Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
завідувачка кафедри кібербезпеки
та захисту інформації
_____Наталія ЛУКОВА-ЧУЙКО
«14» червня 2022р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

дипломної роботи
бакалавра
(назва освітнього ступеня)

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

спеціальність 125 Кібербезпека
(код і назва спеціальності)

освітня програма Кібербезпека
(назва освітньої програми)

на тему: «Програмний модуль забезпечення конфіденційності в глобальній мережі»

Виконавець: студент IV курсу, групи КБ-42

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

	Прізвище, ініціали	**Підпис**
Керівник	Лариса МИРУТЕНКО	
Нормоконтроль	Олександр ТОРОШАНКО	

Київ 2022

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
завідувачка кафедри кібербезпеки
та захисту інформації
_____Наталія ЛУКОВА-ЧУЙКО
«01» листопада 2021 р.

ЗАВДАННЯ
на виконання дипломної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньої програми)

Студентові **КБ-42** (група) **Холостенку Олексію Валентиновичу** (прізвище ім'я по-батькові)

Тема дипломної роботи Програмний модуль забезпечення конфіденційності в глобальній мережі

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема дипломної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №5 від 29.10.2021 р.

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Протоколи, архітектури та засоби забезпечення захисту у віртуальних приватних мережах (VPN). Алгоритми шифрування даних, обміну ключами та автентифікації.

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Сучасні протоколи роботи VPN мереж, їх архітектура, завдання та призначення, методи забезпечення захисту інформації даних. Актуальні стандарти алгоритмів блокового симетричного шифрування. Програмна реалізація VPN мережі із використанням БСШ для забезпечення конфіденційності даних.

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність Реалізація програмного модулю забезпечення конфіденційності даних, що використовує сучасний алгоритм БСШ.

5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання: 29.10.2021 року

Завдання видав _____ (підпис) Лариса МИРУТЕНКО (ініціали, прізвище)

Завдання прийняв до виконання _____ (підпис) Олексій ХОЛОСТЕКНО (ініціали, прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів робіт	Строки виконання робіт (початок-кінець)	Відмітка про виконання
1	Постановка задачі	29.10.2021 – 28.01.2021	*виконано*
2	Аналіз відкритої літератури	29.01.2021 – 10.02.2021	*виконано*
3	Розгляд архітектури VPN	11.02.2021 – 23.02.2021	*виконано*
4	Дослідження основних методів захисту інформації у VPN	24.02.2021 – 24.03.2021	*виконано*
5	Розгляд і аналіз сучасних алгоритмів БСШ	25.03.2021 – 08.04.2021	*виконано*
6	Вибір стеку технологій	09.04.2021 – 21.04.2021	*виконано*
7	Реалізація тунельного з’єднання	21.04.2021 – 06.05.2021	*виконано*
8	Впровадження засобів шифрування трафіку та алгоритмів обміну ключовими даними	07.05.2021 – 18.05.2021	*виконано*
9	Реалізація механізму автентифікації	19.05.2021 – 01.06.2021	*виконано*
10	Оформлення пояснювальної записки	02.06.2021 – 05.06.2021	*виконано*
11	Підготовка до захисту	06.06.2021 – 13.06.2022	*виконано*

Завдання видав _____ (підпис) Лариса МИРУТЕНКО (ініціали, прізвище)

Завдання прийняв до виконання _____ (підпис) Олексій ХОЛОСТЕКНО (ініціали, прізвище)

Термін подання дипломної роботи до ЕК 06 червня 2022 року

РЕФЕРАТ

Пояснювальна записка дипломної роботи складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел та додатків. Основний текст займає 75 сторінок, включає в себе зміст, вступ, три розділи дипломної роботи, висновки, список джерел та 5 додатків із загальною кількістю сторінок 16. У пояснювальній записці дипломної роботи міститься 10 рисунків і 6 таблиць.

Метою роботи є розробка програмного модулю забезпечення конфіденційності в глобальній мережі.

Для досягнення зазначеної мети поставлено наступні завдання:

- дослідити архітектуру VPN-мережі та особливостей її побудови;
- провести аналіз сучасних протоколів роботи VPN-мережі;
- провести аналіз актуальних стандартів симетричного шифрування;
- побудувати програмну реалізацію VPN-мережі із підтримкою визначених в ході роботи критичних засобів захисту інформації.

Об'єктом дослідження є процес забезпечення конфіденційності інформації при її передачі через мережу інтернет.

Предметом дослідження є набір засобів забезпечення конфіденційності інформації при її передачі мережею інтернет.

Практичною цінністю отриманих результатів є порівняння сучасних протоколів роботи VPN, актуальних алгоритмів симетричного блочного шифрування, реалізація захищеного каналу передачі інформації через мережу інтернет із простою архітектурою і модульною структурою. Отримані в ході виконання даної роботи результати можна використати для подальшого аналізу використаного криптоалгоритму, VPN-мереж, як засобу захисту даних, передаваних через інтернет, а також в якості навчального макету.

Ключові слова: VPN-мережа, захищений канал зв'язку, DSTU7624, передача даних через інтернет, забезпечення конфіденційності даних, цілісність даних, квантова криптостійкість.

ЗМІСТ

ВСТУП

На сьогоднішній день існують високі потреби в захищеності передачі даних на рівні підприємств та державних установ. Так, уряд США, наприклад, використовує сертифікований NIST 256-бітний AES-шифр [1].

Водночас стрімко розвиваються квантові технології, в які інвестуються мільярди доларів [2], тим самим підвищуючи актуальність розвинення методів постквантової криптографії [3].

Сама потреба у встановлені максимально захищеного каналу зв'язку обумовлена ще й тим, що останнім часом як в Україні, так і у світі для багатьох компаній можливість забезпечення віддаленої роботи, за якої співробітникам необхідно отримувати доступ до внутрішніх мережевих ресурсів компанії із глобальної мережі, стала питанням виживання компанії, через карантинні обмеження [4, 5], та війну.

У сфері кіберзахисту подібну задачу вирішують завдяки віртуальним приватним мережам (VPN), що широко використовуються в корпоративному середовищі [6] та державних установах [7, 8].

Слід зауважити, що існує не мало досліджень самої VPN мережі, що описують її архітектуру, принципи роботи, та актуальні протоколи. Серед вітчизняних дослідників, що внесли вклад в дану тему, можна виділити: Турчинова О .О., Репецького Б. С., Брайко В. В. Серед іноземних: Ajay K. Sood, J. Myles Powell та ін.

Метою роботи є забезпечення максимальної конфіденційності, а також цілісності та автентичності даних, при передачі їх мережею VPN. Завданням практичної направленості буде побудова програмного додатку, що реалізує VPN-мережу.

РОЗДІЛ 1
АНАЛІЗ АРХІТЕКТУРИ VPN

1.1 Визначення і призначення VPN мережі

Віртуальна приватна мережа (VPN) – один із засобів безпечного розширення корпоративної мережі компанії, використовуючи існуючу структуру глобальної мережі, такої, як Інтернет. Використовуючи VPN компанія здатна контролювати мережевий трафік, забезпечуючи при цьому такі функції безпеки, як автентифікація та конфіденційність даних [9].

Використовуючи VPN можна встановити безпечне з'єднання між будь-якою кількістю хостів і шлюзів. Для забезпечення захищеності з'єднання між кінцевими точками у VPN використовуються такі механізми, як автентифікація користувачів, криптографічний захист інформації шляхом шифрування тощо.

VPN реалізується на мережевому рівні моделі багаторівневого стеку комунікацій TCP/IP. Зокрема, невід'ємною частиноюVPN є реалізація Internet Protocol Security (IPSec). IPSec призначений для забезпечення основних функцій безпеки у глобальній мережі та надає гнучкі примітиви, що можуть бути використані для створення надійних та гнучких віртуальних приватних мереж.

Окрім мережевого рівня, VPN може бути також реалізований і на другому рівні моделі OSI, для чого використовується тунельний протокол другого рівня L2TP, що призначений для створення з'єднань типу «віртуальна лінія». Даний спосіб з'єднання є досить економічним, та дозволяє ефективно отримувати доступ до віддалених клієнтів, завдяки тому, що управління IP-адресами віддається серверу приватної мережі. При використанні L2TP разом із протоколом IPSec можна досягти високого рівня безпеки у створеному каналі зв'язку.

Правильне налаштування VPN мережі є вкрай важливим, оскільки ця технологія є комплексною, та має багато складових, неправильне використання яких може суттєво підвищити ризики компрометації каналу зв'язку і кінцевих систем.

За призначенням та фактичною структурною різницею виділяють два типи реалізацій віртуальних приватних мереж: Remote Access (віддалений доступ) і Site-to-Site [10]. VPN з віддаленим доступом (Remote Access) використовується для надання віддаленим користувачам безпечного каналу зв’язку, по якому вони можуть взаємодіяти із внутрішньою мережею компанії та використовувати її ресурси. Віддалені співробітники можуть знаходитись в будь-якому місці поза приватною мережею компанії, оскільки усі дані, що вони будуть надсилати та отримувати із корпоративного центру обробки даних, шифруються VPN-сервером, тому надійно захищені при передачі. Слід зауважити, що трафік шифрується VPN-сервером лише в рамках тунелю, що встановлюється для безпечного з’єднання, та фактично зберігається на пристрої віддаленого користувача у відкритому вигляді, поза межею корпоративної мережі, тому для захисту даних слід також забезпечити захист даних на самому пристрої (шифрування жорсткого диску, антивірусні програми) і бажано встановити DLP-систему на межі безпечного периметру приватної мережі, щоб унеможливити «витік» надто чутливих даних за межі корпоративної мережі, де ризик її потенційної компрометації значно вищий. Саме завдяки таким рішенням можна створити тунель, що буде фактично приватним, навіть попри те, що дані передаються загальнодоступною глобальною мережею.

Попри свої переваги VPN з віддаленим доступом має і недоліки, пов’язані із складністю масштабування даного рішення, оскільки велика кількість віддалених клієнтів та хмарних додатків може бути важкою для підтримки та значно збільшує площину атак на приватну мережу компанії. Тобто при компрометації одного із віддалених клієнтів зловмисник може отримати доступ до певної частини внутрішньої мережевої інфраструктури компанії. Через це велика кількість таких віддалених робочих місць може підвищити ризики для безпеки самої приватної мережі. Це також доводить, що неправильне налаштування та використання VPN мереж може стати причиною зростання ризиків інформаційної безпеки компанії, тому як для компенсації проблем пов’язаних із VPN організації часто впроваджують певні компромісні рішення, що не задовольняють вимогам до безпеки інформації.

Тому для ефективного використання такої архітектури організаціям необхідно впроваджувати її згідно усіх рекомендацій щодо мережевої безпеки та оптимізувати її роботу при великій кількості користувачів, уникаючи компромісів з безпеки.

Site-to-Site VPN. Дана архітектура відрізняється від «віддаленого доступу» в першу чергу своїм призначенням. Якщо VPN віддаленого доступу призначений для надання доступу окремим користувачам до внутрішньої мережі компанії з будь-якої точки мережі, то VPN «сайт-сайт» використовується для об'єднання фізично віддалених мереж. Так декілька географічно віддалених офісів або філій можуть бути безпечно об'єднані в одну VPN мережу, використовуючи при цьому загальнодоступні канали зв'язку глобальної мережі інтернет. Оскільки увесь трафік буде передаватися в зашифрованому вигляді, користувачі зможуть безпечно обмінюватись даними і використовувати віддалені ресурси корпоративної мережі, при цьому користуючись нею як однією локальною мережею [11].

Дана архітектура сильно відрізняється за призначенням, однак можна сказати, що є і більш захищеною, адже інформація не тільки шифрується при передачі, але й в цілому не знаходиться у відкритому вигляді ніде більше, окрім самої приватної мережі, що, як правило, знаходиться всередині захищеного периметру, над яким компанія має значно більший контроль, ніж над віддаленим працівником.

1.2 Основні складові і вимоги до VPN

Для забезпечення інформаційної безпеки у віртуальних приватних мережах використовують такі протоколи TCP/IP рівня:

Протокол IP-безпеки. IP Security (IPSec) забезпечує стабільну та довговічну основу для забезпечення безпеки на мережному рівні. Більш детальний огляд протоколу наведений в підрозділі 1.3 [12].

Керування ключами. Для забезпечення захищеності каналів зв'язку від компрометації та значного зменшення площини для атак на ключі алгоритмів шифрування використовується протокол Internet Key Exchange (IKE) для управління ключовими даними та їх безпечного обміну через певні проміжки часу. Завдяки

цьому протоколу для кожного з’єднання користувачі використовують новий унікальний ключ, що значно підвищує складність зламу шифротексту і безпеку відповідно [13].

IKE версії 2 – вдосконалення протоколу обміну ключами в Інтернеті. Більш детальний огляд протоколу наведений в підрозділі 1.3.

Тунельний протокол другого рівня. L2TP (Layer 2 Tunneling Protocol) – протокол тунельних з'єднань (віртуальних лній), що дозволяє встановлювати ефективний доступ між віддаленими кінцями тунелю. Дозволяє керувати IP-адресами користувачів всередині корпоративної мережевої інфраструктури. Так, при використанні L2TP для побудови каналу зв’язку, поєднавши даний протокол із IPsec, можна забезпечити безпечний доступ до віддаленої системи чи мережі [14].

Трансляція мережевих адрес для VPN. VPN NAT – трансляція мережевих адрес у віртуальній приватній мережі. Трансляція адрес у VPN NAT відбувається перед тим, як застосовуються міри безпеки, визначені протоколами IKE та IPSec, на відміну від традиційного NAT.

NAT (трансляція мережевих адрес) використовується та перетворення приватних IP-адрес в загально доступні для економічного їх використання. Саме дана технологія забезпечує можливість кінцевих хостів в приватній мережі встановлювати зв’язок із віддаленими кінцями тунелю через Інтернет, отримувати доступ до віддалених ресурсів приватної мережі, тощо.

Крім того, якщо ви використовуєте приватні IP-адреси, вони можуть конфліктувати з подібними вхідними IP-адресами. Це можливо, наприклад, будь-коли ви хочете зв’язатися з іншою мережею, але обидві мережі використовують адресу типу 10. *.*.*.*, що призводить до зіткнення адрес і відкидання всіх пакетів. Застосування NAT до ваших вихідних адрес може здатися вирішенням цієї проблеми. Однак, якщо трафік даних захищений VPN, звичайний NAT не працюватиме, оскільки він змінює IP-адреси асоціацій безпеки (SA), які потрібні для роботи VPN. Щоб уникнути цієї проблеми, VPN пропонує власну версію трансляції мережевих адрес під назвою VPN NAT. VPN NAT виконує трансляцію адреси перед

перевіркою SA, призначаючи адресу підключення, коли з’єднання починається. Адреса залишається пов’язаною із з’єднанням, доки ви не видалите з’єднання.

IPSec з підтримкою NAT та UDP [9]. Для проходження трафіку, що захищений протоколом IPSec, через звичайний пристрій NAT використовується інкапсуляція із використанням протоколу UDP. Проблема полягає в тому, що звичайний NAT перешкоджає роботі VPN-мережі. Технологія трансляції мережевих адрес (NAT) використовується для приховання внутрішні IP-адреси приватної мережі за набором публічних IP-адрес дозволяє приховати ваші незареєстровані приватні IP-адреси за набором зареєстрованих IP-адрес, що також може бути засобом захисту внутрішньої мережі, оскільки її структура повністю прихована. Також, основною властивістю NAT є зменшення необхідної кількості публічних IP-адрес, що вирішує проблему з їх дефіцитом, адже фактично за однією IP-адресою може бути прихована повноцінна мережа компанії із великою кількістю кінцевих точок, своєю власною сегментацією і рішеннями з безпеки.

Проте традиційний NAT не має підтримки роботи із пакетами IPSec, оскільки звичайний NAT змінює адресу призначення в пакеті після його проходження через пристрій NAT і тому в даному випадку адреса призначення перестає бути валідною. Через таку маніпуляцію із пакетом він не зможе дійти до кінцевого користувача VPN мережі і пакет буде відкинуто, що повністю руйнує з’єднання.

Для вирішення цієї проблеми використовується інкапсуляція із використанням протоколу UDP. Пакет IPsec упаковується із використанням інкапсуляції у новий UDP пакет із подвійним IP/UDP заголовком. Так при проходженні звичайного NAT пристрою транслюється адреса лише самого IP заголовка, після чого пакет направляється до кінцевого користувача VPN мережі, що може відкинути зайві IP/UDP заголовки та залишити лише сам IPSec пакет із усією необхідною інформацією.

Інкапсуляція UDP може бути застосована лише для віртуальних приватних мереж, які використовують IPSec протокол у тунельному або транспортному режимах. Крім того за даної конфігурації кінцеві користувачі зможуть

встановлювати з’єднання лише за UDP-протоколом, через те що усі дані необхідно буде інкапсулювати в UDP-пакети.

