричне припущення, яке ще не має фактів).

Окрім генетичної атаки (Klimov та співавт., 2002), геометричну атаку, яка базується на принципах нейронних мереж та ймовірнісної атаки, позначають як допустимі. Однак обидва вони є теоретичними уявленнями і навіть не мають офіційного алгоритму.

Це, у свою чергу, доводить, що в даний час нейрокриптографічні системи є єдиними системами постквантового коду, які можуть бути

реалізовані і будуть цілком криптозахищеними, щоб протистояти атакам навіть з боку квантового комп'ютера.

Це дослідження мало завдання знайти метод постквантового шифрування, який дозволив би створити стійку криптосистему, в той же час досить просту для масової реалізації.

Проаналізовано різні постквантові криптографічні системи та оцінено їх реалізацію в даний час. Як було визначено в процесі оцінки, нейрокриптографічні системи мають найвищу перспективу серед них.

За допомогою існуючих розробок був показаний варіант нейрокриптографічного протоколу, який забезпечує достатній криптозахист і в той же час не припускає складної реалізації. Це підтверджує згадану вище гіпотезу про те, що нейрокриптографічні системи є саме найбільш перспективними серед усіх алгоритмів постквантового шифрування.

1.3. Постановка задачі

У ході цієї роботи потрібно детально розглянути та проаналізувати існуючі методи та інструменти для криптографічного шифрування. Обгрунтувати вибір інтелектуальних методів для шифрування. Необхідно розробити інформаційну технологію для формування вхідного математичного опису, виконати фізичне моделювання та проаналізувати результати.

Результатом успішного виконання дипломної роботи повинні бути моделі та методи інформаційної технології нейрокриптографічного шифрування повідомлень користувача у вигляді програмного засобу.

2. ІНФОРМАЦІЙНА СИСТЕМА КРИПТОЗАХИСТУ ПОВІДОМЛЕНЬ

2.1. Модель нейромережі для шифрування та дешифрування повідомлень

Згорткова нейронна мережа (ConvNet / CNN) - це алгоритм глибокого навчання, який може приймати вхідне зображення, призначати важливість (зважувані ваги та упередження) різним аспектам / об'єктам на зображенні та мати можливість диференціювати один від іншого. Попередня обробка, необхідна в CNN, набагато нижча порівняно з іншими алгоритмами класифікації. Хоча в примітивних методах фільтри розробляються вручну, при достатній підготовці, згорткова мережа має можливість вивчати ці фільтри / характеристики[5].

Архітектура CNN аналогічна архітектурі зв'язку нейронів в мозку людини і натхненна організацією зорової кори. Окремі нейрони реагують на подразники лише в обмеженій області зорового поля, відомому як рецептивне поле. Колекція таких полів накладається на всю зорову зону.

CNN краще за звичайну нейронну мережу тому що вона здатна успішно фіксувати просторові та тимчасові залежності в зображенні за допомогою застосування відповідних фільтрів. Архітектура краще підходить до набору даних зображення завдяки зменшенню кількості задіяних параметрів та повторному використанню ваг. Іншими словами, мережу можна навчити краще розуміти витонченість зображення.

Одновимірні згорткові нейронні мережі (1D CNN)

Зазвичай при роботи зі згортковими мережами використовують двохвимірні шари conv2d для роботи з картинками. Але в згорткових мережах є і одновимірні шари. Одновимірні CNN краще справляються з одновимірними сигналами тому що:

- Замість матричних операцій прямого поширення і зворотнього поширення в 1В CNN вимагають простих операцій з масивами. Це

означає, що обчислювальна складність одновимірних CNN значно нижча, ніж двовимірних [5].

- 1D CNN з відносно неглибокою архітектурою (тобто невеликою кількістю прихованих шарів та нейронів) здатні вивчати складні завдання, що включають 1D сигнали. З іншого боку, 2D CNN зазвичай вимагають більш глибоких архітектур для вирішення таких завдань. Очевидно, що мережі з неглибокими архітектурами набагато простіші у навчанні та впровадженні [5].
- Зазвичай навчання глибоких 2D CNN вимагає спеціального обладнання (наприклад, хмарних обчислень або GPU ферм). З іншого боку, будь-яка реалізація процесора на стандартному комп'ютері є здійсненною та відносно швидкою для навчання компактним 1D CNN з кількома прихованими шарами (наприклад, 2 або менше) та нейронами (наприклад, <50) [5].

Завдяки низьким обчислювальним вимогам, компактні одновимірні мережі CNN добре підходять для обчислень реального часу та недорогих додатків, особливо на мобільних або ручних пристроях.

Програмна реалізація нашої моделі, що відтворює "безпечне з`єднання" можу бути описана наступним чином: генерація ключа, алгоритм шифрування/дешифрування, відправка повідомлення. В якості прикладу можливий наступний сценарій: Ціллю є передача повідомлення Аліси Бобу та Еві, а сторонні групи не повинні втрутитися в комунікацію і мати можливість прочитати повідомлення. Аліса створює повідомлення та використовує ключ, яким володіє і Боб, для шифрування свого повідомлення. Створений шифротекст відправляється до Бобу, котрий має ключ та розшифровує повідомлення, Єва перехоплює шифротекст і намагається розшифрувати повідомлення без знання ключа та алгоритму шифрування.

Наша модель нейромережі складається з трьох учасників листування, Аліси, Бобу, та Єви. Аліса в данному випадку являється Енкодером, Боб

декодером, а Єва стороннім енкодером (рис. 2.1.1). Енкодер буде виконувати функцію кодування тексту в шифротест використовуючи ключ. Декодер же буде виконувати розшифровку тексту використовуючи ключ та шифротекст. Тому для зручної роботи з нейромережею користувач повинен мати доступ як до енкодеру так і до декодеру, для зручного шифрування та дешифрування повідомлень. Цей набір інструментів повинен бути на двох кінцях системи передачі повідомлень, що без сумнівів є вразливістю, але в якості тестової системи такий варіант імплементації має право на існування.