Інкапсульований пакет через мережу надсилається на інший кінець віртуальної приватної мережі до UDP-порту 4500. Безпечний обмін ключовими даними зазвичай виконується із використанням UDP-порту 500. Однак, коли IKE реєструє NAT під час узгодження ключа, наступні пакети IKE надсилаються через вихідний порт. Також важливо забезпечити правильне налаштування фільтрації мережевого трафіку, згідно якому порт 4500 має бути відкритим, оскільки через нього відбувається з’єднання. Відрізнити пакет IPSec, що інкапсульовано в UDP, від пакету IKE можна за допомогою спеціального формату пакету UDP, оскільки якщо в ньому інкапсульований IKE пакет, у початок корисного навантаження UDP пакету записуються нуль, довжиною 4 байти. Підтримка інкапсуляції UDP обов’язкова для обох кінцевих користувачів для успішного встановлення з’єднання.

IP-стиск. IPComp – протокол що використовується для зменшення розміру корисного навантаження IP пакету, завдяки чому може бути суттєво зменшений розмір самої дейтаграми IP методом її стиснення, що поліпшує швидкість передачі інформації в межах встановленого VPN з’єднання.

Задачею даного протоколу є підвищення продуктивності з’єднання при передачі даних у повільних або сильно навантажених мережах. Стиснення пакету не забезпечує підвищення безпечності з’єднання і має використовуватися у поєднанні із алгоритмами шифрування трафіку.

Інтернет-інженерна робоча група (IETF) офіційно визначає IPComp у запиті на коментарі (RFC) 2393, протокол стиснення корисної інформації IP (IPComp) [15].

VPN та IP-фільтрація. Більшість з’єднань у віртуальних приватних мережах вимагають правил IP-фільтрації для належної роботи. Характер даних правил залежить як від типів з’єднань, що будуть реалізовані в рамках VPN-мережі, так і від типу трафіку, що буде проходити через канали зв’язку створені в даній мережі. Таким чином для кожного з'єднання слід встановлювати власну політику фільтрації, що буде визначати IP-адреси, протоколи і проти, що використовують конкретне з’єднання в мережі. Якщо з’єднання має підтримувати протокол Internet Key

Exchange (IKE), зазвичай може бути необхідним встановити окремі правила що явно будуть дозволяти процес IKE в рамках даного з’єднання. Більш ефективним та безпечним рішенням буде дозволити VPN автоматично формувати подібні правила, оскільки це дозволить зменшити ризик помилки при налаштуванні власноруч, що може бути причиною різкого зменшення захищеності з’єднання.

VPN має основні складові (наведені в пунктах 1.2.8 - 1.2.12) призначені для відповідності основним вимогам до інфраструктури VPN щодо захисту інформації.

Шифрування даних (конфіденційність). Слід зазначити, що для захисту даних від стороннього прослуховування при їх передачі усі дані мають бути зашифровані. Таким чином необхідно визначити деякий оптимальний за швидкодією та стійкістю алгоритм шифрування даних та застосовувати його для кожного повідомлення, відправленого у VPN-мережі. Крім того важливо захистити не лише саме повідомлення, але й сам алгоритм шифрування та його секретні дані, такі, як ключ, що можна зробити завдяки використанню протоколів безпечного обміну ключовими даними, такого як DiffieHellman на еліптичних криви. Також даний алгоритм має забезпечувати відсутність будь-якої зв’язаності між поточними та майбутніми ключами.

Цілісність даних. Однією із ключових вимог до VPN мережі є забезпечення захищеності даних від неавторизованих змін їх вмісту. Для виконання даної вимоги може бути застосований такий механізм забезпечення цілісності даних як генерація гешованого коду аутентифікації повідомлення (HMAC або імітовставка), що дозволить захистити повідомлення від несанкціонованих змін.

Аутентифікація. Для процесу надійного визначення ідентичності кожного із кінцевих користувачів VPN-мережі можуть бути використані надійні механізми аутентифікації, одним з яких може бути використання системи відкритих та закритих сертифікатів. Сам метод полягає у формуванні спільного секрету кожного для кожного із двох кінцевих користувачів з’єднання. Спільний секрет утворюється за допомогою публічного (відкритого) сертифікату, який може бути переданий у відкритому вигляді будь-ким способом, та приватного (закритого) сертифікату, що має зберігатися лише у користувача-власника даного секрету, бажано – в

зашифрованому вигляді. Даний спосіб аутентифікації вважається дуже надійним, та значно кращий за аутентифікацію паролем, за якої секретні дані мають зберігатися на обидвох кінцях з'єднання.

Централізований транспортний потік. Загалом дана рекомендація не є обов'язковою, однак має свої переваги. При використанні такої архітектури VPN-мережі, за якої увесь трафік, що направлений за межу захищеного периметру, проходить через певний шлюз, можна досягти контролю над усім «зовнішнім» трафіком мережі. Завдяки цьому, на даному шлюзі (сервері) можна розгорнути повноцінні додаткові заходи із забезпечення інформаційної безпеки, що замкнуть захищений периметр приватної мережі. Прикладами таких додаткових заходів можуть бути: налаштування фільтрації усього трафіку із використанням мережевого екрану, розгортання антивірусного екрану та інших сканерів вмісту пакетів, впровадження DLP-рішень, використання IDS/IPS систем для зменшення ризику компрометації мережі.

При такій архітектурі доречним буде використання концентраторів, щоб знизити вартість загального рішення, усунувши необхідність реплікації системи захисту периметра компанії на кожному шлюзі, підключеному до Інтернету. Забезпечуючи передачу всього трафіку, пов'язаного з Інтернетом, через шлюзи вузлів, компанія може скористатися перевагами перевірки стану, фільтрації вмісту, антивірусних та інших механізмів захисту в усій організації. Жоден трафік не повинен проходити зовнішні шлюзи у незашифрованому вигляді. Це також дає перевагу наявності центрального пункту для реєстрації доступу до Інтернету.

Крім того дана архітектура знижує рівень вимог до самих VPN-пристроїв дозволяючи безпечно масштабувати VPN-мережу із затратою менших ресурсів. Обумовлена така особливість даної архітектури тим, що завдяки ній на VPN-пристроях достатньо налаштувати відкидання всього трафіку, окрім того, що необхідний для створення VPN тунелів (IKE, IPSec і адміністративний трафік). Також важливим налаштуванням для таких мереж є заборона пропускання трафіку з мережі будь-яким іншим способом, окрім як передачі даних через VPN-шлюз, для унеможливлення витоку даної інформації побічними каналами.

Дана архітектура має невелику кількість недоліків, проте вони є значущими та мають бути прийняті до уваги про побудові на її основі VPN-мережі. Одним з таких недоліків є можливе зменшення пропускної здатності VPN-мережі, що напряму залежить від пропускної здатності VPN-шлюзу, котра в свою чергу базується на потужності шлюзу (швидкодії шифрування) та фактичній пропускній здатності мережевого обладнання, що встановлене на даному шлюзі. Необхідна мінімальна потужність шлюзу має розраховуватись з огляду на максимальну кількість користувачів, що має обслуговувати даний шлюз одночасно. Ще одним з можливих недоліків є те, що компрометація VPN-шлюзу може значити компрометацію кожної із захищених сесій на момент зламу. Рішенням має стати посилений захист шлюзу із використанням принципу мінімального необхідного доступу до даного ресурсу користувачам.

Доступність і резервування. Для шлюзів забезпечення високого рівня доступності є обов'язковою необхідністю. Таким чином усі пристрої та сервіси, що необхідні для роботи даного шлюзу, мають бути налаштовані та побудовані з метою забезпечити в тому числі і максимально можливу продуктивність. Звичайно доступність має бути забезпечена для кожного із пристроїв і ресурсів у всій VPN-мережі, одна порушення доступності на шлюзі може спричинити порушення доступності критично значного за розміром сегменту, що обслуговується даним шлюзом.

Як додатковий засіб забезпечення доступності у рамках VPN-мережі можна впровадити резервні шлюзи для підключення до VPN та Інтернет мережі, із написанням спеціальних правил маршрутизації у випадку мережевих збоїв.

1.3 Протоколи роботи VPN

PPTP – протокол тунелювання точка-точка. Даний протокол був розроблений раніше за всіх розглянутих в даному розділі та використовується до сих пір. Хоча він вважається застарілим, розгляд його роботи дозволить визначити основі ознаки VPN-протоколів. Цей протокол є запатентованою розробкою компанії Microsoft [16].

Для роботи даного протоколу між клієнтом та сервером встановлюється два з’єднання. Перше з’єднання використовується для керування, та використовує TCP протокол для своєї роботи із портом сервера 1723. Друге з’єднання відбувається з використанням протоколу GRE – транспортним протоколом обміну даними, прикладами якого можуть бути TCP або UDP. Через використання даного протоколу клієнти, що знаходяться за мережею NAT можуть зіштовхнутися із проблемою неможливості з’єднання із сервером, що має бути типу «точка-точка». Рішення цієї проблеми закладене в самому протоколі PPTP, який використовує розширену версію GRE, завдяки якій в заголовок пакету передається ідентифікатор виклику. Використовуючи цей заголовок NAT-маршрутизатори здатні ідентифікувати і визначити адресу призначення трафіку, що створений клієнтом і передається з локальної мережі через NAT за протоколом GRE до сервера і навпаки. Завдяки такій модифікації клієнти, що знаходяться в мережах які використовують NAT, здатні встановлювати з’єднання типу «точка-точка» із застосування протоколу GRE. Назва даної технології – VPN Pass Through. Протокол PPTP доступний для використання як в операційних системах Windows, так і в широкому спектрі інших платформ. Даний протокол має добру швидкодію, однак має низьку надійність та у порівнянні з іншими сучасними протоколами довго відновлює втрачене з’єднання. Також даний протокол вважається застарілим, тому сама компанія-розробник не рекомендує його використання. Таким чином даний протокол насьогодні не придатний для забезпечення конфіденційності передаваних даних та безпеки в цілому, тому може використовуватись лише для інших цілей, а не як інструмент безпеки, наприклад: розблокування вмісту з обмеженим за країною вмістом.

SSTP – протокол VPN що для забезпечення конфіденційності та в цілому безпеки даних використовує протокол SSL, передаючи дані протоколом TCP за стандартним портом 443. Дана властивість алгоритму дозволяє використовувати його в мережах із обмеженими можливостями, жорсткою фільтрацією, тощо. Основною платформою на якій використовується даний протокол є Windows, однак підтримуються також інші операційні системи. За продуктивністю цей протокол

можна оцінити як швидкий та стабільний. За рахунок використання SSL забезпечується високий рівень безпеки з’єднання [17].

IP Security (IPSec) – протокол, що забезпечує безпечне з’єднання забезпечує стабільну та довготривалу основу для безпеки на рівні мережі. Для забезпечення конфіденційності даних даний протокол підтримує та використовує широкий спектр криптографічних алгоритмів, що включає в себе всі актуальні насьогодні стандарти шифрування, а також здатний до масштабування та підтримки будь-яких нових стандартів. Протокол IPSec використовується для забезпечення таких важливих властивостей безпечного з’єднання [12]:

- Аутентифікація передаваних пакетів – забезпечується перевірка кожного пакету на те, чи був він дійсно відправлений тим, від кого очікувався.
- Цілісність даних – перевірка даних на предмет несанкціонованих змін або спотворень при передачі.
- Конфіденційність даних – дозволяє скрити вміст повідомлення при передачі ненадійними каналами зв’язку, як правило, шифруючи його.
- Захист від атак типу відтворення – унеможливлює повторне надсилання «валідного» пакету, що був перехоплений зловмисником.
- Автоматизація процедури управління ключовими даними та з’єднаннями – забезпечує незмінність політики віртуальної приватної мережі при її масштабуванні, та значно зменшує ризики неправильного, потенційно небезпечного налаштування.

Для забезпечення безпеки у віртуальній приватній мережі IPSec зазвичай використовують іще з одним протоколом, кожен з яких призначений для:

- IPSec – визначає заголовок аутентифікації (AH) та інкапсуляцію корисного навантаження безпеки (ESP).
- Internet Key Exchange (IKE) – забезпечує автоматизацію безпечного обміну ключовими даними, забезпечуючи узгодження безпечних з’єднань, а також оновлення ключів для шифрування трафіку протоколом IPSec.

Спеціальна група з розробки Інтернету (IETF) офіційно визначає IPSec у Запиті на коментар (RFC) 4301, Архітектура безпеки для протоколу Інтернету.

Комплексне рішення VPN має забезпечувати не лише конфіденційність інформації а й додатковий захист безпечних з'єднань, наприклад, використовуючи протокол Internet Key Exchange (IKE), що призначений для керування ключовими даними. Завдяки цьому протоколи клієнти VPN мережі, розташовані на різних кінцях з'єднання, мають можливість безпечно узгоджувати оновлені з часом ключові дані, що значно підвищує безпеку самого алгоритму шифрування [13].

Таким чином завдяки використанню даного протоколу значно зменшується ризик компрометації зловмисником безпечного каналу зв'язку та порушення конфіденційності інформації. Для того, щоб оновлення ключів було ефективним, згідно протоколу необхідно забезпечити відсутність будь-якого зв'язку між поточним та оновленим ключем, щоб унеможливити здатність зловмисника для прогнозування наступного ключа, шляхом аналізу перехопленого трафіку.

Генерація та узгодження криптографічних ключових даних згідно протоколу керування ключами VPN (IKE) має бути реалізована повністю автоматичною, задля зменшення ризиків, пов'язаних із неправильним ручним налаштуванням а також для підвищення продуктивності даного засобу захисту.

Окрім безпечного обміну ключами, IKE забезпечує узгодження даних підключення безпеки (SA), що визначає усю інформацію, використовувану протоколами IPSec протоколами. Таким чином завдяки SA відбувається ідентифікація самих алгоритмів шифрування, що використовуються для забезпечення конфіденційності даних, довжину ключів і термін їх використання, дані про кінцевих користувачів для аутентифікації, дані про способи інкапсуляції, що використовуються в рамках з'єднання.

Від криптографічних ключів шифрування даних залежить степінь захищеності безпечного каналу зв'язку, оскільки компрометація ключових даних означає повну компрометацію захищеного з'єднання і нівелює цінність усіх засобів забезпечення його безпеки, такі як автентифікація, шифрування, забезпечення цілісності, тощо, незважаючи на їх ефективність. З цього випливає, що забезпечення безпечного створення ключових даних є одним із вкрай важливих факторів для створення дійсно безпечного і приватного каналу зв'язку.

Керування ключовими даними у віртуальних приватних мережах, що реалізують протокол IKE відбувається у два етапи:

Перший етап полягає у безпечному обміні початковим секретним ключем, що буде використаний для отримання користувачами наступних секретних ключів при оновлені. Оскільки на момент обміну початковим ключем ще не створений захищений канал, для його передачі на обидва кінця з'єднання, попередньо автентифіковані із застосуванням паролів або сертифікатів, використовуються алгоритми асиметричного шифрування даних або попередній обмін секретами із застосуванням інших безпечних каналів передачі інформації. Також автентичність забезпечується не лише для учасників з'єднання, а й для самих даних, використовуючи або підписи, або спеціальні ключі призначені лише для забезпечення автентичності.

Одним із варіантів початкового ключа є попередній спільний ключ, що представляє собою випадковий рядок бітів розмір якого не перевищує 128 біт. Цей спільний секрет узгоджується обидвома користувачами з'єднання, і його перевагою є простота використання, однак він має і недолік: подібний секрет розповсюджується за межами встановлюваного безпечного з'єднання, що не є безпечним за замовчуванням, тому безпека самого з'єднання в такому випадку залежить від сторонніх факторів, які набагато важче контролювати. Такий спосіб обміну початковим ключем за стійкістю можна порівняти із використанням паролю.

Використання сертифікатів для забезпечення обміну ключовими даними а також автентичності кожного повідомлення є значно більш безпечним засобом обміну спільними секретами. Самі сертифікати можуть бути створені як власноруч, так і автоматично, а головним при їх створенні є безпечне зберігання приватних сертифікатів на кожному клієнтському хості. Через свою надійність, сертифікати використовуються в багатьох рішеннях VPN як стандартні методи автентифікації. Ще однією перевагою даного методу є забезпечення легкості масштабованості рішення, оскільки генерація сертифікатів може бути легко автоматизована, та передаватися використовуватися в рамках будь-якого конкретного з'єднання, на відміну від попередньо виділених ключів. Як додаткова міра безпеки або при

великих масштабах VPN-мережі, яка містить в собі велику кількість користувачів, для зберігання та обміну сертифікатами використовується центр сертифікації, що є спільним довіреним ресурсом серед усіх користувачів.

Другий етап полягає в узгодженні усієї необхідної інформації для захисту фактичного з'єднання та безпечного передавання самих даних. Цією інформацією, наприклад, є посилання безпеки (SA) та подальші ключові дані. Важливо зазначити, що жодні дані, окрім необхідних для роботи протоколу IKE, не передаються до завершення перших двох його етапів роботи. Дані, що передаються під час другого етапу роботи захищені із використанням спільного секрету, що був розподілений під час виконання першого етапу.

Після завершення другого етапу VPN мережа має можливість встановити саме безпечне з'єднання між кінцевими клієнтами, що були попередньо визначені та автентифіковані. Дані в рамках цього з'єднання будуть захищені за тим ступенем безпеки та передаватися із тією ефективністю, що були визначені під час роботи першого та другого етапів роботи ключових серверів.

Для регулярного оновлення ключових даних та перевірки автентичності користувачів при довготривалих з'єднаннях, перший та другий етапи роботи протоколу IKE два необхідно повторювати. Для першого етапу роботи може бути застосований інтервал повторювання в один день, другий етап має повторюватись значно частіше, від 60 хвилин, до 5. Менші інтервали повторів першого та другого етапів підвищують захищеність з'єднань, однак можуть значно знизити ефективність передачі даних, тому слід визначати компроміс між продуктивністю системи та забезпечуваним рівнем безпеки. Для найбільш важливих конфіденційних даних слід встановлювати найменші оптимальні інтервали.

Кожен з етапів виконується автоматично, однак можуть бути налаштовані основні параметри, що визначають режими роботи цих з'єднань, утворюючи політику IKE. Прикладом такого налаштування може бути визначення максимального та мінімально інтервалу повторення кожного з двох етапів, згаданих вище. Налаштування даної політики слід проводити визначивши вимоги до створюваного каналу безпечного зв'язку.

IKE версії 2 — є вдосконаленням першої версії протоколу обміну ключовими надими через інтернет (IKE), що була розроблена командою (RFC4306), та мало на меті покращення процедури обміну ключовими даними і процесу автентифікації кінцевих користувачів захищеного з'єднання в рамках VPN мережі. Даний протокол спрощує передачу даних, необхідних для своєї роботи, та запроваджує заходи що усувають відомі вразливості першої версії протоколу. Як і протокол першої версії, вдосконалена його версія працює в два етапи.

Першим етапом роботи IKEv2 є IKE_SA, що складається з декількох повідомлень IKE_SA_INIT. IKE_SA можна порівняти з перим етапом роботи IKEv1. Параметри роботи першого етапу IKE_SA можуть бути визначені в політиці обміну ключовими даними [13].

Другим етапом IKEv2 є CHILD_SA. Перший CHILD_SA є парою повідомлень IKE_AUTH. Цей етап роботи IKEv2 можна порівняти з другим етапом IKEv1. Для процедури оновлення ключових даних можуть бути надіслані додаткові пари повідомлень CHILD_SA, що можуть бути використані також і для обміню інформаційними повідомленнями. Параметри роботи даного етапу також можуть бути визначені у політиці обміну ключовими даними.

На відміну від IKEv1 інтерфейс IKEv2 є більш простим та ефективним. Це обумовлено наступними відмінностями:

В рамках першого етапу роботи IKEv1 можуть реалізовуватись два можливих режиму обміну даними: основний та агресивний, в той час як в IKEv2 для даного етапу існує лише обмін парою повідомлень IKEV_SA.

На другому етапі протоколу IKEv2 використовується простіший обмін двома парами повідомлень CHILD_SA, в той самий час IKEv1 вимагає принаймні однієї пари обмінної пари на другому етапі.

Попри відмінності, задачі роботи обидвох цих протоколів однакові, та результатом їх виконання є реалізація безпечного обміну ключовими даними, набору параметрів, зазначених в повідомленнях безпеки SA, що будуть використані для забезпечення захисту каналу зв'язку із застосуванням ESP та AH протоколів.

Протокол тунелювання другого рівня (L2TP) призначення для створення з’єднань типу віртуальних ліній зв’язку. Даний тип з’єднання є економічним та дозволяє приватним мережам компаній виконувати керування призначеними IP-адресами віддалених кінцевих користувачів. Для створення безпечного з’єднання із використанням даного протоколу необхідно використовувати його у поєднанні із засобами безпеки, що можуть бути забезпечені, наприклад, впровадженням протоколу IPSec [14].

Протокол створення віртуальних каналів зв’язку здатний працювати в двох режимах: обов’язковому та необов’язковою. Головною відмінністю двох даних протоколів є кінцеві точки з’єднання. При роботі в необов’язковому режимі роботи кінцем тунелю є віддалений користувач, із яким встановлюється з’єднання, а при роботі в обов’язковому режимі тунель закінчується на стороні L2TP сервера або маршрутизатора.

При роботі в обов’язковому режимі тунелювання за протоколом L2TP хост віддаленого клієнта ініціює з’єднання із маршрутизатором / сервером L2TP, котрий в свою чергу має встановити L2TP з’єднання із клієнтом, що знаходиться на іншому кінці тунелю, з чого випливає, що для цього режиму роботи маршрутизатор / сервер зобов’язаний підтримувати роботу за L2TP протоколом. Не дивлячись на те, що саме сервер / провайдер встановлює з’єднання з іншим кінцем тунелю, саме клієнт визначає параметри захисту з’єднання із використанням технологій VPN.

При роботі в необов’язковому режимі за протоколом L2TP тунель встановлюється напряму із віддаленим клієнтом на іншому кінці з’єднання, без використання проміжного L2TP сервера або маршрутизатора, що в даному випадку, не створюючи жодних тунелів, просто пересилають повідомлення за призначенням, з чого слідує, що за таких умов роботи від серверу чи маршрутизатору не потребується підтримка роботи за протоколом L2TP. При використанні L2TP поверх IPSec протоколу він придатний для створення VPN-мережі за архітектурою Site-to-site, що в даному випадку дозволяє завдяки використанню шлюзів об’єднати декілька віддалених мереж завдяки захищеним VPN-тунелям L2TP, розгорнутим між цими шлюзами.

За принципом своєї роботи L2TP можна назвати різновидом протоколу інкапсуляції, оскільки його тунель формується шляхом включення кадру L2TP в UDP пакет користувача, що в свою чергу, інкапсулюється в зовнішній IP-пакет при передачі зовнішньою мережею, необхідний для маршрутизації в рамках приватної мережі, тому він включає в себе адреси відправника та отримувача, що знаходяться на різних кінцях VPN тунелю. Через те, що зовнішнім пакетом для всіх інкапсульованих даних є IP-пакет, до нього застосовний протокол забезпечення безпечного з'єднання IPSec, що дозволяє захистити дані, які передаються створеним L2TP тунелем. При отримуванні та деінкапсуляції даних, користувачі можуть використовувати в повній мірі усі складові IPSec протоколу для забезпечення безпечності встановленого каналу зв'язку, такі як заголовок аутентифікації (AH) та інкапсульоване корисне навантаження (ESP). При сумісній роботі з IPSec, L2TP здатний також використовувати і сумісний IKEv2 протокол для забезпечення безпечного обміну ключовими даними.

OpenVPN — протокол реалізації віртуальної приватної мережі з відкритим вихідним кодом. Він є дуже поширеним та універсальним протоколом за рахунок високої гнучкості конфігурацій, а також підтримки широкого спектру криптографічних алгоритмів, оскільки для шифрування в даному протоколі використовується відкрита бібліотека OpenSSL. Даний протокол може працювати, використовуючи будь-який як TCP так і UDP порт [18]. Відкритість даного протоколу з урахуванням його поширеності, свідчить про надійність реалізації, оскільки він був протестований та перевірений великою кількістю незалежних експертів та користувачів. Також відкрита реалізація унеможливлює створення непомітних для користувачів засобів, що можуть відкрити захищене з'єднання в інтересах власника протоколу або за запитом держави. OpenVPN є стабільним та продуктивним протоколом VPN мережі, однак і в даного рішення є певні недоліки. Наприклад, на відміну від IKE та IPSec, підтримка яких зазвичай впроваджена на рівні операційної системи, OpenVPN для своєї роботи потребує встановлення спеціального програмного забезпечення як на клієнтській стороні, так і для реалізації OpenVPN-серверу. Однак даний недолік пом'якшується тим, що дане

клієнтське забезпечення є доступним для усіх основних платформ, таких як: операційні системи Windows, Linux, MacOS, Android, IOS, тощо.

WireGuard – це протокол VPN, який визначає як має встановлюватись та підтримуватись з'єднання клієнта із сервером для безпечної передачі даних між ними. Клієнтом можуть бути як десктопні комп'ютери, сервери так і мобільні пристрої, планшети [19]. WireGuard працює виключно за UDP протоколом, що не реалізую будь-яких handshake протоколів, і тому з'єднання і передача даних відбувається швидше, у порівнянні із тим же самим TCP, оскільки фактично відбувається менше запитів у мережу, які також необхідно скеровувати, контролювати.

Криптографічні рішення WireGuard вважаються дуже сучасними, серед них:

- Curve25519 – обмін ключовими даними;
- ChaCha20 – симетричний потоковий шифр для забезпечення конфіденційності передаваних даних;
- Poly1305 автентифікація даних, шляхом вироблення імітовставки;
- SipHash – формування ключових даних хеш-таблиці,
- BLAKE2 – хешування інформації.

WireGuard – дійсно унікальний VPN із відкритим вихідним кодом, що використовує нестандартні для ринку рішення, власну архітектуру, чітко визначений стек новітніх алгоритмів криптографічних перетворень. Серед переваг даного протоколу виділяють наступні:

Стабільність. В порівнянні з існуючими протоколами VPN (такими як OpenVpn, IPsec, SSL та інш.) WireGuard забезпечує швидке и стабільне з'єднання, має низький час відновлення втраченого підключення, і гарно проявляє себе у складних ситуаціях, таких як часті зміни IP адрес при використанні мобільного інтернету.

Стійкість. WireGuard використовує сучасні криптографічні примітиви із високою доказовою стійкістю, більшість налаштувань залишає за замовчуванням і має компактну кодову базу, не додаючи підтримку якомога більшої кількості алгоритмів шифрування, які було б важко підтримувати. Тільки потрібні на думку

розробників рішення впроваджуються в додатку, що дає можливість гарного контролю над якістю реалізації.

Швидкість. В основі WireGuard використовується швидкісний протокол шифрування даних, який також інтегрований у ядро операційної системи Linux, що значно поліпшує швидкість роботи додатку. До того ж використання лише UDP протоколу також відіграє свою роль в пришвидшенні передачі трафіку за даним VPN-протоколом.

Простота конфігурації. У порівнянні із іншими протоколами шифрування, що потребують певних компетенцій з питань роботи мережі, шифрування трафіку та формування сертифікатів, налаштування як клієнтського так і серверного додатку WireGuard є дуже простим. Дана властивість протоколу значно зменшує ризик помилки користувача при налаштуванні роботи VPN мережі, тому також впливає і на безпеку.

Але даний протоко також має і недоліки, серед яких:

- Необхідність встановлення додаткового програмного забезпечення для роботи додатку на деяких платформах.
- Не реалізує захист від аналізу інкапсульованих пакетів.

Для узагальнення отриманих в результатів досліджень відкритої інформації про актуальні протоколи VPN мереж, вони були внесені в таблицю 1.1. Ця таблиця була створена з метою:

- визначити найбільш стійкі протоколи, перелічивши відомі вразливості;
- визначити стек технологій, що використовується для реалізації даних алгоритмів, а саме протокол з’єднання, та алгоритми шифрування даних;
- зробити висновок про можливість глибокого дослідження реалізації протоколу на основі даних про його ліцензію: відкриту або приватну;
- на основі зібраних даних обрати протоколи, що можуть слугувати прототипами при реалізації власного VPN-додатку.

Таблиця 1.1

Порівняння протоколів за якими будуються VPN-мережі

Протокол	**Ліцензія**	**Шифрування**	**Порти**	**Вразливості**
PPTP	Запатентована	MPPE, RSA-RC4-128	TCP-1723	MSCHAP-v2 - атака за словником., алгоритм RC4 - атака Bit-flipping
SSTP	Запатентована	SSL	TCP-443	Критичних вразливостей не виявлено
L2TP + IPsec	Запатентована	3DES, AES	UDP-500, UDP-1701, UDP-5500	3DES вразливий до Meet-in-the-middle и Sweet32
IKEv2 + Ipsec	Запатентована	AES, Blowfish, Camellia і т.д.	UDP-500, UDP-4500	Критичних вразливостей не виявлено
OpenVPN	GNU GPL	OpenSSL	Будь-який UDP/TCP-порт	Критичних вразливостей не виявлено
WireGuard	GNU GPL	1-RTT, Curve25519, ChaCha20, Poly1305, BLAKE2s	Будь-який UDP-порт	Критичних вразливостей не виявлено

Висновки за розділом 1

На основі аналізу, проведеного в даному розділі, була сформована таблиця із узагальненою інформацією про сучасні протоколи віртуальних приватних мереж (таблиця 1.1).

Використовуючи результати досліджень в даному розділі будуть сформовані вимоги до властивостей та засобів захисту VPN мережі, що буде побудована в результаті реалізації програмного модулю забезпечення конфіденційності в глобальній мережі, згідно теми даної роботи. Головними критерієм при визначені даних властивостей є вплив їх застосування на рівень забезпечення конфіденційності даних у встановлюваному з’єднанні, а також підвищення загального рівня безпечності каналу зв’язку.

На основі досліджень даного розділу, можна сказати, що протокол роботи VPN з точки зору безпеки має реалізовувати наступні механізми, спрямовані на захист інформації:

- шифрування усього трафіку, що передається мережею та недопущення передачі даних у відкритому вигляді;
- реалізація безпечного алгоритму обміну ключовими даними;
- бажана реалізація оновлення ключових даних через певні проміжки часу;
- реалізація політики обміну ключовими даними;
- аутентифікація підключень до ресурсів внутрішньої VPN мережі;
- забезпечення автентичності кожного повідомлення, передаваного мережею;
- забезпечення цілісності даних, що передаються мережею;
- забезпечення доступності інформації (мережевого ресурсу);
- бажане визначення політики фільтрації трафіку на мережевому рівні;
- забезпечення доступності інформації шляхом використання оптимізованого алгоритму шифрування із достатньою швидкодією;
- захист з’єднання від атаки відтворення повідомлень;
- реалізація внутрішньої IP адресації.

Таким чином, за основу для реалізації програмного модулю забезпечення конфіденційності даних, будуть взяті властивості таких протоколів роботи VPN-мережі, як IPSec та IKE, оскільки їх реалізації повністю задовольняють вимогам, визначеним в даному розділі.

РОЗДІЛ 2
ПОРІВНЯЛЬНИЙ АНАЛІЗ СТАНДАНТІВ БСШ

2.1 Огляд алгоритмів БСШ

В даному розділі розглянуті сучасні алгоритми симетричного блокового шифрування, їх архітектура та властивості. Серед розглянутих алгоритмів представлені наступні: Blowfish, Twofish, RC6, SERPENT, MARS , DES, 3DES, AES, DSTU 7624 (Калина)

Blowfish. Автором алгоритму є Брюс Шнайер. З точки зору безпеки даний алгоритм вважається висококласним, має структуру та функціональність відмінну від тих алгоритмів, що будуть розглядатися далі. Blowfish забезпечує гнучке співвідношення швидкодії та безпеки завдяки змінній довжині ключа, його алгоритм є простим і компактним. Його реалізаціє є відкритою та не має патенту. Blowfish використовується для шифрування в протоколі SSL. За структурою Blowfish є симетричним блоковим алгоритмом в основі якого лежить мережа Фейстеля. Він має дві фази розширення ключа та власне фазу шифрування даних. Довжина блоку у Blowfish має розрядність 64. Алгоритм реалізовує 16 раундів шифрування, має змінну довжину ключа до 448 біт, що розкладається на 18 раундових ключів 32-бітної довжини, що можна реалізувати як на 32- так і на 64-бітних процесорах. Алгоритм використовує 4 Sboxes (таблиці перестановок). Для розшифрування шифрованого тексту використовується той самий алгоритм із використанням обернених операцій [20].

За структурою Twofish є симетричним блоковим алгоритмом в основі якого лежить мережа Фейстеля. Його реалізаціє є відкритою та не має патенту. Довжина ключа Twofish може приймати наступні значення: 128, 192 та 256 біт. Довжина блоку в даному алгоритмі є 128 біт. Даний алгоритм має оптимізацію для 32-розрядних процесорів. Може бути ефективно реалізованим як програмно так і апаратно. По суті даний алгоритм є наслідником алґоритму Blowfish [21].

Архітектурно даний алґоритм містить в собі такі операції, як використання SBoxses (блоки нелінійної перестановки) розміром 8 на 8 біт, Maximum distance separable (MDS – матриці дифузійних перетворень), забілювання та розширення ключа.

RC6 – симетричний блоковий шифр побудований за структурою Мережі Фейстеля. Авторами даного шифру є Рон Рівест, Метт Робшоу, Рей Сідні та Іцюнь Ліза Інь. Він походить від RC5. Реалізація даного алгоритму запатентована RSA Security. Блок даних, що обробляє RC6, має розмір блоку 128 біт. RC6 виконує шифрування із ключами, що мать розмір 128, 192 і 256 біт. RC6 має досить гнучку архітектуру, завдяки якій може бути налаштований для роботи із різними параметрами, що задаються на початку його роботи, серед яких: довжина блоку, розмір ключа і кількість раундів. RC6 для шифрування використовує зсув на змінну кількість біт, додавання за модулем та XOR [22].