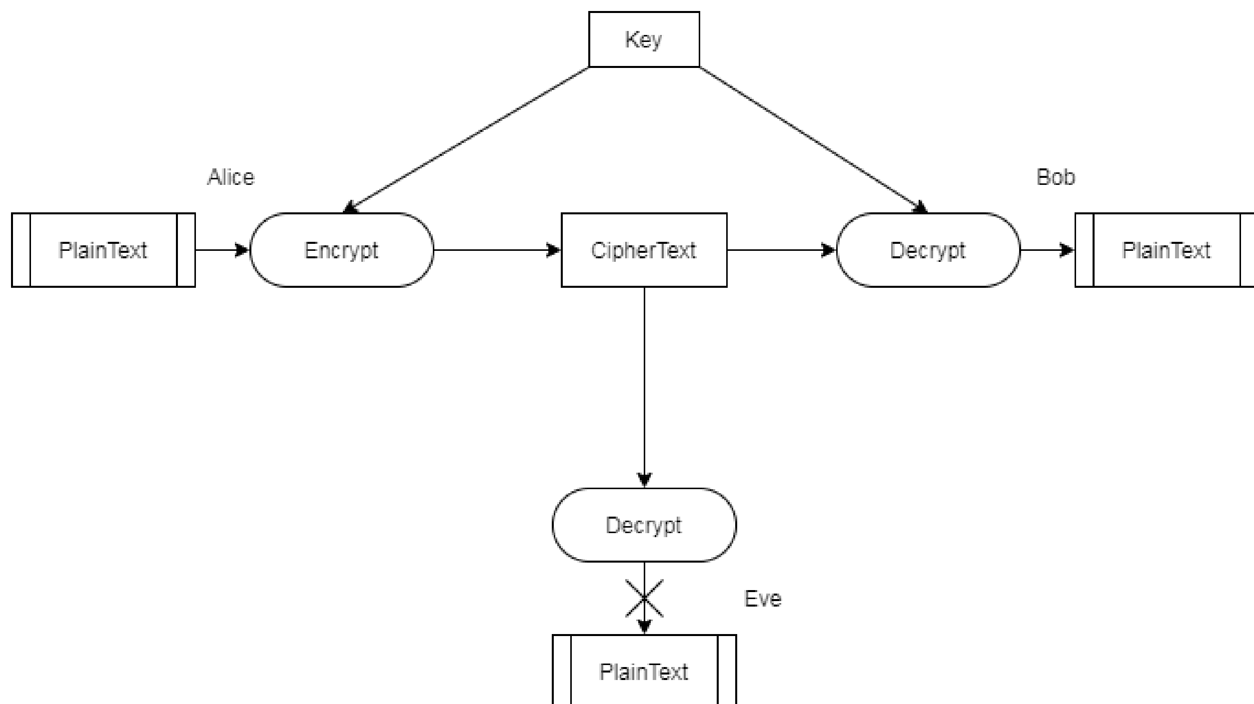

Рисунок 2.1.1 – Модель нейромережі

Енкодер (Аліса)

- Стек із декількох повторюваних одиниць (комірки conv1d), де перший елемент приймає на вхід конкатенацію ключа з повідомленням і поширює вихідні дані в наступні комірки.
- У задачі шифрування послідовність введення – це сукупність повідомлення з певним ключем що використовується для шифрування *alice_input*
- Прихований стан *alice_hidden* обчислюється за формулою:

$$\text{sigmoid}(x) = y = 1/(1 + \exp(-x)).$$
$$\text{For } x \in (-\infty, \infty), \text{sigmoid}(x) \in (0, 1).$$

Сигмоїдна функція – це обмежена, диференційована реальна функція, яка визначена для всіх реальних вхідних значень і має невід’ємну похідну в кожній точці і рівно одну точку перегину.

Енкодер складається з чотирьох послідовних шарів conv1d (рис 2.1.2)

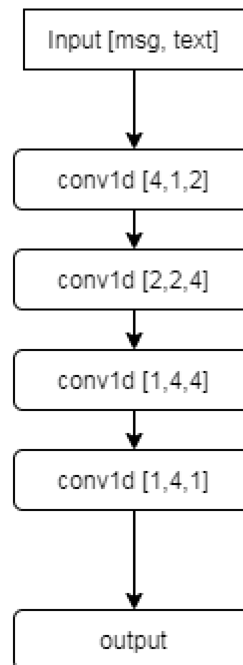

Рисунок 2.1.2 – Модель енкодера

Декодер (Боб)

- Стек з декількох повторюваних одиниць, (комірки conv1d), де перший елемент приймає на вхід конкатенацію ключа з шифротекстом і поширює вихідні дані в наступні комірки.
- Кожен повторюваний блок приймає прихований стан від попереднього і виробляє та виводить власний прихований стан.
- У задачі розшифрування вихідна послідовність - це сукупність всього тексту до шифрування.
- Перший прихований стан обчислюється за формулою:

$$\text{sigmoid}(x) = y = 1/(1 + \exp(-x)).$$

For $x \in (-\infty, \infty)$, $\text{sigmoid}(x) \in (0, 1)$.

Декодер складається з чотирьох послідовних шарів conv1d (рис 2.1.3)

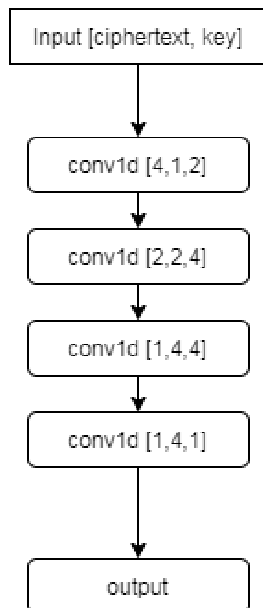

Рисунок 2.1.3 – Модель декодеру

Проміжний декодер (Ева)

- Це окремий шар, подібний до декодеру та енкодеру
- Виконує імітацію втручення до системи шифрування зловмисника
- Не повинен мати можливості відновити шифротекст без ключа
- Складається з conv1d шарів (рис. 2.1.4)

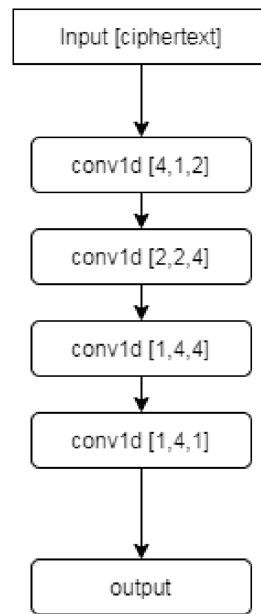

Рисунок 2.1.4 – модель проміжного декодера

Згортковий шар (convolution layer)

Згортковий шар це перший шар в згорткових мережах. Він приймає на вхід матрицю розмірів *[h1 * w1 * d1]* (рис. 2.1.5)

Далі ми маємо ядра (фільтри). Це матриця розміру *[h2 * w2 * d1]*, яка є одним кубоїдом з безлічі кубоїдів (ядер), складених один на одного (в шарі ядер). Для кожного згорткового шару існує декілька ядер, складених один на одного, саме це утворює тривимірну матрицю, яка має розміри *[h2 * w2 * d2]*, де d2 - кількість ядер. Для кожного ядра ми маємо відповідне зміщення, яке є скалярною величиною. І тоді, у нас є вихід для цього шару, матриця, яка має розміри *[h3 * w3 * d2]* [6].

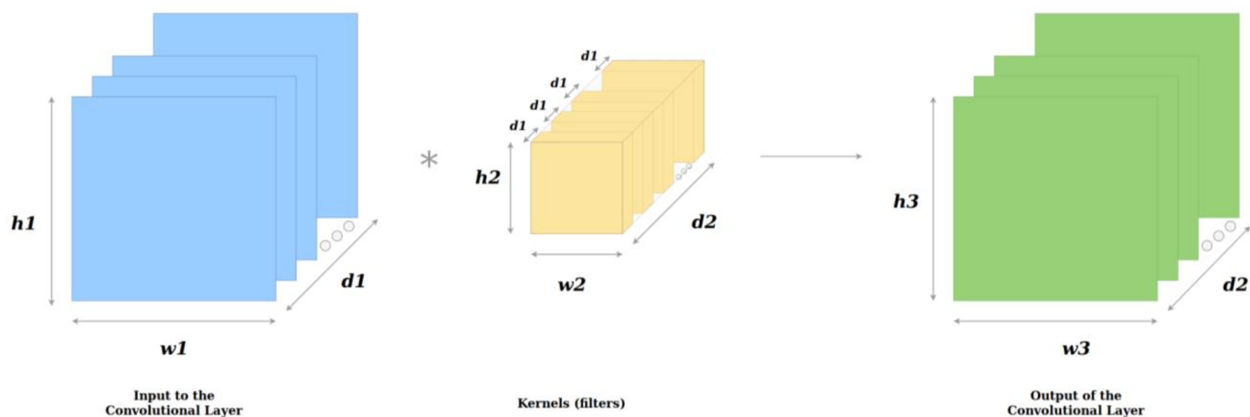

Рисунок 2.1.5 – Згортковий шар

Отже:

1. Глибина (d1) (або кількість каналів) вводу та одного ядра однакова.
2. Глибина (d2) виходу дорівнює кількості ядер (тобто глибині тривимірної матриці).

2.2. Метод навчання нейромережі

Структуру нейромережі можна записати як гру з нульовою сумою. У цьому випадку це означає, що Аліса та Боб повинні не тільки перемогти стандартну нейтральну нейронну мережу, але і найкращу версію Єви після тренування. Аліса і Боб повинні максимізувати помилку відновлення повідомлення на стороні Єви і одночасно мінімізувати помилку реконструкції на стороні Боба. Проблему оптимізації гри з нульовою сумою можна записати так [8]:

$$L_{AB}(\theta_A, \theta_B) = L_B(\theta_A, \theta_B) - L_E(\theta_A, O_E(\theta_A))$$

- $L_E(\theta_A, \theta_E)$ – це функція втрат, що характеризує помилку розшифрування повідомлення Аліси на стороні Єви;

$L_E(\theta_A, O_E(\theta_E))$ – оптимальне значення функції втрат, що характеризує помилку розшифрування повідомлення Аліси на стороні Єви, на заданій ітерації навчання;

- $L_{AB}(\theta_A, \theta_B)$ – функція втрат, що характеризує помилку розшифрування повідомлення Аліси на стороні Боба.

Ця комбінація відображає, що Аліса та Боб хочуть мінімізувати помилку реконструкції Боба та максимізувати помилку реконструкції "оптимальної Єви", метою нашої задачі оптимізації є пошук наступного рішення :

$$(O_A, O_B) = argmin_{(\theta_A, \theta_B)}(L_{AB}(\theta_A, \theta_B))$$

Для мінімізації описаної фунації втрат використовується традиційний алгоритм навчання – алгоритм зворотного поширення (Backpropagation), – це алгоритм контрольованого навчання нейронних мереж з використанням градієнтного спуску. Враховуючи штучну нейронну мережу та функцію помилок, метод обчислює градієнт функції помилки щодо ваг нейронної мережі. Це узагальнення правила дельта для персептронів до багатошарових нейромереж прямого поширення [7].

Частина назви "зворотне" випливає з того, що обчислення градієнта відбувається назад через мережу, причому градієнт кінцевого шару ваг обчислюється першим, а градієнт першого шару ваг - останнім. Часткові обчислення градієнта з одного шару повторно використовуються при обчисленні градієнта для попереднього шару. Цей зворотний потік інформації про помилку дозволяє ефективно обчислювати градієнт на кожному шарі порівняно з наївним підходом обчислення градієнта кожного шару окремо.

Варто відмітити, що алгоритм, як правило, не знаходить глобального мінімального рішення і що може бути декілька рівнооптимальних рішень. Що стосується класичного GAN, навчання починається з випадкової ініціалізації. На цьому етапі можна стверджувати, що рамки навчання та проблеми оптимізації надзвичайно прості.

Зворотне розповсюдження було одним з перших методів, здатних продемонструвати, що штучні нейронні мережі можуть засвоїти хороші внутрішні уявлення, тобто їх приховані шари засвоїли нетривіальні особливості. Експерти, які вивчали багатошарові мережі прямого передавання, навчені з використанням зворотного розповсюдження, насправді виявили, що багато вузлів засвоїли функції, подібні до тих, що були розроблені експертами-людьми, і тих, які були знайдені неврологами, що досліджують біологічні нейронні мережі в мозку ссавців (наприклад, деякі вузли навчилися виявляти краї, а інші обчислювали фільтри Габора). Навіть важливіше, що через ефективність алгоритму та той факт, що експерти доменів більше не вимагали виявлення відповідних функцій, зворотне розповсюдження

дозволило застосовувати штучні нейронні мережі до значно ширшого поля проблем, які раніше були обмежені через час та обмеження витрат.

Адам - це алгоритм оптимізації, який може бути використаний замість класичної стохастичної процедури градієнтного спуску для оновлення вагових коефіцієнтів мережі ітеративно на основі навчальних даних.

Адам був представлений Дідеріком Кінгмою з OpenAI та Джиммі Ба з Університету Торонто у своєму документі ICLR 2015 року (плакат) під назвою „Адам: метод стохастичної оптимізації“ [9].

Представляючи алгоритм, автори перелічують привабливі переваги використання Адама в неопуклих задачах оптимізації, наступним чином:

- Прямо реалізовані.
- Обчислювально ефективні.
- Невеликі вимоги до пам'яті.
- Інваріант діагональному масштабуванню градієнтів.
- Добре підходить для великих проблем з точки зору даних та (або) параметрів.
- Підходить для нестаціонарних цілей.
- Підходить для проблем із дуже шумними / або рідкісними градієнтами.
- Гіперпараметри мають інтуїтивну інтерпретацію і, як правило, вимагають незначних налаштувань.

Адам відрізняється від класичного стохастичного градієнтного спуску. Стохастичний градієнтний спуск підтримує єдину швидкість навчання (називається альфа) для всіх оновлень ваги, і швидкість навчання не змінюється під час навчання.

Швидкість навчання підтримується для кожної ваги мережі (параметра) і окремо адаптується в міру розгортання навчання.

Автори описують Адама як поєднання переваг двох інших розширень стохастичного градієнтного спуску. Зокрема:

- Адаптивний градієнтний алгоритм (AdaGrad), який підтримує швидкість навчання за параметром, що покращує ефективність при проблемах з розрідженими градієнтами (наприклад, проблеми з природною мовою та комп'ютерним зором).
- Розмноження середньоквадратичного корінця (RMSProp), яке також підтримує коефіцієнти навчання за параметрами, які адаптуються на основі середнього значення останніх величин градієнтів для ваги (наприклад, наскільки швидко вона змінюється). Це означає, що алгоритм добре справляється з онлайновими та нестаціонарними проблемами (наприклад, шумними).

Адам використовує переваги як AdaGrad, так і RMSProp. Замість того, щоб адаптувати швидкість навчання параметрів на основі середнього першого моменту (середнього значення), як у RMSProp, Адам також використовує середнє значення других моментів градієнтів (нецентрова дисперсія) [10].

Зокрема, алгоритм обчислює експоненціальну ковзну середню градієнта та квадратичний градієнт, а параметри beta1 та beta2 контролюють швидкість занепаду цих ковзних середніх.

Початкове значення ковзних середніх та значень beta1 та beta2, близьких до 1,0 (рекомендовані), призводять до зміщення оцінок моменту до нуля. Це упередження долається спочатку обчисленням упереджених оцінок, а потім обчисленням, скоригованим із зміщенням.

3. РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1. Короткий опис програмного забезпечення

Дана нейромережа побудована за допомогою бібліотеки Tensorflow – бібліотека для глибокого машинного навчання, котра використовується при побудові і тренуванні нейронних мереж для різних задач. Також використовуються і інші бібліотеки, котрі наведені в таблиці 3.1.1.

Таблиця 3.1.1 – Використані бібліотеки

Назва бібліотеки	**Опис**
Numpy	Бібліотека для мови Python, що додає підтримку розширених багато-вимірних масивів та матриць, разом з великою колекцією високорівневих математичних функцій для роботи з цими масивами.
Tensorflow	Головна утиліта глибокого навчання. Це відкрита бібліотека машинного навчання, що використовує графіки потоку даних для побудови моделей. Вона дозволяє розробнику створювати широкомасштабні нейронні мережі з багатьма шарами.
Pandas	Бібліотека для маніпуляцій з даними та їх аналізу. Зокрема, вона пропонує структури даних та операції для маніпулювання числовими таблицями та часовими рядами
Math	Стандартний модуль для використання математичних функцій
Matplotlib	Це бібліотека для побудови графіків для мови програмування Python. Вона надає об'єктно-орієнтований API для вбудовування графіків у додатки з використанням загальних інструментів графічного інтерфейсу, таких як Tkinter, wxPython, Qt або GTK.

Продовження таблиці 3.1.1

Назва бібліотеки	**Опис**
Seaborn	Це бібліотека візуалізації даних Python, заснована на matplotlib. Вона забезпечує інтерфейс високого рівня для малювання привабливих та інформативних статистичних графічних зображень.
Socket	Sockets та socket API використовуються для надсилання повідомлень через мережу. Вони забезпечують форму міжпроцесорної комунікації (IPC). Мережа може бути логічною, локальною мережею до комп’ютера або тією, яка фізично пов’язана із зовнішньою мережею власними зв’язками з іншими мережами.
Sys	Модуль sys у Python надає різні функції та змінні, які використовуються для маніпулювання різними частинами середовища виконання Python. Це дозволяє працювати з інтерпретатором, оскільки забезпечує доступ до змінних та функцій, які сильно взаємодіють з інтерпретатором.
Pickle	Pickle у Python в основному використовується для серіалізації та десеріалізації структури об’єктів Python. Іншими словами, це процес перетворення об’єкта Python в байтовий потік, щоб зберегти його у файлі / базі даних, підтримувати стан програми в сеансах або транспортувати дані через мережу.

Основними в нейромережі є два шари, вхідний (Аліса) та вихідний (Боб), також для перевірки криптографічної стійкості було додано проміжний шар (Ева). Дана робота має на меті зімітувати безпечний зв'язок між користувачами.

Реалізація "безпечного зв'язку" може бути описана наступним чином: Генерація ключа, алгоритм шифрування / дешифрування, відправка повідомлення. Приклад сценарію такий: мета полягає в тому, щоб Аліса надіслала повідомлення Бобу, а Єва, сторона, яка намагається зламати спілкування Аліси та Боба, не змогла прочитати повідомлення. Аліса приймає повідомлення (звичайний текст) і використовує ключ, яким вона ділиться з Бобом, для шифрування звичайного тексту в шифротекст. Цей шифротекст надсилається Бобу, який використовує зв'язок і ключ, і розшифровує шифротекст в звичайний текст. Єва перехоплює передачу шифротексту і намагається відновити відкритий текст без знання ключів, а в деяких випадках і алгоритму шифрування.

Здатність швидко адаптуватися до змін і вчитися на помилках поступово стають характеристиками, що визначають стійкість з точки зору безпеки. Нейронні мережі успішно виконують цю динамічну можливість самонавчання для багатьох інших завдань, таких як розпізнавання зображень для великих наборів даних. Нейронні мережі можна використовувати у різних варіантах стосовно криптографії: для створення симетричних ключів для протоколу (ключі приховані у вагах), для шифрування/дешифрування (алгоритм є секретом у моделі) та для підпису/перевірки. Потенціал нейронних мереж полягає в тому, що мережі вигадують власні алгоритми шифрування/дешифрування та підпису/перевірки. Однак експерти з криптографії не враховують застосування нейромереж для криптографії, оскільки нейромережі мають великий простір рішень і не є ефективними для криптоаналізу через невідповідність алгоритму.

За допомогою цієї роботи проводиться дослідження, як нейронні мережі можуть бути ефективними для криптографії. Тут представляється нейронний алгоритм криптографії для симетричного шифрування, який може не тільки гарантувати секретність, але й виконує три принципи безпеки (конфіденційність, цілісність та доступність). Замість того, щоб зосередитися на криптоаналізі (зміцненні Єви), робота була зосереджена на поліпшенні

конфіденційності повідомлення, яке передається між Алісою та Бобом. Метою є досягнення низького рівня помилок шляхом зміни гіперпараметрів алгоритму, а також реалізація протоколу через мережеві сокети та імітація зв'язку між Алісою та Бобом.

Як згадувалося вище, було використано згорткові нейронні мережі (CNN) для шифрування / дешифрування (128-бітний ключ). Щоб створити оригінальний алгоритм, було змінено код від Ліама Шоневельда («Змагальна нейронна криптографія в Theano») та користувача GitHub ankeshanand («Змагальна нейронна криптографія в TensorFlow»), щоб включити згорткові шари та експериментувати з параметрами, змінивши швидкість навчання оптимізатора, епохи навчання та тестування, початкові ваги тощо. Було використано фреймворк TensorFlow

Опис функцій:

- gen_data – функція для генерації даних для навчання, на вхід приймає розмір батчу, довжину ключа та повідомлення, і повертає батч данних
- conv1d – налаштування над шаром conv1d, на вхід приймає форму фільтру, вхідні данні, крок та ім’я. Ініціалізує нову змінну та використовує її в шарі.
- train – функція для тренування нейромережі на нових даних
- _train – більш подрібна функція тренування, використовується як налаштування над функцією тренування
- test – функція перевірки данних, по закінченню навчання перетворює текст з ключем в шифротекст
- plot_results – допоміжна функція – виводить графік навчання
- sock_send створює сокет TCP / IP для Аліси, прив'язує сокет до порту (порт localhost 10000) і серіалізує дані в масив байтів.
- sock_recv_init створює сокет TCP / IP для Боба, прив'язує сокет до порту і змушує його прослуховувати вхідні з'єднання.

- sock_recv чекає з'єднання, і як тільки він отримує дані, він десеріалізує їх.