Serpent – симетричний блоковий шифр побудований за структурою SP-мережі. Авторами шифру Росс Андерсон, Елі Біхам і Ларс Кнудсен. Реалізація Serpent є відкритою і не має патентів. Довжина блоку в даному алгоритмі 128 біт, а довжина ключа може приймати три значення: 128, 192 та 256 біт. Кількість раундів шифрування і дешифрування – 32. Для нелінійних перетворень в даному алгоритмі використовується 8 SBox (таблиць підстановки) [23].

MARS – симетричний блоковий шифр. Серед авторів даного шифру, а саме працівників виділеної команди компанією IBM для його розробки, можна виділити Дона Копперсміта, що розробляв DES. Метою розробки MARS було протиставити його майбутнім загрозам криптографії. Він має перестановку підвищеної безпеки. Шифр є досить швидким та гнучким [24]. MARS є симетричним ключовим алгоритмом, заснованим на мережі Фейстеля. Довжиною блока в алгоритмі є 128 біт. Для шифрування та дешифрування виконується 32 раунди, а довжина ключа може варіюватися від 128 до 448 біт (із кроком у 32). Для нелінійних перетворень MARS використовує один SBox (таблиця підстановок).

DSTU7624 або Калина – симетричний блоковий шифр, побудований на основі AES (Rijndael), тому, природно, що в його основі використовується SP мережа. На відміну від AES Калина підтримує роботу із блоками даних 128, 256 та 512 біт.

Довжина ключа також може бути 128, 256 та 512 біт відповідно, а кількість раундів може бути 10, 14 та 18. Також стандарт визначає 10 можливих режимів роботи [25]:

- Проста заміна (базове перетворення) (ECB).
- Гамування (CTR).
- Гамування зі зворотним зв'язком за шифротекстом (CFB).
- Вироблення імітовставки (CMAC).
- Зчеплення шифроблоків (CBC).
- Гамування зі зворотним зв'язком за шифрогамою (OFB).
- Вибіркове гамування із прискореним виробленням імітовставки (GCM, GMAC).
- Вироблення імітовставки і гамування (CCM).
- Індексована заміна (XTS).
- Захист ключових даних (KW).

DES – симетричний блоковий шифр, структурно заснований на мережі Фейстеля. Був першим алгоритмом, прийнятим як стандарт визнаний NIST. На сьогоднішній день дана версія алгоритму вважається застарілою. Розроблений DES компанією IBM. Довжина блоку даного алґоритму – 64 біт. Для шифрування та дешифрування DES використовує 16 раундів, при цьому довжина ключа дорівнює 56 бітам [26].

Потрійний DES (3DES) було розроблено на заміну DES, при цьому структура алгоритму залишилась незмінною. Фактично для шифрування даних TDES використовує сам DES для шифрування даних трьома різними ключами. Довжина блоку при шифруванні TDES залишається такою самою, а довжина ключа (168 біт) та кількість раундів (48) збільшуються, відповідно, утричі. На даний момент 3DES може забезпечити задовільний рівень безпеки, однак рекомендується використовувати більш надійні альтернативи (напр. AES) [27].

Rijndael або Advanced Encryption Standard (AES) – ітераційний симетричний блоковий алгоритм шифрування в основі якого лежить SP-мережа. Довжина блоку при шифруванні AES – 128 біт. Алгоритм підтримує роботи із такими довжинами

ключа, як 128, 192, 256 біт, так кількість раундів шифрування-дешифрування даного алгоритму є 10, 12 та 14 відповідно [28].

Для шифрування даний алгоритм використовує такі криптографічні примітиви, як нелінійна підстановка, циклічний зсув біт, поліноміальне множення, забілювання.

2.2 Порівняння криптостійкості

В даному підрозділі розділі будуть порівняні за стійкістю до класичних та квантових атак сучасні стандарти симетричного блокового шифрування. Для дослідження буде в тому числі використаний метод аналізу відкритих джерел, та існуючих досліджень [29, 30].

В першу чергу алгоритми будуть порівняні за класичною криптостійкістю, після чого, лише найнадійніші з них будуть порівняні за стійкістю до атак квантовим криптоаналізом.

2.2.1 Класична криптостійкість

Класична стійкість криптоалгоритму може бути виміряна по-перше максимальною кількістю необхідних ітерацій перебору ключа, щоб розшифрувати повідомлення (стійкістю до Brute Force атак), по-друге наявністю відомих вразливостей, що можуть значно зменшити кількість необхідних ітерацій перебору ключів. Обидві величини залежать від властивостей самого алгоритму, його архітектурних рішень та використаних в ньому криптографічних примітивів, довжини ключа, блоку, процедури розширення ключа (якщо вона присутня в алгоритмі).

Таким чином вибір максимально криптостійкого алгоритму буде проведений шляхом порівняння абсолютної криптостійкості, що вирахувана шляхом аналізу перетворень, що він виконує з даними, і наступним врахуванням наявності відомих вразливостей.

Blowfish. Безпечність даного алгоритму забезпечується варіативністю довжини ключа, що використовується для шифрування, а саме від 128 до 448 бітів. Blowfish був підданий активному криптоаналізу як тільки був опублікований, хоча, відносно інших розглянутих шифрів, кількість таких спроб була меншою. Завдяки змінній довжині ключів Blowfish стійкий до атак диференціальним криптоаналізом, оскільки усі біти головного ключа включають в себе велику кількість бітів раундових ключів, зв'язок між котрими дуже слабкий., тому подібні атаки на даний алгоритм вкрай неефективні або й зовсім нездійсненні[29]. Однак, хоч Blowfish не був зламаний, через малу довжину блоку у 64 біт він є вразливим до Birthday attacks, тому його використання на даний момент – погана практика.

Twofish має подвоєну довжину ключа у 128 біт відносно Blowfish, та одночасно з цим його можливість використання змінної довжини ключа зберігається, що робить його досить безпечним алгоритмом шифрування. Також подвоєна довжина блоку значно поліпшує його криптостійкість до атак грубої сили, оскільки за таких умов для її успішного виконання на шифротексті довжиною 128 біт необхідна значно більша обчислювальна потужність [30]. В той самий час, якщо попередньо обчислити SBoxes, що пов'язані із ключем, алгоритм стає вразливим до атак побічними каналами. Для того, щоб мінімізувати можливість подібної атаки, необхідно обчислювати SBoxes пов'язуючи їх не з основним ключем, а з раундовими. Таким чином фактично Twofish не є зламаним алгоритмом шифрування, попри деяку кількість атак на даний алгоритм. Відмінною рисою Twofish від інших алгоритмів є можливість перетворювати вхідний текст в великий за обсягом шифротекст, збільшуючи розмір даних майже у 4 рази. Для порівняння алгоритм AES при шифруванні може утворити шифротекст що більший за вхідні дані лише у 3,3 рази. Однак, суто с практичної точки зору, більший розмір шифротексту буде означати більш низьку швидкість передачі даних мережею, тому така особливість алгоритму може мати і негативні аспекти.

RC6 може забезпечити високу стійкість шифротексту до атак завдяки своїй властивості використання абсолютно випадкової серії вихідних бітів для 15 раундів (допускаються і менші значення), що використовуються на блоках довжиною 128

біт. Стійкість даного алгоритму до лінійного криптоаналізу забезпечується неможливістю її проведення, оскільки щоб використати її протягом 16 раундів атакуючому необхідно мати не менше ніж 2 у степені 119 відкритих повідомлень, тому така атака на даний момент вважається неможливою. RC6 при роботі із 12 або більшою кількістю раундів є також стійкий до атак диференційним криптоаналізом [30]. Однак існують дослідження, що RC6 може бути вразливим до атак статистичного аналізу. На даний момент ці атаки не є практичними, оскільки потребують більше ніж 2000 відкритих повідомлень, однак, сам факт наявності вразливості до статистичного аналізу демонструє, що при збільшенні раундів криптостійкість алгоритму не змінюється принципово (злам алгоритму був доведений при роботі із 14 раундами), оскільки вразливість полягає у роботі самого шифру.

Serpent використовує класичні засоби забезпечення стійкості шифротексту до атак, закріплюючи його великою кількістю раундів (шифр використовує 32 раунди) та запобіжною зміною ключа шифрування до досягнення обробки 264 блоків відкритого тексту (захист від атаки зіткнення). Однак, подібний підхід визначає невисоку здатність шифру адаптуватися до нових потенційних загроз, оскільки використовує добре вивчені методи, що підвищує ризик їх успішного криптоаналізу у майбутньому. Якщо ж прибрати додаткові 16 раундів, фактично криптостійкість алгоритму буде порівняною із 3DES, що знову ж таки говорить про те, що даний алгоритм не винаходить чогось принципово нового [30].

MARS забезпечує безпеку даних своєю особливою гетерогенною структурою із 32 різних раундів шифрування, що відрізняє його від розглянутих раніше шифрів. Так, шифр вважається захищеним від атак диференціальним криптоаналізом, візуальним криптоаналізом, атак із відносними ключами а також атак за часом. При ключах, що містять велику кількість одиниць або нулів, що йдуть неперервним рядком, MARS фактично може не шифрувати частину повідомлення. Також була опублікована стаття, в якій завдяки атаці типу MITM (Man In The Middle) в даному алгоритмі були зламані 21 з 32 раундів.

DES забезпечує високу стійкість шифрування завдяки своєму основному ключу, з якого можу бути згенеровано понад 7 тисяч ймовірних ключів. Дана особливість алгоритму забезпечує високу складність відтворення оригінального ключа. При достатньо високій частоті зміни ключа ризик зламу шифру ще більш значно знижується. Однак на даний момент суто за своєї малої довжини ключа алгоритм вважається небезпечним та вразливим до атак грубої сили. Так у 1998 році силами Electronic Frontier Foundation (EFF) був розроблений комп'ютер для зламу даного шифру, і завдяки йому вдалось «дешифрувати DES» отримавши ключ менше ніж за день. Таким чином використання даного алгоритму не є доцільним при передачі важливої конфіденційної інформації [29].

3DES є покращеною версією DES, в якій збільшується довжина ключа та фактично текст тричі шифрується самим алгоритмом DES, що значно зменшує вірогідність атак типу MITM. Однак, у 2016 році в алгоритмах DES та 3DES була знайдена критична вразливість CVE-2016-2183, таким чином NIST опублікував рішення, за якого 3DES має бути виключений із вживання в усіх додатках до 2023 року [29].

Rijndael реалізує шифрування, надійність якого напряму залежить від довжини ключа, що використовується при шифруванні. Так довжина ключа 256 біт вважається на даний момент часу надлишковою, одна була впроваджена з огляду на потенційні ризики виникнення майбутніх атак а також розвитку квантового криптоаналізу [29, 30].

Проти алгоритму Rijndael було знайдено немало атак, серед яких:

- Square Attack
- Improved Square Attack
- Impossible Differential Attack
- Reversed Key Schedule Attack

Однак усі ці атаки були визнані неможливими на практиці. Тобто фактично даний алгоритм не був зламаним.

Шифр ДСТУ 7642, що побудований на основі алґоритму AES на даний момент не має відомих вразливостей, оскільки при його розробці була проведена робота над усуненням слабких місць AES, а в самому алгоритмі, можливо через його відносну новизну та не високу поширеність, вразливостей не було знайдено. В той самий час ДСТУ 7642 може надати вищу абсолютну криптостійкість за рахунок можливості використання більшої довжини ключа та блоку (512 біт).

Серед покращень, що були впроваджені в алгоритмі Калина з огляду на виявлені в AES вразливості, слід виділити наступні [31]:

- була збільшена кількість раундів шифрування;
- при забілюванні використовується додавання за модулем 2 а також використання додавання за модулем 2^{64}, що забезпечує кращу стійкість до атак лінійного і диференціального криптоаналізу, інтерполяційної атаки, алгебраїчних атак;
- використання 4 SBoxes (в AES використовується 1), що поліпшує розсіювання алгоритму, а також надає додатковий захист від алгебраїчних атак;
- повністю перероблений механізм розширення початкового ключа, що поліпшує захист від усіх атак на механізми перетворень ключів, що відомі на даний момент;

За результатами даного розділу можна зробити проміжні висновки:

- алгоритми DES, 3DES і Blowfish є фактично застарілими та не можуть бути використані в будь-яких додатках.
- Алгоритм SERPENT забезпечує достатній рівень захищеності інформації, однак експлуатує добре вивчені класичні криптографічні примітиви, що обмежує його потенціал що може бути необхідним у разі, якщо буде знайдена серйозна вразливість в його роботі.
- Алгоритми Twofish, MARS і RC6 фактично не були зламані, однак існують дослідження, що не тільки знаходять певні вразливості в даних алгоритмах, а і експлуатують їх до часткового але значущого успіху, що говорить про те, що дані алгоритми цілком можуть використовуватись на даний момент для забезпечення

достатнього рівня захисту конфіденційності інформації, однак ризик, що вони будуть зламані в ближчому майбутньому вищий, аніж у AES та DSTU7624.

- AES фактично не був зламаний, а також «витримав» велику кількість атак, здійснених на нього з огляду на його поширеність, однак слід все ж таки приймати за дійсне те, що самі вразливості в ньому існують.
- DSTU7624 через свою незначну поширеність а також відносну новизну не був значно досліджений. Однак, він заснований на алгоритмі AES, при цьому закриває усі відомі його вразливості, а також може надати більшу криптостійкість до атак грубої сили за рахунок збільшеної довжини ключа, блоку та кількості раундів.

Таким чином, шифри AES та DSTU7624 з точки зору забезпечення максимально можливого захисту конфіденційності даних можна розглядати як самі актуальні на даний момент.

2.2.2 Квантова криптостійкість

Важливість визначення криптостійкості алгоритмів до квантових атак підтверджується активними дослідженнями спрямованими на розробку алгоритмів, що мають дану властивість а також вдосконалення вже існуючих [32].

Квантовою атакою відновлення ключа є процедура, що дозволяє відновити ключ шифрування за меншу кількість ітерацій ніж використовуючи атаку грубої сили, тому що в даній процедурі не є необхідним випробовувати усі можливі значення ключа.

Існує дослідження в якому були проведені квантові атаки із експлуатацією усіх відомих вразливостей AES і отримані наступні результати [33]:

- Квантова Square attack була успішно проведена для AES-192 і AES-256 при 7 раундах;
- Квантова атака DS-MITM була успішно проведена для AES-256 на 8 раундах.

Щодо атак на відновлення ключа, можна дійти висновку, що у постквантовому світі у цьому аспекті AES має більшу криптостійкість, оскільки найвідоміші класичні спеціальні атаки спроможні зламати до 9 раундів AES-256, в той час як в дослідженні отримані результати обмежуються 8-ма раундами.

Тим не менш, якщо окрім атак на відновлення ключів також брати до уваги інші можливі атаки, слід зазначити, що слабким місцем AES може стати його довжина блоку у 128 біт, оскільки не дивлячись на довжину ключа також у 128 біт, існують результати криптоаналізу поширених режимів роботи AES, таких як CBC та CTR, за яких AES не забезпечує 128-бітний захист повідомлення. А оскільки дані атаки можна квантово покращити, існує необхідність створення більш захищених режимів роботи для AES, на заміну двом переліченим вище.

В той самий час, порівнюючи AES із DSTU7624, що заснований на першому, можна точно сказати, що перелічені атаки на відновлення ключа останнього мають бути значно менш ефективними, через повну переробку механізму розширення ключа в DSTU7624 з урахуванням відомих вразливостей в AES. Щодо слабкого місця AES в фіксованій довжині блоку у 128 біт, шифр DSTU7624 дає можливість збільшити його довжину вдвічі та вчетверо, що має суттєво покращити його криптостійкість до подібних атак із роботою в певних режимах.

Таким чином можна зробити висновок, що на даний момент DSTU7624 можна вважати більш стійким до квантового криптоаналізу.

2.3 Порівняння швидкодії

Швидкодія криптоалгоритму випливає з його архітектури. Якщо перетворення, що виконує алгоритм потребують занадто багато обчислювальних ресурсів або часу, для забезпечення достатньою криптостійкості, або ж алгоритм потребує занадто ускладненого обміну ключовими даними, такий спосіб може бути визнаний недоцільним, навіть попри можливу високу криптостійкість.

Недоцільність випливає напряму з порушенням такої властивості інформації, як доступність, при занадто довгому очікуванні користувача необхідної інформації.

У випадку програмної реалізації алгоритму шифрування даний критерій може мати ще більшу значущість, оскільки апаратні реалізації, як правило, є більш продуктивними, тому з огляду на мету дипломної роботи, швидкодія алгоритму є значним критерієм вибору алгоритму.

Тестове середовище (хост), що буде використане для заміру швидкодії блочних алгоритмів має наступні характеристики:

- Операційна система (ОС) – Ubuntu 22.04 jammy;
- Ядро ОС – x86_64 Linux 5.15.0-33-generic;
- Центральний процесор – Intel Core (Haswell, no TSX) @ 2.594GHz;
- ОЗУ – 1929МіВ.

Для тестування швидкодії алгоритмів використана бібліотека криптографічних перетворень із відкритим вихідним кодом crypto++, котра написана мовою програмування С та містить реалізації найпоширеніших алгоритмів шифрування, написаних мовою Assembler, тим самим оптимізованих для найпопулярніших платформ. Результати заміру швидкодії алгоритмів симетричного блокового шифрування в різних режимах роботи наведені в таблиці 2.1.

Таблиця 2.1

Порівняння швидкодії симетричних блокових шифрів

Алгоритм	**Мова**	**Швидкодія (МіВ/секунду)**	**Мікросекунди для ініціалізації Ключа та IV**
AES/CTR (128-bit key)	AESNI	2472	0.336
AES/ECB (128-bit key)	AESNI	2359	0.115
AES/CTR (192-bit key)	AESNI	1591	0.379
AES/CTR (256-bit key)	AESNI	1492	0.364
AES/GCM	AESNI	1091	0.647
AES/XTS (256-bit key)	AESNI	702	0.596
AES/XTS (512-bit key)	AESNI	612	0.458
AES/XTS (384-bit key)	AESNI	599	0.432
AES/CFB (128-bit key)	AESNI	524	0.361
AES/CBC (128-bit key)	AESNI	518	0.285

продовження таблиці 2.1

AES/OFB (128-bit key)	AESNI	510	0.252
AES/CBC (192-bit key)	AESNI	442	0.283
AES/CCM (128-bit key)	AESNI	407	0.523
AES/EAX (128-bit key)	AESNI	403	0.810
AES/CBC (256-bit key)	AESNI	387	0.224
MARC4 (128-bit key)	C++	285	2.120
RC6/CTR (128-bit key)	C++	136	2.268
Twofish/CTR (128-bit key)	C++	113	4.261
MARS/CTR (128-bit key)	C++	111	1.064
Kalyna-128(128)/CTR (128-bit key)	C++	95	0.483
Blowfish/CTR (128-bit key)	C++	88	44.380
Kalyna-256(256)/CTR (256-bit key)	C++	83	0.831
Kalyna-128(256)/CTR (256-bit key)	C++	72	0.570
Kalyna-256(512)/CTR (512-bit key)	C++	70	0.947
Kalyna-512(512)/CTR (512-bit key)	C++	68	1.525
DES/CTR (64-bit key)	C++	57	3.895
Serpent/CTR (128-bit key)	C++	56	0.552
DES-XEX3/CTR (192-bit key)	C++	53	3.062
DES-EDE3/CTR (192-bit key)	C++	20	10.135

З результатів, наведених у таблиці 2.1, видно, що AES має найбільшу швидкодію. За ним йдуть Mars, RC6, Twofish, Калина, DES, SERPENT та 3DES.

Однак, можна також помітити, що усі шифри реалізовані в даному вимірі за допомогою мови програмування C++, окрім AES, що в даній бібліотеці має оптимізовану версію, що використовує апаратну оптимізацію – AESNI (розширення системи команд AES). Калина, побудована на базі AES, значно уступає йому в даному тестуванні у швидкості через його апаратну оптимізацію [34].

Для уточнення порівняння проведемо ще одне дослідження на базі того ж хоста із використанням бібліотеки cryptonite – сертифікованої бібліотеки криптографічних перетворень від Приват Банку. Результати вимірів наведені в таблиці 2.2 і таблиці 2.3.

Таблиця 2.2

Порівняння швидкодії AES та DSTU7624

(дані: 2Кб, проходів: 5000, потоків: 1)

Шифр	**Швидкодія (MB\sec)**
AES-128-ECB	41.7
AES-192-ECB	38.3
AES-256-ECB	39.6
DSTU7624-128-128-ECB	74.6
DSTU7624-256-256-ECB	41.2
DSTU7624-512-512-ECB	29.8

Таблиця 2.3

Порівняння швидкодії AES та DSTU7624

(дані: 2Кб, проходів: 5000, потоків: 8)

Шифр	**Швидкодія (MB\sec)**
AES-128-ECB	39.9
AES-192-ECB	33.8
AES-256-ECB	29.0
DSTU7624-128-128-ECB	57.2
DSTU7624-256-256-ECB	39.5
DSTU7624-512-512-ECB	26.03

З даних замірів можна зробити висновок, що за певних умов та відсутності апаратної підтримки алгоритму AES шифр Калина не тільки не відстає від першого у швидкодії, а й може перевищувати його продуктивність. З огляду на те, що фактично в першому досліді шифр AES був найпродуктивнішим, можна зробити висновок, що Калина також має дуже високу швидкодію, що підтверджується і високими показниками її продуктивності в апаратній реалізації [35].

Висновки за розділом 2

З точки зору класичної криптостійкості найбільш стійкими виявились AES та DSTU7624. AES попри велику кількість спроб зламати алгоритм, та знайдені вразливості, залишається безпечним алгоритмом шифрування через незначний успіх

активних досліджень на тему його криптоаналізу. З іншого боку DSTU7624 попри новизну і малу кількість досліджень з метою виявити вразливості алгоритму, все ж таки базується на AES та водночас закриває усі відомі вразливості, щодо стандарту, від якого він походить. До того ж даний алгоритм забезпечує більшу теоретичну криптостійкість завдяки можливості використовувати більші довжини ключів, блоків та кількість раундів.

AES є алгоритмом стійким до квантового криптоаналізу, однак для квантових реалізацій існуючих класичних атак на певні режими його роботи існує ризик зламу шифру через невелику довжину блоку. Цю проблему AES вирішує DSTU7624 завдяки підтримці блоків довжиною до 512 біт (в AES довжина блоку 128).

З точки зору швидкодії з усіх розглянутих алгоритмів найшвидшими виявилися AES та DSTU7624 (Калина). Програмна реалізація DSTU7624 (Калина) значно поступається реалізації AES із використанням AESNI (розширення системи команд AES). Однак, якщо помістити алгоритми в однакові умови, можна побачити, що Калина не поступається значно у швидкості AES, а в деяких випадках навіть демонструє вищу продуктивність.

Таким чином, майже за усіма критеріями на сьогоднішній день алгоритм DSTU7624 (Калина) є найбільш оптимальним симетричним шифром серед розглянутих для шифрування трафіку з метою забезпечення максимально можливого захисту конфіденційності інформації.

РОЗДІЛ 3
ОПИС ПРОГРАМНОГО МОДУЛЮ ЗАХИСТУ КОНФІДЕНЦІЙНОСТІ

3.1 Архітектура додатку

Додаток побудовано за клієнт серверною архітектурою. Така архітектура дозволяє централізувати трафік користувачів перед тим, як він буде відправлений через зовнішню мережу. Так, сервер може фактично використовуватись в якості шлюзу для з'єднання із віддаленими фізично приватними мережами, з'єднаними захищеним VPN-підключенням. Також подібне рішення дозволяє використовувати додаткові засоби безпеки на такому вузлі, наприклад: DLP-системи, антивірусні рішення, мережеві екрани.

Для IP-адресації в створеній VPN-мережі використовуються внутрішні адреси TUN-інтерфейсів, що створюються при з'єднанні . Внутрішні IP-адреси видаються сервером кожному із клієнтів, при успішному з'єднанні користувача із мережею після проходження процесу автентифікації та авторизації. Для створення TUN-інтерфейсу необхідно мати права адміністратора.

Для передачі даних через мережу інтернет використовується протокол TCP та довільний, обраний користувачем, порт TCP сокету. Увесь трафік, що відправляється за допомогою TCP (тобто до віддаленого користувача) перед відправкою шифрується (усе, окрім заголовків TCP-пакету).

Сервер підтримує кілька одночасних клієнтських з'єднань та дозволяє клієнтам обмінюватись будь-якими даними. Кожне з'єднання проходить через сервер, тому важливо забезпечити його максимальну пропускну здатність, для покращення доступності реалізованого сервісу.

Користувачі можуть під'єднуватись до сервера як із внутрішньої мережі, так і із зовнішньої, таким чином дозволяючи реалізувати структуру Remote Accesses VPN, за якої користувачі можуть безпечно отримувати доступ до внутрішніх ресурсів компанії, використовуючи при цьому підключення через глобальну

мережу. Безпечність зовнішнього під’єднання користувачів гарантується надійним методом автентифікації заснованим на використанні публічних та приватних ключів. Публічні ключі користувачів, що мають повноваження підключатись до серверу, зберігаються на ньому у відритому вигляді, а секретний ключ кожного з клієнтів має безпечно зберігатись на пристрої користувача, тому при створенні шифрується паролем. Для автентифікації серверу використовується аналогічний принцип: сервер має свою пару приватного та публічного ключа. Кожний клієнт має копію відкритого ключа сервера, а секретний ключ має зберігатися лише на сервері, та бути зашифрованим, розташованим в захищеному середовищі.

Далі будуть описані механізми забезпечення безпечності з’єднання та захисту конфіденційності даних, а також яким чином вони реалізовані.

Автентифікація та авторизація. Підтвердження ідентичності користувача відбувається за допомогою використання публічних та приватних ключів. Як тільки клієнт намагається встановити з’єднання із сервером, він має відправити серверу свій публічний ключ. Якщо даний ключ зберігається на сервері як авторизований, то запускається процедура перевірки автентичності користувача. Для неї сервер формує випадкову послідовність байтів довжиною 512 біт та підписує її, використовуючи власний секретний ключ. Сервер відправляє користувачеві підписану версію випадково згенерованої послідовності біт, тому при її отриманні користувач має можливість перевірити автентичність сервера, оскільки зберігає копію відкритого ключа. Якщо за допомогою публічного ключа вдається розшифрувати підпис сервера, він може вважатися автентифікованим, інакше з’єднання переривається на стороні клієнта. При успішній автентифікації сервера користувач підписує відправлені сервером дані власним приватним ключем, та відправляє їх серверу. Оскільки сервер вже має відкритий ключ користувача, він верифікує за його публічним ключем відправлений ним підпис, і якщо операція проходить вдало, клієнт успішно завершує процедуру автентифікації, інакше – сервер розриває з’єднання. Після успішного проходження автентифікації користувачем, сервер запускає процедуру авторизації, що полягає у безпечному обміну ключовими даними для захищеної шифруванням сесії та видачі сервером

внутрішньої IP-адреси користувача, за якою він буде ідентифікуватися у приватній мережі.

Щодо атак типу DDoS, їх ризик повинен в першу чергу нівелюватися правильними налаштуваннями мережевих екранів. Також можливе використання додаткових послуг захисту від DDoS атак, такі, як наприклад, ті, що надає компанія CloudFlare.

Також для автентифікації використовуються й криптографічні засоби. Так, для шифрування трафіку можна обрати режим роботи симетричного блокового алгоритму ДСТУ7624 під назвою GCM або Galois/Counter Mode– вибіркове гамування із прискореним виробленням імітовставки. Завдяки методу формування імітовставки в даному режимі, забезпечується автентичність повідомлень, оскільки імітовставка залежить від: секретного ключа, шифротексту, вектору ініціалізації і відкритих даних аутентифікації користувача (наприклад, IP-адреса і порт підключення). Таким чином, дана імітовставка може забезпечувати автентичність передачі кожного повідомлення, що надходить мережею, за допомогою криптографічних засобів.

Шифрування. Увесь трафік, що проходить через TCP з'єднання, тобто до віддаленого кінця тунелю, шифрується із використанням симетричного блокового алгоритму ДСТУ7624. Для шифрування використовується відкрита сертифікована реалізація ДСТУ7624 від ПриватБанку, а саме, бібліотека kryptonite, написана мовою низького рівня C, що значно прискорює її швидкодію порівняно із реалізаціями на мовах більш високого рівня, такою як Python, за допомогою якої і реалізований даний модуль. В програмі реалізовані два режими роботи даного алгоритму: CBC (зчеплення шифроблоків), що забезпечує лише конфіденційність, та вибіркове гамування із прискореним виробленням імітовставки (GCM), що забезпечує конфіденційність, цілісність та автентичність повідомлень. Алгоритм DSTU7624 є гнучким та масштабованим, підтримує 10 режимів роботи, різну довжину ключів і блоків а також кількість раундів, дозволяючи знайти компроміс між захищеністю даних та швидкодією. Усі можливі комбінації довжин ключів і блоків, а також кількості раундів, наведені в таблиці 3.1.

Таблиця 3.1

Можливі комбінації параметрів роботи ДСТУ7624

Довжина блоку (біт)	**Довжина ключа (біт)**	**Кількість раундів**
128	128	10
	256	14
256	256	14
	512	18
512	512	18

Так шифрування може забезпечити дуже високий рівень стійкості до атак Brute Force при використанні DSTU7624 із довжиною ключа і блок 512 біт, однак це може суттєво вплинути на швидкодію передач даних через захищений канал. Також, використання режиму роботи GCM дозволяє забезпечити конфіденційність, цілісність та автентичність повідомлень, однак він може бути значно довшим за CBC, що не має за необхідність виробляти імітовставку, при цьому даний режим роботи може забезпечити лише конфіденційність передаваного повідомлення. Просте реалізація CBC дозволяє використовувати власні алгоритми забезпечення цілісності та автентичності повідомлень, виконуючи лише шифрування, що надає гнучкості програмному рішенню.

Безпечний обмін ключовими даними в програмному модулі реалізується використанням протоколу обміну ключами Діффі-Хелмана на еліптичних кривих (NIST 800-56Ar2). Даний алгоритм дозволяє підвищити захищеність ключових даних, що використовуються для сесії з’єднання, тим самим підвищити захищеність конфіденційності передаваних даних. Процедура обміну ключами відбувається лише після успішного проходження клієнтом автентифікації методом challenge response, що полягає у відправці сервером запиту до клієнта, на який він має дати «правильну відповідь», наприклад – підписати дані своїм секретним ключем для наступної верифікації його підпису. Для обміну ключами, сервер генерує випадковий ключ сесії та відправляє його клієнтові за протоколом безпечного обміну, що полягає у формуванні тимчасових відкритого та закритого ключа на сервері та клієнті, за допомогою яких формується спільний секрет, що після розширюється до ключа довжиною 512 біт. Ключ сесії не записується у постійну

пам’ять, а знаходиться лишу у оперативній пам’яті, а після розриву з’єднання видаляється, та ніколи не використовується повторно. Таким чином забезпечується безпека ключа сесії, до я кого зловмисник не може отримати доступ, не скомпрометувавши повністю клієнтський або серверний хост.

Оновлення ключових даних сесії із часом в програмному додатку може бути реалізоване достатньо просто. При проходженні процедури авторизації клієнтом, сервер записує інформацію про час першого обміну ключовими даними. Після, орієнтуючись на цей час, кожні 30 хвилин сервер надсилає клієнту запит на оновлення ключових даних сесії, використовуючи протокол безпечного обміну ключовими даними. Дана міра дозволить суттєво зменшити можливість накоплення зловмисником інформації, пов’язаної із поточним ключом сесії, такої як шифротекст із використаними векторами ініціалізації та виробленими імітовставками, що значно зменшує площу можливих атак на алгоритм шифрування.

Доступність VPN-мережі в даному програмному модулі забезпечується:

- простотою реалізації – сервер можна перезавантажити дуже швидко, а відстеження помилок при виконанні спрощене малою кодовою базою, яку легко підтримувати;
- продуктивністю алгоритмів шифрування – для шифрування був обраний алгоритм, що не поступається швидкодії програмній реалізації AES, сучасному еталонному стандарту симетричного блокового шифрування. Це зменшує затримки передачі інформації, пов’язані із шифрування трафіку;
- використання паралельних та конкурентних обчислювань для шифрування та автентифікації користувачів, що прискорює обслуговування сервером декількох з’єднань.

Доступність VPN-мережі також напряму залежить від правильних налаштувань фільтрації трафіку за допомогою мережевих екранів, та безпечності середовища, в якому буде розгорнутий даний програмний додаток. Рекомендації щодо налаштувань мережевих екранів та забезпечення безпеки середовища

розгортання VPN-серверу виходять за рамки тематики цієї роботи та мають бути досліджені окремо.

Захист повідомлень від атаки типу «відтворення повідомлення» може бути забезпечене впровадженням для кожного пакету додаткової інформації, щодо часу його створення. Ця інформація передається у відкритому вигляді як заголовок пакету, однак має бути включена в дані аутентифікації повідомлення, при формуванні гешу в режимі роботи GCM ДСТУ7624, щоб його не можна було підробити. Так, кожне повідомлення буде вважатися валідним тільки якщо:

- геш, сформований в тому числі на основі часу відправлення пакету, обчислюється вірно;
- з моменту відправлення повідомлення пройшло не більше ніж 30 секунд.

В іншому випадку повідомлення відкидається. Таким чином унеможливлюється збереження зловмисником перехопленого повідомлення для повторного його використання у майбутньому. Для забезпечення захисту від дублювання пакетів, можна встановити обмеження на унікальність пакетів за часом відправки (неможливо відправити 2 пакети буквально одночасно).