3.2. Результати навчання

Навчання було проведене з такими параметрами (рис. 3.2.1). Розмір батчу – 512, кількість епох – 50, швидкість навчання – 0,0008, розмір ключа та шифротексту – 16 символів.

```
[4]  crypto_msg_len = N = 16
     crypto_key_len = 16
     batch_size = 512
     epochs = 50
     learning_rate = 0.0008 #Optimizer learning rate
```

Рисунок 3.2.1 – Ділянка коду, що опису параметри навчання

Функції втрат були сформовані наступним чином (рис 3.2.2): була вирахувана помилка шифрування – різниця повідомлення та виходу приведена до модулю та взято середнє значення елементів по розміру тензору. Функція втрат вираховується за формулою

$$loss_alice = decrypt_err_alice + (1 - decrypt_err_eve)^2$$

Аналогічно для Бобу

$$loss_bob = decrypt_err_bob + (1 - decrypt_err_eve)^2$$

Де decrypt_err_name – помилка шифрування.

```
# Loss Functions
decrypt_err_eve = tf.reduce_mean(tf.abs(msg - eve_output))
decrypt_err_alice = tf.reduce_mean(tf.abs(msg - alice_output))
loss_alice = decrypt_err_alice + (1. - decrypt_err_eve) ** 2.
decrypt_err_bob = tf.reduce_mean(tf.abs(msg - bob_output))
loss_bob = decrypt_err_bob + (1. - decrypt_err_eve) ** 2.
```

Рисунок 3.2.2 – Код, що реалізує функції втрат

Навчання було виконане з оптимізаторами Adagrad, Momentum та Adam, параметри навчання вказано вище, найкращим себе показав Адам (рис 3.2.3).

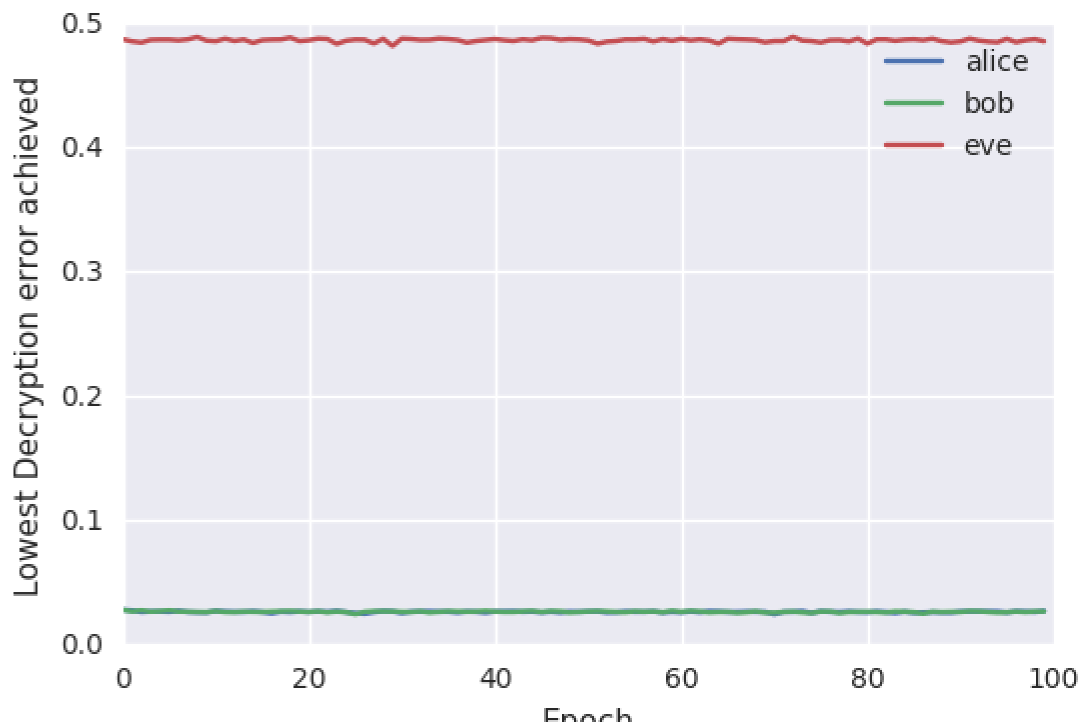

Рисунок 3.2.3 – Графік залежності втрат від кількості епох навчання при використанні оптимізатора Adam

Як видно з графіку, помилка у Єви максимальна, тоді як помилка у Боба та Аліси мінімальна, тобто шифрування вдалося, і наші дані відомі лише Алісі та Бобу під час шифрування.

Результат навчання (рис. 3.2.4 – 3.2.8):

```
Enter some text: abc
01100001 01100010 01100011
plaintext_to_Alice =  [[ 0.  1.  1.  0.  0.  0.  0.  1.  0.  1.  1.  0.  0.  0.  1.  0.]
 [ 0.  1.  1.  0.  0.  0.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.]]
```

Рисунок 3.2.4 – Ілюстрація кодування введеного повідомлення

```
Testing Alice
messages =
 [[ 0.  1.  1.  0.  0.  0.  0.  1.  0.  1.  1.  0.  0.  0.  1.  0.]
 [ 0.  1.  1.  0.  0.  0.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.]]
keys =
 [[0 1 1 1 1 0 1 1 1 1 0 0 1 0 1 0]
 [0 1 0 1 1 0 0 0 0 0 1 1 0 1 1 0]]
plaintext_to_Alice =  [[ 0.  1.  1.  0.  0.  0.  0.  1.  0.  1.  1.  0.  0.  0.  1.  0.]
 [ 0.  1.  1.  0.  0.  0.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.]]
test alice_output (***Cipher Text***) =
 [[ 0.  1.  1.  0.  0.  0.  0.  1.  0.  1.  1.  0.  0.  0.  1.  0.]
 [ 0.  1.  1.  0.  0.  0.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.]]
SOCK_SEND
message =  [[ 0.  1.  1.  0.  0.  0.  0.  1.  0.  1.  1.  0.  0.  0.  1.  0.]
 [ 0.  1.  1.  0.  0.  0.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.]]
alice_errors_test =  0.0
```

Рисунок 3.2.5 – Ілюстрація щодо шифрування повідомлення Аліси

```
Testing Bob
messages =
 [[ 0.  1.  1.  0.  0.  0.  0.  1.  0.  1.  1.  0.  0.  0.  1.  0.]
 [ 0.  1.  1.  0.  0.  0.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.]]
keys =
 [[0 0 1 1 0 1 1 1 0 1 0 0 0 1 1 1]
 [0 0 1 1 0 0 1 0 1 1 0 1 0 0 1 1]]
SOCKET_RECV ====

waiting for a connection
connection from ('127.0.0.1', 49548)
received  [[ 0.  1.  1.  0.  0.  0.  0.  1.  0.  1.  1.  0.  0.  0.  1.  0.]
 [ 0.  1.  1.  0.  0.  0.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.]]
[[ 0.  1.  1.  0.  0.  0.  0.  1.  0.  1.  1.  0.  0.  0.  1.  0.]
 [ 0.  1.  1.  0.  0.  0.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.]]
test bob_input (***Cipher Text***) =
 [[ 0.  1.  1.  0.  0.  0.  0.  1.  0.  1.  1.  0.  0.  0.  1.  0.  0.  0.
   1.  1.  0.  1.  1.  1.  0.  1.  0.  0.  0.  1.  1.  1.]
 [ 0.  1.  1.  0.  0.  0.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
   1.  1.  0.  0.  1.  0.  1.  1.  0.  1.  0.  0.  1.  1.]] 2
test bob_output (***Plain Text***)=
 [[ 0.  1.  1.  0.  0.  0.  0.  1.  0.  1.  1.  0.  0.  0.  1.  0.]
 [ 0.  1.  1.  0.  0.  0.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.]] 2
bob_errors_test =  0.0
```

Рисунок 3.2.6 – Ілюстрація щодо передачі повідомлення Бобу

```
Testing Eve
messages =
 [[ 0.  1.  1.  0.  0.  0.  0.  1.  0.  1.  1.  0.  0.  0.  1.  0.]
 [ 0.  1.  1.  0.  0.  0.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.]]
keys =
 [[0 1 0 1 0 1 1 1 1 1 1 0 1 1 0 0]
 [1 0 1 1 1 1 1 0 0 0 1 1 1 1 1 1]]
eve_errors_test =  0.3125
Time taken for Testing (seconds):  0.053194522857666016
bob_ouput_1 ==  [ 0.  1.  1.  0.  0.  0.  0.  1.  0.  1.  1.  0.  0.  0.  1.  0.  0.  1.
  1.  0.  0.  0.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.]
```

Рисунок 3.2.7 – Ілюстрація передачі повідомлення Єві

```
b2 =  0b01100001011000100110001100000000
abc
Bob recovered Plain Text =  abc
```

Рисунок 3.2.8 – Розшифроване повідомлення на стороні Боба

В результаті на стороні Боба отримано те ж саме повідомлення, що і було введено на стороні Аліси (рис. 3.2.8). При цьому Єва розшифровує повідомлення з певним рівнем помилок, що доводить захищеність шифру.