3.2 Огляд програмного модулю

Програмний модуль реалізований мовою Python із використанням сторонніх бібліотек, написаних різними мовами. Список основних використаних сторонніх бібліотек та їх призначення:

- pytun – стороння бібліотека, написана мовою C, дозволяє створювати tun-інтерфейси;
- scapy – стороння бібліотека, дозволяє працювати із мережевими пакетами;
- cryptography – стороння бібліотека, містить реалізації багатьох сучасних криптографічних алгоритмів та примітивів. Була використана для реалізації автентифікації та безпечного обміну ключовими даними (вихідний код наведений в додатку Ж);

- cryptonite – бібліотека криптографічних перетворень, написана мовою C, реалізована компанією ПриватБанк. Була використана для реалізації шифрування трафіку симетричним алгоритмом шифрування ДСТУ7624.

В якості тестового середовища було використано 3 VPS сервери, що працюють на базі операційної системи Linux Ubuntu. Їх основні технічні характеристики наведені в таблиці 3.1 та 3.2 відповідно. Характеристики перших двох хостів однакові і відносяться до таблиці 3.1.

Таблиця 3.2

Технічні характеристики хостів тестового середовища

Характеристика	**Значення**
Операційна система (ОС)	Ubuntu 22.04 jammy
Ядро ОС	x86_64 Linux 5.15.0-33-generic
Центральний процесор	Intel Core (Haswell, no TSX) @ 2.594GHz
ОЗУ	1929MiB

Таблиця 3.3

Технічні характеристики хоста тестового середовища

Характеристика	**Значення**
Операційна система (ОС)	Ubuntu 20.04 focal
Ядро ОС	x86_64 Linux 5.4.0-113-generic
Центральний процесор	AMD EPYC 7282 16-Core @ 4x 2.795GHz
ОЗУ	7957MiB

Програмний модуль фактично має дві реалізації: клієнтську (вихідний код реалізації розміщений в додатках В, Д) та серверну (вихідний код реалізації розміщений в додатках А, Б). Серверна реалізація має додатковий функціонал, що дозволяє обробляти декілька підключень: зберігати дані про активні сесії, проводити автентифікацію клієнтів, видавати внутрішні IP-адреси, тощо. Клієнтська реалізація

– спрощена та призначена суто для підключення до внутрішньої мережі шляхом встановлення захищеного з’єднання із сервером.

Налаштування клієнтського та серверного додатків відбувається за допомогою конфігураційних файлів у форматі json. Приклад клієнтської та серверної реалізації наведені на рис. 3.1 та 3.2 відповідно.

```json
{
  "address": "51.89.165.32",
  "port": 12345,
  "iface_name": "tun0",
  "iface_mtu": 10000,
  "iface_netmask": "255.255.255.0",
  "pub_key_path": "./cred/id_ec.pub",
  "sec_key_path": "./cred/id_ec.private",
  "server_pub_key_paths": "./cred/server.pub",
  "encryption": true,
  "encryption_mode": "cbc"
}
```

Рис. 3.1 – Файл конфігурації клієнтського додатку

```json
{
  "address": "51.89.165.32",
  "port": 12345,
  "iface_name": "tun1",
  "iface_mtu": 10000,
  "iface_addr": "10.0.0.1",
  "iface_netmask": "255.255.255.0",
  "pub_key_path": "./cred/my_id_ec.pub",
  "sec_key_path": "./cred/my_id_ec.private",
  "auth_hosts_path": "./cred/authorized_hosts",
  "encryption": true,
  "encryption_mode": "cbc"
}
```

Рис. 3.2 – Файл конфігурації серверного додатку

Пояснення щодо зображених конфігурацій на рис. 3.1 та 3.2:

- address – визначає адресу сервера;

- port – визначає порт VPN-серверу, на якому буде запущений TCP-сервер для віддаленого з’єднання;
- iface_name – визначає назву TUN-інтерфейсу, що буде створений для тунельного з’єднання;
- iface_addr (лише сервер) – визначає адресу TUN-інтерфейсу серверного додатку (для клієнта адреса автоматично призначається сервером);
- iface_netmask – визначає маску підмережі, що буде вризначена створеномуу TUN-інтерфейсу;
- pub_key_path – шлях до файлу із відкритим ключем, необхідний для процедури автентифікації;
- sec_key_path – шлях до фалу із секретним ключем, необхідний для процедури автентифікації;
- auth_hosts_path (лише сервер) – шлях до фалу, що містить авторизовані публічні ключі клієнтів, необхідний для процедури автентифікації;
- server_pub_key_path (лише клієнт) – шлях до файлу із публічним ключем серверу, необхідний для процедури автентифікації;
- encryption – визначає, чи буде шифруватись трафік;
- encryption_mode – визначає режим роботи симетричного алгоритму (CBC або GCM).

Після виконання налаштувань, необхідно запустити серверний та клієнтський додатки. Їх запуск виконується шляхом виконання python-файлу в терміналі. Після запуску серверного додатку, він на основі файлу конфігурації створить на сервері TUN-інтерфейс, запустить TCP-сервер для віддаленого з’єднання, та перейде в режим очікування та обслуговування вхідних запитів. Приклад вдалого запуску серверного додатку і створеного TUN-інтерфейсу зображений на рис. 3.3 і 3.4 відповідно.

```
root@vps-9662a29f:/home/ubuntu/vpn_server# /home/u
buntu/vpn_server/venv/bin/python /home/ubuntu/vpn_
server/main_with_simple_tcp_server.py
[MAIN] TUN interface addr: 10.0.0.1
[MAIN] TUN interface name: tun1
[TCP SERVER] binding to 51.89.165.32, port 12345
```

Рис. 3.3 – Запуск серверного додатку

```
ubuntu@vps-9662a29f:~/vpn_server$ ifconfig tun1
tun1: flags=4305<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST>  mtu 10000
        inet 10.0.0.1  netmask 255.255.255.0  destination 10.0.0.1
        inet6 fe80::8440:55fa:9245:3e6  prefixlen 64  scopeid 0x20<link>
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuel
en 500  (UNSPEC)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8  bytes 384 (384.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Рис. 3.4 – Серверний TUN-інтерфейс

З рис. 3.3 та 3.4 видно, що сервер був запущений та налаштований згідно визначених конфігурацій. Запуск клієнтського додатку відбувається аналогічно, окрім того, що створюється не TCP-сервер, а TCP-клієнт, що одразу намагається під’єднатися до серверу, тим самим запустивши процедуру автентифікації і наступної авторизації. Приклад успішного запуску клієнтського додатку та під’єднання до внутрішньої мережі зображений на рис. 3.5.

```
root@vmi888022:/home/ubuntu/Desktop/vpn_client# /home/
ubuntu/Desktop/vpn_client/venv/bin/python /home/ubuntu
/Desktop/vpn_client/main_with_tcp_client.py
[TCP CLIENT] connecting to 51.89.165.32, port 12345
[TCP CLIENT] connected!
[AUTH] successes
[TCP CLIETN] auth data b'149.102.156.65 45140'
[MAIN] TUN interface addr: 10.0.0.2
[MAIN] TUN interface name: tun0
```

Рис. 3.5 – Запуск клієнтського додатку. Успішна автентифікація

З рис. 3.5 можна побачити, що клієнт спочатку під’єднався до серверу за TCP-протоколом, після чого успішно пройшов автентифікацію та отримав необхідні для роботи дані, такі як внутрішню адресу (10.0.0.2) та ключ сесії. У разі неправильних даних для авторизації зі сторони клієнта або сервера, з’єднання не буде встановлено. Приклад такої роботи додатку зображений на рис. 3.6.

```
(venv) ubuntu@vps-cce5a01e:~/vpn_client$ /home/ubuntu/vpn_client
/venv/bin/python /home/ubuntu/vpn_client/main_with_tcp_client.py
[TCP CLIENT] connecting to 51.89.165.32, port 12345
[TCP CLIENT] connected!
[AUTH] wrong auth credentials
[AUTH] failed
(venv) ubuntu@vps-cce5a01e:~/vpn_client$
```

Рис. 3.6 – Запуск клієнтського додатку. Невдала автентифікація

Після встановлення вдалого з’єднання усі учасники мережі можуть повноцінно взаємодіяти один з іншим використовуючи внутрішні IP-адреси. Так, для перевірки з’єднання між клієнтом та сервером була виконана команда ping із використанням внутрішньої адреси сервера (рис. 3.7).

```
ubuntu@vmi888022:~/Desktop/vpn_client$ ping 10.0.0.1 -c 3
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=688 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=504 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=505 ms

--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2006ms
rtt min/avg/max/mdev = 503.826/565.536/687.594/86.309 ms
ubuntu@vmi888022:~/Desktop/vpn_client$
```

Рис. 3.7 – Перевірка з’єднання між клієнтським та серверним додатком

Для перевірки шифрування трафіку використано програму WireShark. На VPN сервері був запущений Socks5 проксі-сервер через який клієнт робив HTTP запит,

таким чином цей запит був переданий на віддалений хост, завдяки чому його можна було перехопити. Код програми, що робить запит, наведений на рис. 3.8.

```
import requests

resp = requests.get(
    'http://info.cern.ch',
    proxies={
        'http': 'socks5://aboba:amogus@10.0.0.1:3000',
        'https': 'socks5://aboba:amogus@10.0.0.1:3000'
    })


print(resp.status_code)
print(resp.text)
```

Рис. 3.8 – Програма для тестового HTTP запиту

З рис. 3.8 видно, що запит буде відбуватися через проксі сервер визначений саме внутрішньою IP-адресу VPN мережі.

Для перехоплення пакетів необхідно запустити Wireshark на прослуховування клієнтського мережевого інтерфейсу (в даному випадку eth0) та відфільтрувати усі запити за IP-адресою призначення, а саме ip-адресою VPN сервера, оскільки саме із ним буде встановлений захищений канал зв’язку.

Для того, щоб перевірити працездатність socks5 серверу, виконано запит із відключення шифрування трафіку. Як результат, у найбільшому пакеті, що перехватив WireShark ми можемо бачити тіло HTTP GET запиту у відкритому вигляді (рис. 3.9).

Для демонстрації шифрування трафіку, виконано той самий запит, підключившись до VPN мережі із налаштуванням encryption: true, та був перехоплений пакет (рис. 3.10).

З рис. 3.10 видно, що найдовше повідомлення по-перше має більший розмір, що свідчить про використання padding при шифруванні, по-друге, відсутнє тіло запиту у відкритому вигляді, тобто, він була повністю зашифрований.

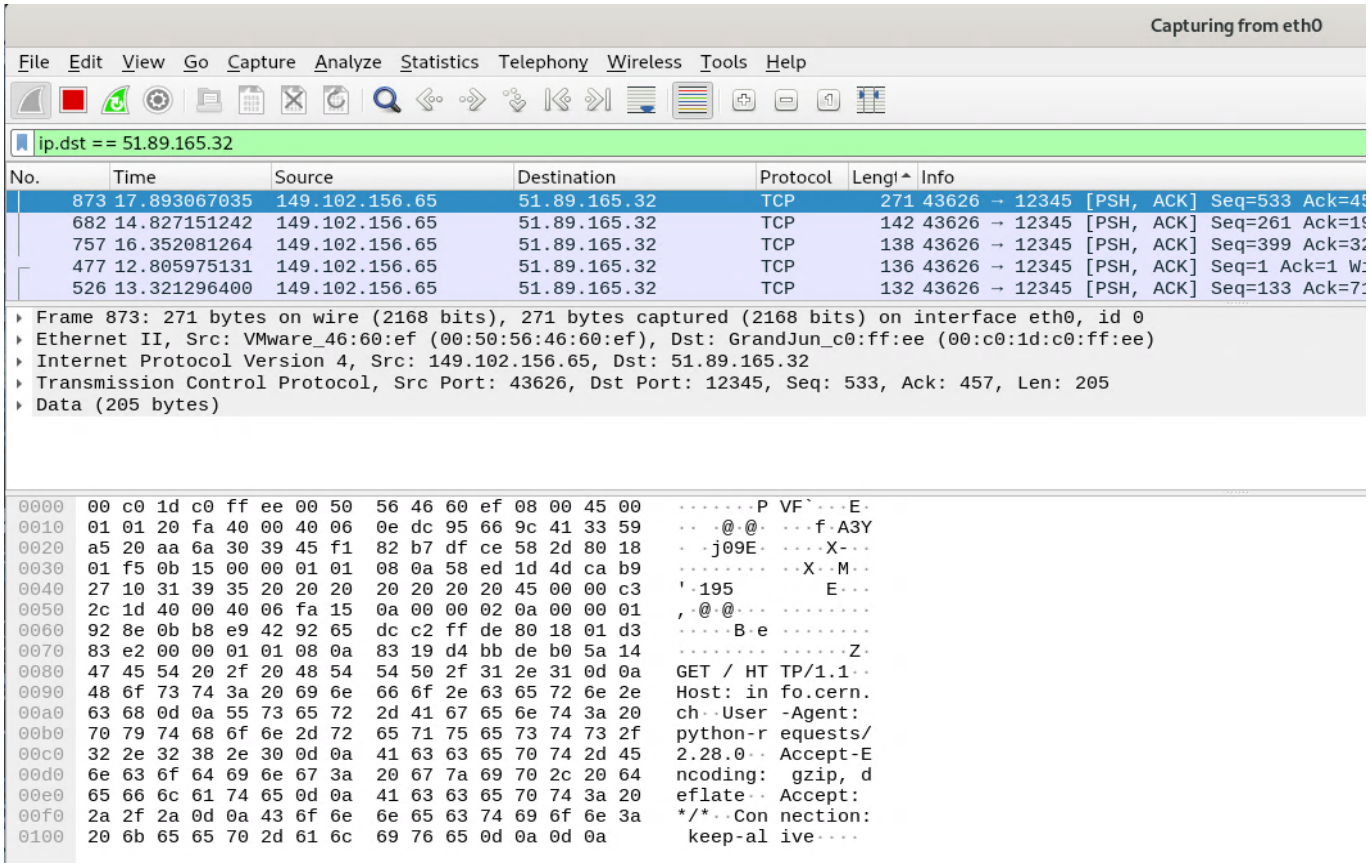

Рис. 3.9 – Перехоплений пакет із клієнтським HTTP запитом у відкритому вигляді

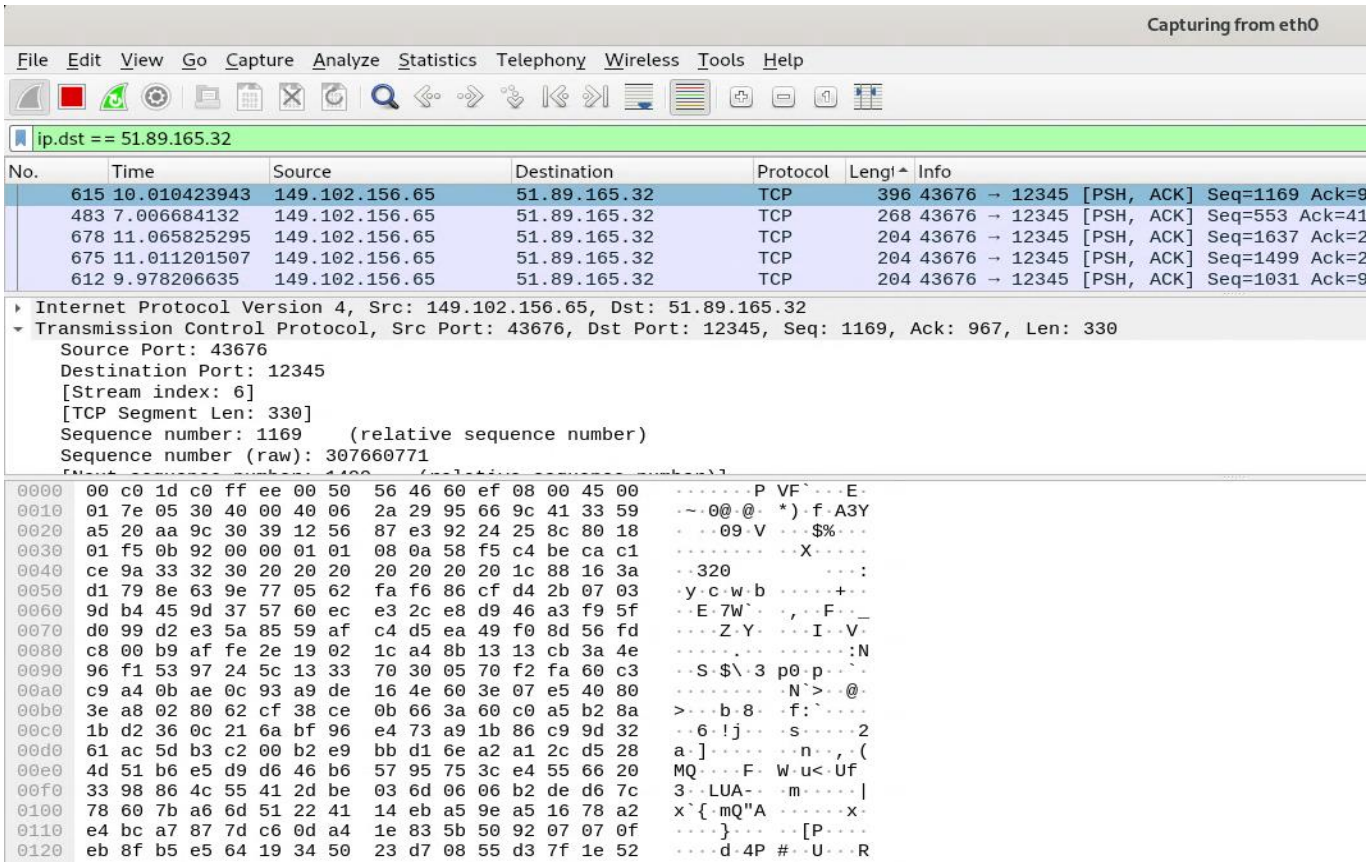

Рис. 3.10 – Перехоплений пакет із зашифрованим клієнтським HTTP запитом

Висновки за розділом 3

В даному розділі був описаний програмний модуль, що реалізує VPN мережу, а саме були розглянуті:

- архітектура додатку;
- впроваджені механізми захисту інформації;
- використані сторонні бібліотеки та їх призначення;
- тестове середовище, на базі якого працював додаток;
- процес налаштування клієнтського та серверного додатків;
- процес тестування розробленого додатку.

В додатку були реалізовані наступні механізмів захисту інформації:

- автентифікація та авторизація користувачів на основі відкритих та закритих ключів;
- автентифікація повідомлень за виробленою імітовставкою;
- шифрування повідомлень симетричним блоковим шифром;
- безпечний обмін ключовими даними.

Даний програмний додаток розроблений у відповідності до вимог актуальних протоколів реалізації VPN-мереж, що були розглянуті у першому розділі, а також використовує найбільш криптостійкий алгоритм симетричного блокового шифрування із достатньо високою швидкодією, що були проаналізовані в першому розділі. Так була виконана задача по розробці програмної реалізації VPN-мережі із підтримкою визначених в ході роботи критичних засобів захисту інформації.

ВИСНОВОК

Результатами виконання дипломної роботи стали виконання таких завдань, як:

- проаналізовані архітектури VPN-мережі;
- проаналізовані існуючі протоколи реалізації VPN-мереж;
- проведено порівняльний аналіз існуючих алгоритмів блокового шифрування;
- реалізовано VPN-мережу із підтримкою максимально криптостійкого та водночас ефективного алгоритму шифрування.

Згідно першого завдання були виведені архітектурні особливості реалізації VPN-мережі, та їх призначення.

Згідно другого завдання були виведені основні вимоги до протоколів реалізації VPN-мережі щодо захисту інформації, що були використані при реалізації програмного додатку.

Згідно третього завдання було виявлено, що алгоритм симетричного блокового шифрування ДСТУ7624 Калина може забезпечити найвищу криптостійкість повідомлень за прийнятної швидкодії, є гнучким та не має відомих вразливостей, стійкий до квантового криптоаналізу.

Практичним результатом дипломної роботи є програмний додаток, реалізований згідно четвертого завдання, що наочно демонструє роботу алгоритму Калина при шифруванні VPN трафіку, захищає конфіденційність інформації при її передачі глобальною мережею у разі забезпечення комплексного механізму забезпечення інформаційної безпеки захищеного каналу зв’язку.

Даний програмний додаток може бути використаний для подальшого аналізу криптоалгоритму і покращення захищеності конфіденційності даних при їх передачі у VPN мережі, а також як навчальний макет для дослідження повного процесу забезпечення захисту інформації у VPN-мережах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. PPTP vs L2TP vs OpenVPN vs SSTP. [Електронний ресурс]. Режим доступу: https://habr.com/ru/post/191874/.
2. Юнусов Р. – Скільки грошей країни світу витрачають на квантові технології. [Електронний ресурс]. Режим доступу: https://is.gd/2YjdwN.
3. Ковтун К. О. – Метод передачі повідомлень з використанням квантово-захищеного криптографічного алгоритму. [Електронний ресурс]. Режим доступу: https://is.gd/PRvk9o.
4. Економічна Правда – Коронавірус VS бізнес: половина підприємств протримається на карантині не більше місяця [Електронний ресурс]. Режим доступу: https://www.epravda.com.ua/rus/publications/2020/04/2/658857/.
5. Аліса В., Гейб Т. – 2022 VPN Usage Statistics [Електронний ресурс]. Режим доступу: https://www.security.org/vpn/statistics/.
6. What Is a Business VPN? Understand Its Uses and Limitations. [Електронний ресурс]. Режим доступу: https://is.gd/mVINCs.
7. Урнова А. – Держвитрати: Держава витратила майже 7 млрд руб. на створення та використання VPN-мереж за 3,5 роки. [Електронний ресурс]. Режим доступу: https://is.gd/6tX4zZ.
8. USING A VPN FOR GOVERNMENT AGENCIES. [Електронний ресурс]. Режим доступу: https://sdi.ai/blog/using-a-vpn-for-government-agencies/.
9. IBM – Virtual Private Networking [Електронний ресурс]. Режим доступу: https://www.ibm.com/docs/en/i/7.1?topic=security-virtual-private-networking.
10. Cisco Security – What Is a VPN? - Virtual Private Network [Електронний ресурс]. Режим доступу: https://www.cisco.com/c/en/us/products/security/vpn-endpoint-security-clients/what-is-vpn.html.
11. Cloud Flare Documentation – What is a business VPN? Business VPN uses and limitations [Електронний ресурс]. Режим доступу: https://www.cloudflare.com/learning/access-management/what-is-a-business-vpn/.
12. Frankel S., Krishnan S. – IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap (RFC6071) [Електронний ресурс]. Режим доступу: https://datatracker.ietf.org/doc/html/rfc6071.
13. Kaufman C., Hoffman P., Nir Y., Eronen P. – Internet Key Exchange Protocol Version 2 (IKEv2). [Електронний ресурс]. Режим доступу: https://datatracker.ietf.org/doc/html/rfc5996.
14. Townsley W., Valencia A., Rubens A. – Layer Two Tunneling Protocol "L2TP" [Електронний ресурс]. Режим доступу: https://datatracker.ietf.org/doc/html/rfc2661.
15. Shacham A., Monsour R., Pereira R. – IP Payload Compression Protocol (IPComp). [Електронний ресурс]. Режим доступу: https://www.rfc-editor.org/rfc/rfc2393.html.

16. Hamzeh K., Pall G., Verthein W. – Point-to-Point Tunneling Protocol (PPTP). [Електронний ресурс]. Режим доступу: https://datatracker.ietf.org/doc/html/rfc2637.

17. Microsoft Documentation – [MS-SSTP]: Secure Socket Tunneling Protocol (SSTP). [Електронний ресурс]. Режим доступу: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-sstp/c50ed240-56f3-4309-8e0c-1644898f0ea8.

18. OpenVPN Documentation – OpenVPN Protocol. [Електронний ресурс]. Режим доступу: https://openvpn.net/community-resources/openvpn-protocol/.

19. WireGuard Documentation – Protocol & Cryptography. [Електронний ресурс]. Режим доступу: https://www.wireguard.com/protocol/.

20. Schneier on Security– The Blowfish Encryption Algorithm. [Електронний ресурс]. Режим доступу: https://www.schneier.com/academic/blowfish/.

21. Schneier on Security – Twofish. [Електронний ресурс]. Режим доступу: https://www.schneier.com/academic/twofish/.

22. Contini S., Rivest R. L., Robshaw M. – The Security of the RC6 Block Cipher. [Електронний ресурс]. Режим доступу: https://is.gd/IGVwVv.

23. SERPENT Documentation– A Candidate Block Cipher for the Advanced Encryption Standard. [Електронний ресурс]. Режим доступу: https://www.cl.cam.ac.uk/~rja14/serpent.html.

24. Burwick C., Coppersmith D., Gennaro R. – The MARS Encryption Algorithm. [Електронний ресурс]. Режим доступу: https://is.gd/pec3bg.

25. Oliynykov R., Gorbenko I., Kazymyrov O.– A New Standard of Ukraine: The Kupyna Hash Function. [Електронний ресурс]. Режим доступу: https://eprint.iacr.org/2015/885.

26. Mehuron W.– DATA ENCRYPTION STANDARD (DES). [Електронний ресурс]. Режим доступу: https://is.gd/me9q9b.

27. Barker E., Mouha N.– Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher. [Електронний ресурс]. Режим доступу: https://csrc.nist.gov/publications/detail/sp/800-67/rev-2/final.

28. National Institute of Standards and Technology – Advanced Encryption Standard (AES). [Електронний ресурс]. Режим доступу: https://is.gd/tAcvgP.

29. NURGALIYEV A.N. – COMPARATIVE STUDY OF SYMMETRIC CRYPTOGRAPHIC ALGORITHMS. [Електронний ресурс]. Режим доступу: https://vestnik.kbtu.edu.kz/jour/article/view/277/287.

30. Берніков В.О. – Порівняльний аналіз криптостійкості симетричних алгоритмів шифрування. [Електронний ресурс]. Режим доступу: https://is.gd/Rlyl1C.

31. Байлюк Є.М., Покотило О.А. – АНАЛІЗ ТА ПОРІВНЯННЯ АЛГОРИТМУ СИМЕТРИЧНОГО БЛОКОВОГО ПЕРЕТВОРЕННЯ «КАЛИНА» (ДСТУ 7624:2014) З МІЖНАРОДНИМ СТАНДАРТОМ ШИФРУВАННЯ ДАНИХ AES. [Електронний ресурс]. Режим доступу: https://conf.ztu.edu.ua/wp-content/uploads/2018/05/3-1.pdf.

32. National Institute of Standards and Technology – Post-Quantum Cryptography. [Електронний ресурс]. Режим доступу: https://csrc.nist.gov/Projects/post-quantum-cryptography.

33. Bonnetain X., Naya-Plasencia M– Quantum Security Analysis of AES [Електронний ресурс]. Режим доступу: https://eprint.iacr.org/2019/272.pdf.

34. Калінін Д.А., Козіна Г.Л. – Швидкодія шифрів «Калина» та AES. [Електронний ресурс]. Режим доступу: https://is.gd/6TYJvl.

35. Система криптографічного захисту каналів. [Електронний ресурс]. Режим доступу: https://cipher.com.ua/ru/products/cipher-vpn.

ДОДАТОК А

Вихідний код VPN серверу

```
import select
import json
import errno

from typing import Literal

import pytun
import scapy.all as scapy
import asymmetric_staff as asym_crypt

from cryptonite import Cryptonite
from tcp_client import TCPClient, SessionCredentials

MESSAGE_FORMAT = 'utf-8'
IP_V4_PROTO = b'\x08\x00'


class Config:
    def __init__(self, data):
        self.address: str   = data['address']
        self.port: int      = data['port']

        self.iface_name: str    = data['iface_name']
        self.iface_netmask: str = data['iface_netmask']
        self.iface_mtu: int     = data['iface_mtu']

        self.pub_key_path: str          = data['pub_key_path']
        self.sec_key_path: str          = data['sec_key_path']
        self.server_pub_key_path: str   = data['server_pub_key_paths']

        self.ENCRYPTION: bool                   = data['encryption']
        self.ENC_MODE: Literal['cbc', 'gcm']    = data['encryption_mode']


def setup_tun_iface(config: Config, iface_addr: str):

    tun_iface = pytun.TunTapDevice(name=config.iface_name)

    print(f'[MAIN] TUN interface addr: {iface_addr}')
    print(f'[MAIN] TUN interface name: {config.iface_name}')

    tun_iface.addr    = iface_addr
```

```
    tun_iface.netmask = config.iface_netmask
    tun_iface.mtu     = config.iface_mtu

    tun_iface.up()
    return tun_iface


def handle_iface_data(tcp_client: TCPClient, tun_iface):
    readable, _, _ = select.select([tun_iface.fileno()], [], [], 0.5)
    if tun_iface.fileno() in readable:
        try:
            buf = tun_iface.read(tun_iface.mtu)
        except Exception as ex:
            print('[Error] reading tun data error')
            print(str(ex))

        # flags = buf[:2]
        proto = buf[2:4]

        if proto != IP_V4_PROTO:
            # print('[TUN IFACE] recive a non IP packet. Droped')
            return

        ip_packet = scapy.IP(buf[4:])

        handle_ip_packet(ip_packet, tcp_client, tun_iface)


def handle_ip_packet(ip_packet, tcp_client: TCPClient, tun_iface):
    print('-' * 100)
    print(f'{ip_packet.src} → {ip_packet.dst} len: {ip_packet.len}')

    ip_packet_bytes = scapy.raw(ip_packet)
    # print('[IP PACKET]', ip_packet_bytes)

    if len(ip_packet_bytes) != ip_packet.len:
        print(len(ip_packet_bytes))
        print(ip_packet.len)
        raise RuntimeError('[IP PACKETS HANDLER] invalid "total length" header value')

    if ip_packet.dst == tun_iface.addr:
        print('[IP PACKETS HANDLER] sending to tun iface')
        tun_packet = b'\x00\x00' + IP_V4_PROTO + ip_packet_bytes
        while len(tun_packet):
            nbytes = tun_iface.write(tun_packet)
            tun_packet = tun_packet[nbytes:]

    else:
        print('[IP PACKETS HANDLER] sending to remote peer')
        tcp_client.messages_to_send.append(ip_packet_bytes)
```

```
# PKI AUTH
def pki_auth(tcp_client: TCPClient) -> bool:
    tcp_client.send_bytes(asym_crypt.serialize_ec_pub_key(tcp_client.pub_key))

    test_data = tcp_client.recieve_message()
    server_signature = tcp_client.recieve_message()

    if test_data is None:
        print(f'[AUTH] wrong auth credentials')
        return False

    if not asym_crypt.verify_message(test_data, server_signature,
tcp_client.server_pub_key):
        print(f'[AUTH] server pub key verify false')
        return False

    signature = asym_crypt.sign_message(test_data, tcp_client.sec_key)
    tcp_client.send_bytes(signature)
    return True


# KEY EXCHANGE PROCEDURE
def key_exchange(tcp_client: TCPClient, server_pub_key: bytes) -> bytes:
    private_key, public_key = asym_crypt.gen_ec_keys()
    tcp_client.send_bytes(asym_crypt.serialize_ec_pub_key(public_key))
    peer_public_key = asym_crypt.load_ec_pub_key(server_pub_key)
    return asym_crypt.derive_key(private_key, peer_public_key)


def authorize(tcp_client: TCPClient) -> str:

    if not pki_auth(tcp_client):
        print(f'[AUTH] failed')
        tcp_client.connection.close()
        exit()
    else:
        print('[AUTH] successes')

    session_key = key_exchange(tcp_client, tcp_client.recieve_message())

    internal_ip = tcp_client.recieve_message().decode(MESSAGE_FORMAT)
    ip, port = tcp_client.connection.getsockname()
    auth_data = bytes(f"{ip} {port}", MESSAGE_FORMAT)
    print("[TCP CLIETN] auth data", auth_data)
    # print(f'[SESSION KEY]', session_key)

    tcp_client.session_cred = SessionCredentials(auth_data, session_key)

    return internal_ip

def main():
```

```
    with open('config.json') as config_file:
        config = Config(json.load(config_file))

    tcp_client = TCPClient(
        config.address,
        config.port,
        asym_crypt.load_pub_key_from_file(config.pub_key_path),
        asym_crypt.load_sec_key_from_file(config.sec_key_path, b'aboba'),
        asym_crypt.load_pub_key_from_file(config.server_pub_key_path))
    tcp_client.connect()
    internal_ip = authorize(tcp_client)
    tun_iface = setup_tun_iface(config, internal_ip)
    cr = Cryptonite()

    while True:
        handle_iface_data(tcp_client, tun_iface)
        readable_sockets, whitable_sockets, excepction_sockets =
select.select([tcp_client.connection], [tcp_client.connection], [tcp_client.connection])
        if tcp_client.connection in readable_sockets:
            try:
                accepted_message = tcp_client.recieve_message()
            except ConnectionResetError:
                print(f'Reset connection. Disconnecting')
                tcp_client.connection.close()
                exit()
            except Exception as ex:
                print(f'Generic exception while recieving')
                tcp_client.connection.close()
                exit()
            else:
                if accepted_message is not None:
                    if config.ENCRYPTION:

                        if config.ENC_MODE == 'gcm':
                            decrypted_message =
cr.gcm_decrypt_64_auto(tcp_client.session_cred.key, accepted_message,
tcp_client.server_auth_data)
                        elif config.ENC_MODE == 'cbc':
                            decrypted_message =
cr.cbc_decrypt_64_auto(tcp_client.session_cred.key, accepted_message)

                        if decrypted_message is None:
                            print('[TCP CLIENT] message decryption false / verify
false')
                            continue