ВИСНОВКИ

Моделювання загроз: за допомогою нейрокриптографічного алгоритму (НКА), навіть якщо симетричний ключ порушений, алгоритм все ще прихований у тренуванні та вагах. Це призводить до підтримання принципу «секретність - не є безпека», оскільки той алгоритм може бути наданий публічно, але динамічна самоконфігуруюча природа нейронних мереж ускладнює злом. Однак, якщо буде розроблено метод вилучення ваг з машин НКА, хакери зможуть скомпрометувати алгоритм, доки не буде примусово проведена перекваліфікація НКА. Ця ж перевага також служить недоліком, оскільки CNN не узгоджуються. Однак алгоритм криптографії може бути посилений для задоволення майбутніх потреб шляхом визначення / налаштування гіперпараметрів без використання тестування на проникнення, що неможливо з існуючими традиційними ланцюжками інструментів для НКА. Відкликання / перевипуск ключа не є складним, оскільки достатньо лише перекваліфікації НКО на законних об'єктах / машинах. На сьогодні ключі та алгоритми НКА можуть не відповідати галузевим стандартам, але якщо їх вдосконалити, НКА можна спробувати застосувати ширше.

Результати демонструють, що шифрування нейрокриптографічним алгоритмом можливе, і показує результати не гірші за відомі алгоритми шифрування але потребує часу на навчання. Також НКА потребує обчислювальних потужностей, тому не відомо коли нейрокриптографічне шифрування стане можливим. Однак, комп’ютери з масовою паралельною обробкою (MPP) можуть пришвидшити навчання з питань НКА та забезпечити можливість частої та ефективної крипто-переробки.

СПИСОК ЛІТЕРАТУРИ

1. “An Introduction to Cryptographic Security Methods and Their Role in Securing Low Resource Computing Devices” 2015.: https://veridify.com/wp-content/uploads/2014/03/Intro-to-SecurityWhitePaper-2015.pdf., SecureRF Corporation, 2015
2. Mohammed Al-Maitah, “Appliance of Neuron Networks in Cryptographic Systems” Proc. King Saud University, 2014
3. “A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way” 2018.: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53., Sumit Saha, 2018
4. Alexander Klimov, Anton Mityagin, and Adi Shamir, “Analysis of Neural Cryptography”, Computer Science Department, The Weizmann Institute
5. “Convolutional Neural Network”, 2018.: https://towardsdatascience.com/convolutional-neural-network-17fb77e76c05 Arc, 2018
6. Dylan Modesitt, Tim Henry, Jon Coden, and Rachel Lathe, “Neural Cryptography: From Symmetric Encryption to Adversarial Steganography”
7. “Gentle Introduction to the Adam Optimization Algorithm for Deep Learning”, 2017.: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/, Jason Brownlee, 2017
8. Khan A. A Survey, of the Recent Architectures of Deep Convolutional Neural
9. Networks / A. Khan, A. Sohail, U. Zahoora, A. S. Qureshi // 2019. – P. 1–62. – Available at: https://arxiv.org/ftp/arxiv/papers/1901/1901.06032.pdf.
10. Ramachandram D. Deep Multimodal Learning / D. Ramachandram, G.W. Taylor // IEEE Signal Processing Magazine. – 2017. – P. 96 – 108.
11. N. Scaife, K. R. B. Butler, P. Traynor, and H. Carter, “CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data”, Proc. – Int. Conf. Dist. Comp. Sys., 2016.

12. Berman D. S., Buczak A. L., Chavis J. S., Corbett C. L. A Survey of Deep Learning Methods for Cyber Security. 10, 122,2019.
13. Deng L., Yu D. Deep learning: Methods and applications. Found. Trends Signal Process. 7, 197–387, 2014.
14. Fraley J. B., Cannady J. The promise of machine learning in cybersecurity. In SoutheastCon, 2017, pp. 1-6. IEEE. March 2017.
15. Jolfaei A., Vizandan A., Mirghadri A. Image encryption using HC-128 and HC-256 stream ciphers. International Journal of Electronic Security and Digital Forensics, vol. 4, no. 1, pp. 19-42, 2012.
16. Dolev S., Lodha S. Cyber Security Cryptography and Machine Learning. In Proceedings of the First International Conference, CSCML 2017, Beer-Sheva, Israel, June 29-30, 2017.
17. Hu X., Wang T., Stoecklin M. P., Schales D. L., Jang J., Sailer R. Asset risk scoring in enterprise network with mutually reinforced reputation propagation. In 2014 IEEE Security and Privacy Workshops (SPW), pp. 61-64. IEEE, May, 2014.