                        ip_packet = scapy.IP(decrypted_message)

                    else:
                        ip_packet = scapy.IP(accepted_message)
```

```
                    handle_ip_packet(ip_packet, tcp_client, tun_iface)


        if tcp_client.connection in whitable_sockets:
            if tcp_client.messages_to_send:
                while message := tcp_client.messages_to_send.pop():
                    if config.ENCRYPTION:
                        if config.ENC_MODE == 'gcm':
                            enc_message_with_iv_and_mac =
cr.gcm_encrypt_64_auto(tcp_client.session_cred.key, message,
tcp_client.session_cred.auth_data)
                        elif config.ENC_MODE == 'cbc':
                            enc_message_with_iv_and_mac =
cr.cbc_encrypt_64_auto(tcp_client.session_cred.key, message)

                        if enc_message_with_iv_and_mac is None:
                            print('[TCP CLIENT] message encryption false.')
                            continue
                        else:
                            message = enc_message_with_iv_and_mac

                    try:
                        tcp_client.send_bytes(message)
                    except IOError as e:
                        if e.errno == errno.EPIPE:
                            print(f'Server excedently broke pipe')
                            tcp_client.connection.sock.close()
                            exit()
                    except Exception as ex:
                        print(f'[TCP CLIENT] Server excedently disconnected')
                        print(str(ex))
                        tcp_client.connection.close()
                        exit()

                    if not tcp_client.messages_to_send:
                        break

        if tcp_client.connection in excepction_sockets:
            print('[TCP CLIENT] server exedently close the connection')
            tcp_client.connection.close()
            exit()


if __name__ == '__main__':
    main()
```

ДОДАТОК Б

Вихідний код TCP-серверу

```
from dataclasses import dataclass
import socket
import atexit
from cryptonite import Cryptonite
from typing import List, Optional
from cryptography.hazmat.primitives.asymmetric.ec import EllipticCurvePrivateKey,
EllipticCurvePublicKey


MESSAGE_FORMAT = 'utf-8'

@dataclass
class Connection:
    sock: socket.socket
    addr: str
    port: int
    internal_ip: str
    auth_data: bytes
    key: Optional[bytes]
    pub_key: Optional[bytes]

@dataclass
class MessageToSend:
    sock: socket.socket
    message: bytes


class TCPServer():
    def __init__(self, ip: str, port: int, pub_key: EllipticCurvePublicKey, sec_key:
EllipticCurvePrivateKey, header_lengh: int = 10):
        self.ip: str = ip
        self.port: int = port
        self.pub_key = pub_key
        self.sec_key = sec_key

        self.HEADERSIZE: int = header_lengh

        self.connections: List[Connection] = list()
        self.accepted_connactions: List[Connection] = list()

        self.message_to_send: List[MessageToSend] = list()
        self.recieved_messages: List[bytes] = list()

        self.cr: Cryptonite = Cryptonite()
```

```
        self.auth_data = bytes(f'{self.ip} {self.port}', MESSAGE_FORMAT)

    def clean_up(self):
        for conn in self.connections:
            conn.sock.close()

    def start(self):
        self.server_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server_sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        print(f'[TCP SERVER] binding to {self.ip}, port {self.port}')
        self.server_sock.bind((self.ip, self.port))
        self.server_sock.listen()
        self.connections.append(Connection(self.server_sock, self.ip, self.port,
'10.0.0.1', self.auth_data, None, None))
        atexit.register(self.clean_up)

    def send_bytes(self, connection: socket.socket, bytes_to_send: bytes):
        header = bytes(f'{len(bytes_to_send):<{self.HEADERSIZE}}',  MESSAGE_FORMAT)
        connection.send(header + bytes_to_send)

    def recieve_message(self, connection: socket.socket) -> Optional[bytes]:
        message_header = connection.recv(self.HEADERSIZE)
        if not len(message_header):
            return None

        return connection.recv(int(message_header))
```

ДОДАТОК В

Вихідний код VPN-клієнта

```
import select
import json
import errno

from typing import Literal

import pytun
import scapy.all as scapy
import asymmetric_staff as asym_crypt

from cryptonite import Cryptonite
from tcp_client import TCPClient, SessionCredentials

MESSAGE_FORMAT = 'utf-8'
IP_V4_PROTO = b'\x08\x00'


class Config:
    def __init__(self, data):
        self.address: str   = data['address']
        self.port: int      = data['port']
        self.iface_name: str    = data['iface_name']
        self.iface_netmask: str = data['iface_netmask']
        self.iface_mtu: int     = data['iface_mtu']
        self.pub_key_path: str          = data['pub_key_path']
        self.sec_key_path: str          = data['sec_key_path']
        self.server_pub_key_path: str   = data['server_pub_key_paths']
        self.ENCRYPTION: bool                   = data['encryption']
        self.ENC_MODE: Literal['cbc', 'gcm']    = data['encryption_mode']

def setup_tun_iface(config: Config, iface_addr: str):

    tun_iface = pytun.TunTapDevice(name=config.iface_name)

    print(f'[MAIN] TUN interface addr: {iface_addr}')
    print(f'[MAIN] TUN interface name: {config.iface_name}')

    tun_iface.addr    = iface_addr
    tun_iface.netmask = config.iface_netmask
    tun_iface.mtu     = config.iface_mtu

    tun_iface.up()
```

```
    return tun_iface


def handle_iface_data(tcp_client: TCPClient, tun_iface):
    readable, _, _ = select.select([tun_iface.fileno()], [], [], 0.5)
    if tun_iface.fileno() in readable:
        try:
            buf = tun_iface.read(tun_iface.mtu)
        except Exception as ex:
            print('[Error] reading tun data error')
            print(str(ex))

        # flags = buf[:2]
        proto = buf[2:4]

        if proto != IP_V4_PROTO:
            # print('[TUN IFACE] recive a non IP packet. Droped')
            return

        ip_packet = scapy.IP(buf[4:])

        handle_ip_packet(ip_packet, tcp_client, tun_iface)


def handle_ip_packet(ip_packet, tcp_client: TCPClient, tun_iface):
    print('-' * 100)
    print(f'{ip_packet.src} → {ip_packet.dst} len: {ip_packet.len}')

    ip_packet_bytes = scapy.raw(ip_packet)
    # print('[IP PACKET]', ip_packet_bytes)

    if len(ip_packet_bytes) != ip_packet.len:
        print(len(ip_packet_bytes))
        print(ip_packet.len)
        raise RuntimeError('[IP PACKETS HANDLER] invalid "total length" header value')

    if ip_packet.dst == tun_iface.addr:
        print('[IP PACKETS HANDLER] sending to tun iface')
        tun_packet = b'\x00\x00' + IP_V4_PROTO + ip_packet_bytes
        while len(tun_packet):
            nbytes = tun_iface.write(tun_packet)
            tun_packet = tun_packet[nbytes:]

    else:
        print('[IP PACKETS HANDLER] sending to remote peer')
        tcp_client.messages_to_send.append(ip_packet_bytes)


# PKI AUTH
def pki_auth(tcp_client: TCPClient) -> bool:
    tcp_client.send_bytes(asym_crypt.serialize_ec_pub_key(tcp_client.pub_key))
```

```
    test_data = tcp_client.recieve_message()
    server_signature = tcp_client.recieve_message()

    if test_data is None:
        print(f'[AUTH] wrong auth credentials')
        return False

    if not asym_crypt.verify_message(test_data, server_signature,
tcp_client.server_pub_key):
        print(f'[AUTH] server pub key verify false')
        return False

    signature = asym_crypt.sign_message(test_data, tcp_client.sec_key)
    tcp_client.send_bytes(signature)
    return True


# KEY EXCHANGE PROCEDURE
def key_exchange(tcp_client: TCPClient, server_pub_key: bytes) -> bytes:
    private_key, public_key = asym_crypt.gen_ec_keys()
    tcp_client.send_bytes(asym_crypt.serialize_ec_pub_key(public_key))
    peer_public_key = asym_crypt.load_ec_pub_key(server_pub_key)
    return asym_crypt.derive_key(private_key, peer_public_key)


def authorize(tcp_client: TCPClient) -> str:

    if not pki_auth(tcp_client):
        print(f'[AUTH] failed')
        tcp_client.connection.close()
        exit()
    else:
        print('[AUTH] successes')

    session_key = key_exchange(tcp_client, tcp_client.recieve_message())

    internal_ip = tcp_client.recieve_message().decode(MESSAGE_FORMAT)
    ip, port = tcp_client.connection.getsockname()
    auth_data = bytes(f"{ip} {port}", MESSAGE_FORMAT)
    print("[TCP CLIETN] auth data", auth_data)
    # print(f'[SESSION KEY]', session_key)

    tcp_client.session_cred = SessionCredentials(auth_data, session_key)

    return internal_ip

def main():
    with open('config.json') as config_file:
        config = Config(json.load(config_file))
```

```
    tcp_client = TCPClient(
        config.address,
        config.port,
        asym_crypt.load_pub_key_from_file(config.pub_key_path),
        asym_crypt.load_sec_key_from_file(config.sec_key_path, b'aboba'),
        asym_crypt.load_pub_key_from_file(config.server_pub_key_path))
    tcp_client.connect()

    internal_ip = authorize(tcp_client)

    tun_iface = setup_tun_iface(config, internal_ip)


    cr = Cryptonite()

    while True:
        handle_iface_data(tcp_client, tun_iface)

        readable_sockets, whitable_sockets, excepction_sockets =
select.select([tcp_client.connection], [tcp_client.connection], [tcp_client.connection])

        if tcp_client.connection in readable_sockets:
            try:
                accepted_message = tcp_client.recieve_message()
            except ConnectionResetError:
                print(f'Reset connection. Disconnecting')
                tcp_client.connection.close()
                exit()
            except Exception as ex:
                print(f'Generic exception while recieving')
                tcp_client.connection.close()
                exit()
            else:
                if accepted_message is not None:
                    if config.ENCRYPTION:

                        if config.ENC_MODE == 'gcm':
                            decrypted_message =
cr.gcm_decrypt_64_auto(tcp_client.session_cred.key, accepted_message,
tcp_client.server_auth_data)
                        elif config.ENC_MODE == 'cbc':
                            decrypted_message =
cr.cbc_decrypt_64_auto(tcp_client.session_cred.key, accepted_message)

                        if decrypted_message is None:
                            print('[TCP CLIENT] message decryption false / verify
false')
                            continue
                        ip_packet = scapy.IP(decrypted_message)

                    else:
```

```
                        ip_packet = scapy.IP(accepted_message)

                    handle_ip_packet(ip_packet, tcp_client, tun_iface)
        if tcp_client.connection in whitable_sockets:
            if tcp_client.messages_to_send:
                while message := tcp_client.messages_to_send.pop():
                    if config.ENCRYPTION:
                        if config.ENC_MODE == 'gcm':
                            enc_message_with_iv_and_mac =
cr.gcm_encrypt_64_auto(tcp_client.session_cred.key, message,
tcp_client.session_cred.auth_data)
                        elif config.ENC_MODE == 'cbc':
                            enc_message_with_iv_and_mac =
cr.cbc_encrypt_64_auto(tcp_client.session_cred.key, message)

                        if enc_message_with_iv_and_mac is None:
                            print('[TCP CLIENT] message encryption false.')
                            continue
                        else:
                            message = enc_message_with_iv_and_mac
                    try:
                        tcp_client.send_bytes(message)
                    except IOError as e:
                        if e.errno == errno.EPIPE:
                            print(f'Server excedently broke pipe')
                            tcp_client.connection.sock.close()
                            exit()
                    except Exception as ex:
                        print(f'[TCP CLIENT] Server excedently disconnected')
                        print(str(ex))
                        tcp_client.connection.close()
                        exit()
                    if not tcp_client.messages_to_send:
                        break
        if tcp_client.connection in excepction_sockets:
            print('[TCP CLIENT] server exedently close the connection')
            tcp_client.connection.close()
            exit()


if __name__ == '__main__':
    main()
```

ДОДАТОК Д

Вихідний код TCP-клієнта

```
import socket
from typing import Optional, List
from dataclasses import dataclass
from cryptography.hazmat.primitives.asymmetric.ec import EllipticCurvePrivateKey,
EllipticCurvePublicKey
MESSAGE_FORMAT = 'utf-8'

@dataclass
class SessionCredentials:
    auth_data: bytes
    key: bytes

class TCPClient():
    def __init__(self, server_ip: str, server_port: int, pub_key:
EllipticCurvePublicKey, sec_key: EllipticCurvePrivateKey, server_pub_key:
EllipticCurvePublicKey, header_size: int = 10):
        self.server_ip: str = server_ip
        self.server_port: int = server_port
        self.pub_key = pub_key
        self.sec_key = sec_key
        self.server_pub_key = server_pub_key
        self.server_auth_data: bytes = bytes(f"{self.server_ip} {self.server_port}",
MESSAGE_FORMAT)
        self.HEADERSIZE: int = header_size
        self.recieved_messages: List[bytes] = list()
        self.messages_to_send: List[bytes] = list()
        self.session_cred: Optional[SessionCredentials] = None

    def connect(self):
        self.connection: socket.socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        print(f'[TCP CLIENT] connecting to {self.server_ip}, port {self.server_port}')
        self.connection.connect((self.server_ip, self.server_port))
        print('[TCP CLIENT] connected!')

    def send_bytes(self, bytes_to_send: bytes):
        header = bytes(f'{len(bytes_to_send):<{self.HEADERSIZE}}',  MESSAGE_FORMAT)
        self.connection.send(header + bytes_to_send)

    def recieve_message(self) -> Optional[bytes]:
        message_header = self.connection.recv(self.HEADERSIZE)
        if not len(message_header):
```

```
    return None
return self.connection.recv(int(message_header))
```

ДОДАТОК Ж

Вихідний код функцій асиметричних криптографічних перетворень

```
from typing import List, Optional, Tuple
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import ec
from cryptography.hazmat.primitives.asymmetric.ec import EllipticCurvePrivateKey,
EllipticCurvePublicKey
from cryptography.hazmat.primitives.kdf.hkdf import HKDF
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.backends import default_backend
from cryptography import exceptions as crypt_exc


def gen_ec_keys() -> Tuple[EllipticCurvePrivateKey, EllipticCurvePublicKey]:
    """
    generate and return (private_key, public_key)
    """
    private_key = ec.generate_private_key(ec.SECP384R1())
    return (private_key, private_key.public_key())


def derive_key(private_key: EllipticCurvePrivateKey, peer_pub_key:
EllipticCurvePublicKey) -> bytes:
    derived_key = HKDF(
        algorithm=hashes.SHA512(),
        length=64,
        salt=None,
        info=None,
    ).derive(private_key.exchange(ec.ECDH(), peer_pub_key))

    return derived_key


def gen_certificate_files(password: Optional[bytes] = None) -> None:
    sec_key, pub_key = gen_ec_keys()

    with open('id_ec.private', 'wb') as f:
        f.write(serialize_ec_sec_key(sec_key, password))

    with open('id_ec.pub', 'wb') as f:
        f.write(serialize_ec_pub_key(pub_key))


def load_sec_key_from_file(path: str = './cred/my_id_ec.private', password:
Optional[bytes] = None) -> EllipticCurvePrivateKey:
```

```
    with open(path, 'rb') as f:
        return load_ec_sec_key(f.read(), password)


def load_pub_key_from_file(path: str = './cred/my_id_ec.pub') -> EllipticCurvePublicKey:
    with open(path, 'rb') as f:
        return load_ec_pub_key(f.read())


def serialize_ec_pub_key(pub_key: EllipticCurvePublicKey) -> bytes:
    return pub_key.public_bytes(
        serialization.Encoding.PEM,
        serialization.PublicFormat.SubjectPublicKeyInfo)


def serialize_ec_sec_key(sec_key: EllipticCurvePrivateKey, password: bytes) -> bytes:
    return sec_key.private_bytes(
        serialization.Encoding.PEM,
        serialization.PrivateFormat.PKCS8,
        serialization.BestAvailableEncryption(password))


def load_ec_pub_key(pub_key: bytes) -> EllipticCurvePublicKey:
    return serialization.load_pem_public_key(pub_key)


def load_ec_sec_key(sec_key: bytes, password: Optional[bytes] = None) ->
EllipticCurvePrivateKey:
    return serialization.load_pem_private_key(sec_key, password)


def list_of_authorized_host(path: str = './cred/authorized_hosts') ->
List[EllipticCurvePublicKey]:
    with open(path, 'rb') as f:
        return [load_ec_pub_key(pub_key) for pub_key in f.read().split(b'&')]

def get_authorized_host(path: str = './cred/authorized_hosts') -> bytes:
    with open(path, 'rb') as f:
        return f.read()

def sign_message(message: bytes, private_key: EllipticCurvePrivateKey) -> bytes:
    return private_key.sign(message, ec.ECDSA(hashes.SHA256()))

def verify_message(message: bytes, signature: bytes, public_key: EllipticCurvePublicKey)
-> bool:
    try:
        public_key.verify(signature, message, ec.ECDSA(hashes.SHA256()))
    except crypt_exc.InvalidSignature:
        return False
    else:
        return True
```