ДОДАТКИ

Додаток А

```
!pip uninstall tensorflow
!pip uninstall tensorflow-gpu
!pip install tensorflow-gpu==1.14.0
!wget https://developer.nvidia.com/compute/cuda/9.0/Prod/local_installers/cuda-repo-ubuntu1604-9-0-local_9.0.176-1_amd64-deb
!dpkg -i cuda-repo-ubuntu1604-9-0-local_9.0.176-1_amd64-deb
!apt-key add /var/cuda-repo-9-0-local/7fa2af80.pub
!apt-get update
!apt-get install cuda=9.0.176-1
!nvcc --version
```

```
import numpy as np
import tensorflow as tf
import pandas as pd
import math
import matplotlib.pyplot as plt
import seaborn as sns
import socket
import sys
import pickle
```

```
crypto_msg_len = N = 16
crypto_key_len = 16
batch_size = 512
epochs = 50
learning_rate = 0.0008 #Optimizer learning rate

# Function to generate n random messages and keys
def gen_data(n=batch_size, msg_len=crypto_msg_len, key_len=crypto_key_len):
    #return (np.random.randint(0, 2, size=(n, msg_len))*2-1), (np.random.randint(0, 2, size=(n, key_len))*2-1)
    return (np.random.randint(0, 2, size=(n, msg_len))), (np.random.randint(0, 2, size=(n, key_len)))
```

```
def conv1d(input_, filter_shape, stride, name="conv1d"):
    with tf.variable_scope(name):
        w = tf.get_variable('w', shape=filter_shape, initializer=tf.contrib.layers.xavier_initializer())
        conv = tf.nn.conv1d(input_, w, stride, padding='SAME')
        return conv
```

```
tf.reset_default_graph()
```

```
# Placeholder variables for Message and Key
msg = tf.placeholder("float", [None, crypto_msg_len])
key = tf.placeholder("float", [None, crypto_key_len])

# Weights for fully connected layers
w_alice = tf.get_variable("alice_w", shape=[2 * N, 2 * N], initializer=tf.contrib.layers.xavier_initializer())
w_bob = tf.get_variable("bob_w", shape=[2 * N, 2 * N], initializer=tf.contrib.layers.xavier_initializer())
w_eve1 = tf.get_variable("eve_w1", shape=[N, 2 * N], initializer=tf.contrib.layers.xavier_initializer())
w_eve2 = tf.get_variable("eve_w2", shape=[2 * N, 2 * N], initializer=tf.contrib.layers.xavier_initializer())

# Alice's Machine Network
# FC layer -> Conv Layer (4 1-D convolutions)
#alice_input = tf.concat(concat_dim=1, values=[msg, key])
alice_input = tf.concat(axis=1, values=[msg, key])
alice_hidden = tf.nn.sigmoid(tf.matmul(alice_input, w_alice))
alice_hidden = tf.expand_dims(alice_hidden, 2)

h0 = tf.nn.relu(conv1d(alice_hidden, [4,1,2], stride=1, name="alice"+'_h0_conv'))
h1 = tf.nn.relu(conv1d(h0, [2,2,4], stride=2, name="alice"+'_h1_conv'))
h2 = tf.nn.relu(conv1d(h1, [1,4,4], stride=1, name="alice"+'_h2_conv'))
h3 = tf.nn.tanh(conv1d(h2, [1,4,1], stride=1, name="alice"+'_h3_conv'))
alice_output = tf.squeeze(h3) # eliminate dimensions of size 1 from the shape of a tensor
```

```
# Bob's Machine Network (gets the output (cipher text) of Alice's network)
# FC layer -> Conv Layer (4 1-D convolutions)
#bob_input = tf.concat(concat_dim=1, values=[alice_output, key])
bob_input = tf.concat(axis=1, values=[alice_output, key])
bob_hidden = tf.nn.sigmoid(tf.matmul(bob_input, w_bob))
bob_hidden = tf.expand_dims(bob_hidden, 2)

h0 = tf.nn.relu(conv1d(bob_hidden, [4,1,2], stride=1, name="bob"+'_h0_conv'))
h1 = tf.nn.relu(conv1d(h0, [2,2,4], stride=2, name="bob"+'_h1_conv'))
h2 = tf.nn.relu(conv1d(h1, [1,4,4], stride=1, name="bob"+'_h2_conv'))
h3 = tf.nn.tanh(conv1d(h2, [1,4,1], stride=1, name="bob"+'_h3_conv'))
bob_output = tf.squeeze(h3) # eliminate dimensions of size 1 from the shape of a tensor

# Eve's Machine Network
# FC layer -> FC layer -> Conv Layer (4 1-D convolutions)
eve_input = alice_output
eve_hidden1 = tf.nn.sigmoid(tf.matmul(eve_input, w_eve1))
eve_hidden2 = tf.nn.sigmoid(tf.matmul(eve_hidden1, w_eve2))
eve_hidden2 = tf.expand_dims(eve_hidden2, 2)

h0 = tf.nn.relu(conv1d(eve_hidden2, [4,1,2], stride=1, name="eve"+'_h0_conv'))
h1 = tf.nn.relu(conv1d(h0, [2,2,4], stride=2, name="eve"+'_h1_conv'))
h2 = tf.nn.relu(conv1d(h1, [1,4,4], stride=1, name="eve"+'_h2_conv'))
h3 = tf.nn.tanh(conv1d(h2, [1,4,1], stride=1, name="eve"+'_h3_conv'))
eve_output = tf.squeeze(h3)
```

```
alice_errors, bob_errors, eve_errors = [], [], []

# Loss Functions
decrypt_err_eve = tf.reduce_mean(tf.abs(msg - eve_output))
decrypt_err_alice = tf.reduce_mean(tf.abs(msg - alice_output))
loss_alice = decrypt_err_alice + (1. - decrypt_err_eve) ** 2.
decrypt_err_bob = tf.reduce_mean(tf.abs(msg - bob_output))
loss_bob = decrypt_err_bob + (1. - decrypt_err_eve) ** 2.

# Get training variables corresponding to each network
t_vars = tf.trainable_variables()
alice_vars = [var for var in t_vars if 'alice_' in var.name]
bob_vars =   [var for var in t_vars if 'bob_' in var.name]
eve_vars =   [var for var in t_vars if 'eve_' in var.name]

# Build the optimizers, can play with different optimizers

'''
alice_optimizer = tf.train.AdagradOptimizer(learning_rate, initial_accumulator_value=0.1,
                       use_locking=False, name='Adagrad').minimize(loss_alice, var_list=alice_vars)
bob_optimizer = tf.train.AdagradOptimizer(learning_rate, initial_accumulator_value=0.1,
                        use_locking=False, name='Adagrad').minimize(loss_bob, var_list=bob_vars)
eve_optimizer = tf.train.AdagradOptimizer(learning_rate, initial_accumulator_value=0.1,
                        use_locking=False, name='Adagrad').minimize(decrypt_err_eve, var_list=eve_vars)

alice_optimizer = tf.train.MomentumOptimizer(0.01, 0.9).minimize(loss_alice, var_list=alice_vars)
bob_optimizer = tf.train.MomentumOptimizer(0.01, 0.9).minimize(loss_bob, var_list=bob_vars)
eve_optimizer = tf.train.MomentumOptimizer(0.01, 0.9).minimize(decrypt_err_eve, var_list=eve_vars)

'''
alice_optimizer = tf.train.AdamOptimizer(learning_rate).minimize(loss_alice, var_list=alice_vars)
bob_optimizer = tf.train.AdamOptimizer(learning_rate).minimize(loss_bob, var_list=bob_vars)
eve_optimizer = tf.train.AdamOptimizer(learning_rate).minimize(decrypt_err_eve, var_list=eve_vars)
```

```
def train(sess):
    # Begin Training
    tf.initialize_all_variables().run(session=sess)
    for i in range(epochs):
        iterations = 2000

        alice_loss, _, _ = _train('alice', iterations, sess)
        alice_errors.append(alice_loss)
        print( 'Training Alice, Epoch:', i + 1, ' error: ', alice_loss)

        _, bob_loss, _ = _train('bob', iterations, sess)
        bob_errors.append(bob_loss)
        print( 'Training Bob, Epoch:', i + 1, ' error: ', bob_loss)

        _, _, eve_loss = _train('eve', iterations, sess)
        eve_errors.append(eve_loss)
        print( 'Training Eve, Epoch:', i + 1, ' error: ', eve_loss)
```

```
def _train(network, iterations, sess):
    alice_decrypt_error, bob_decrypt_error, eve_decrypt_error = 1., 1., 1.

    bs = batch_size
    # Train Eve for two minibatches to give it a slight computational edge
    if network == 'eve':
        bs *= 2

    for i in range(iterations):
        msg_in_val, key_val = gen_data(n=bs, msg_len=crypto_msg_len, key_len=crypto_key_len)
        feed_dict={msg: msg_in_val, key: key_val}
        if network == 'alice':
            _, decrypt_err = sess.run([alice_optimizer, decrypt_err_alice], feed_dict = feed_dict)
            alice_decrypt_error = min(alice_decrypt_error, decrypt_err)
        elif network == 'bob':
            _, decrypt_err = sess.run([bob_optimizer, decrypt_err_bob], feed_dict = feed_dict)
            bob_decrypt_error = min(bob_decrypt_error, decrypt_err)
        elif network == 'eve':
            _, decrypt_err = sess.run([eve_optimizer, decrypt_err_eve], feed_dict = feed_dict)
            eve_decrypt_error = min(eve_decrypt_error, decrypt_err)

    return alice_decrypt_error, bob_decrypt_error, eve_decrypt_error
```

```
text = input('Enter some text: ')
bintext = ' '.join('{0:08b}'.format(ord(x), 'b') for x in text)
print (bintext)
b1  = bintext.replace(" ", "")
#b1 = b1.zfill(48)
pad = len(b1)%16

v1 = np.array([])
for i in range(0, len(b1)):
    v1 = np.append(v1, int(b1[i]))

#apply the padding
for i in range(0, pad):
    v1 = np.append(v1, int(0))
total_len = len(b1) + pad

plaintext_to_Alice = v1.reshape(int(total_len/16), 16)
print('plaintext_to_Alice = ', plaintext_to_Alice)
```

```
def sock_send(message):
    # Create a TCP/IP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_address = ('localhost', 10000)
    sock.connect(server_address)
    #To send data over network, you need to serialize it into an array of bytes,
    #then deserialize it back. In Python, serialization of most objects can be done via pickle module:
    print('message = ', message)
    msg2 = pickle.dumps(message)
    #print('msg2 = ', msg2)
    sock.send(msg2)
```

```
def sock_recv_init():
    # Create a TCP/IP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # Bind the socket to the port
    address = ('localhost', 10000)
    print('starting up on %s port %s' % address)
    sock.bind(address)
    # Listen for incoming connections
    sock.listen(1)
    return sock

def sock_recv(sock):
    # Wait for a connection
    print( 'waiting for a connection')
    connection, client_address = sock.accept()
    print('connection from', client_address)
    # Receive the data (max 48 bytes)
    data = connection.recv(1024*10)
    #print('received "%s"' % data)
    data1 = pickle.loads(data)
    print('received ', data1)
    return data1

sock = sock_recv_init()
```

```
test_file_msg = "testmsg.txt"
test_file_keys = "testkey.txt"
```

```
def test(network, sess):
        alice_decrypt_error, bob_decrypt_error, eve_decrypt_error = 1., 1., 1.
        alice_encrypt_time = 0
        bob_decrypt_time = 0
        bob_output_1 = 0

        #bs = 3 #batch_size
        bs = int(total_len/16) #batch_size
        messages, keys = gen_data(n=bs, msg_len=crypto_msg_len, key_len=crypto_key_len)
        #test message to get the code complete
        messages = np.array([
                    [1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1],
                    [0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1],
                    [1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1]
                   ])


        #feed_dict={msg: messages, key: keys}
        feed_dict={msg: plaintext_to_Alice, key: keys}

        print( 'messages = \n', plaintext_to_Alice)
        print( 'keys = \n', keys)

        if network == 'alice':
            print('plaintext_to_Alice = ', plaintext_to_Alice)
            start_time = time.time()
            _, decrypt_err, alice_output_1  = sess.run([alice_optimizer, decrypt_err_alice, alice_output], feed_dict = feed_dict)
            end_time = time.time()
            alice_encrypt_time = end_time-start_time
            print('test alice_output (***Cipher Text***) = \n', alice_output_1)
            alice_decrypt_error = min(alice_decrypt_error, decrypt_err)
            print('SOCK_SEND')
            sock_send(alice_output_1)
```

```
    elif network == 'bob':
        print('SOCKET_RECV ==== \n')
        messages_sock = sock_recv(sock)
        print(messages_sock)
        feed_dict1 = {msg: messages_sock, key: keys}
        start_time = time.time()
        _, decrypt_err, bob_input_1, bob_output_1 = sess.run([bob_optimizer, decrypt_err_bob, bob_input, bob_output], feed_dict = feed_dict1)
        end_time = time.time()
        bob_decrypt_time = end_time-start_time
        bob_input_1 = np.round(bob_input_1, 2)
        bob_ouput_1 = np.round(bob_output_1, 2)

        print('test bob_input (***Cipher Text***) = \n', bob_input_1, 2)
        print('test bob_output (***Plain Text***)= \n',  bob_output_1, 2)
        bob_decrypt_error = min(bob_decrypt_error, decrypt_err)
    elif network == 'eve':
        _, decrypt_err = sess.run([eve_optimizer, decrypt_err_eve], feed_dict = feed_dict)
        eve_decrypt_error = min(eve_decrypt_error, decrypt_err)

    return alice_decrypt_error, alice_encrypt_time, bob_decrypt_error, bob_decrypt_time, bob_output_1, eve_decrypt_error
```

```
%matplotlib inline

def plot_results():
    """
    Plot Lowest Decryption Errors achieved by Alice, Bob, and Eve per epoch
    """
    sns.set_style("darkgrid")
    plt.plot(alice_errors)
    plt.plot(bob_errors)
    plt.plot(eve_errors)
    plt.legend(['alice', 'bob', 'eve'])
    plt.xlabel('Epoch')
    plt.ylabel('Lowest Decryption error achieved')
    plt.show()
```

```
import time
epochs = 200
#sess = tf.InteractiveSession()
sess = tf.Session()

#with tf.Session() as sess:
print('Starting Training Process... ')
start_time = time.time()
train(sess)
end_time = time.time()
print('Time taken for Training (seconds): ', end_time-start_time)
plot_results()

print('alice_errors_train = ', alice_errors)
print('bob_errors_train = ', bob_errors)
print('eve_errors_train = ', eve_errors)
```

```
import binascii

#Test the Neural Crypto model
start_time = time.time()
print( 'Testing Alice' )
alice_loss_test, alice_encrypt_time, _, _, _,_ = test('alice', sess)
print('alice_errors_test = ', alice_loss_test)
print( 'Testing Bob' )
_,_, bob_loss_test, bob_decrypt_time, bob_output_1,_ = test('bob', sess)
print('bob_errors_test = ', bob_loss_test)
print( 'Testing Eve' )
_, _,_,_,_,eve_loss_test = test('eve', sess)
print('eve_errors_test = ', eve_loss_test)
end_time = time.time()
print('Time taken for Testing (seconds): ', end_time-start_time)
```

```
#convert to ASCII
b1 = np.round(bob_output_1, 1)
b1 = b1.ravel();
b1 = np.abs(b1)
print('bob_ouput_1 == ', b1)
print('\n')
b2  = np.array2string(b1)
b2 = b2.strip('[')
b2 = b2.strip(']')
b2 = b2.replace(" ", "")
b2 = b2.replace(".", "")
b2 = b2.replace("\n", "")

b2 = '0b'+b2
print('b2 = ', b2)
n = int(b2,2)
str2 = n.to_bytes((n.bit_length() + 7) // 8, 'big').decode(errors='ignore')
print(str2)

print('Bob recovered Plain Text = ', str2)
```