МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

МАКСЮТЕНКО ІГОР ВІТАЛІЙОВИЧ

Допускається до захисту:
в. о. завідувача кафедри
прикладної математики та
кібербезпеки,
д – р філософії з математики,
_____ А.В.Луценко
«__»_____20__ р.

РЕАЛІЗАЦІЯ МЕТОДУ ЧАУМА ЗАХИСТУ ПЕРСОНАЛЬНИХ ДАНИХ ДЛЯ ВПРОВАДЖЕННЯ РЕГЛАМЕНТУ ЄС GDPR

Спеціальність 125 «Кібербезпека»

Кваліфікаційна (бакалаврська) робота

Керівник:
д-р.техн.наук, професор,
професор кафедри прикладної
математики та кібербезпеки
Крижановський В. Г.

Оцінка:___/___/_____
(бали за шкалою ЄКТС/за національною шкалою)
Голова ЕК:_____
(підпис)

Вінниця – 2024

АНОТАЦІЯ

Максютенко І.В. Реалізація методу Чаума захисту персональних данних для впровадження регламенту ЄС GDPR. Спеціальність 125 “Кібербезпека”.Донецький національний університет імені Василя Стуса, Вінниця 2024. У кваліфікаційній (бакалаврській) роботі досліджено проблему анонімності персональних даних. Проаналізовано методи анонімізації та реалізовано змішану мережу для анонімізації. Ключові слова: Змішані мережі, анонімність, DKG, GDPR. 54 с., 4 табл., 8 рис., 1 додаток, 41 джерело.

ABSTRACT

Maksiutenko I.V. Implementation of the Chaum method of personal data protection for the implementation of the EU GDPR. Speciality 125 ‘Cybersecurity’. Vasyl Stus Donetsk National University, Vinnytsia 2024. The qualification (bachelor's) work investigates the problem of anonymity of personal data. Anonymisation methods are analysed and a mixed network for anonymisation is implemented. Keywords: Mixed networks, anonymity, DKG, GDPR. 54 pages, 4 tables, 8 figures, 1 appendix, 41 sources.

ЗМІСТ

ВСТУП

Актуальність роботи: Недоторканність приватного життя - це право зберігати інформацію про своє особисте життя в таємниці. Історично це проголошено як фундаментальне право людини у Загальній декларації прав людини. В сучасному цифровому світі закони про захист даних і недоторканність приватного життя прийняті в більшості країн світу, загальний регламент щодо захисту даних 2018 року (GDPR) є одним з таких прикладів.

Недоторканність приватного життя дає людям можливість висловлювати свої думки без надмірного контролю. Ніхто не повинен жити в постійному страху, що їх особисті висловлювання можуть спровокувати каральні дії з боку вищих сил.

Інфраструктура Інтернету не була розроблена для забезпечення анонімності: метадані, такі як IP-адреси, відправляються у вигляді відкритого тексту в IP-заголовках, і можуть використовуватись для ідентифікації сторін. Щоб пом'якшити це, було розроблено і розгорнуто кілька
мережевих архітектур що зберігають анонімність.

Мета дослідження: реалізація змішаної мережі для збереження анонімності за регламентом GDPR.

Об’єкт дослідження: Захист персональних данних за стандартами GDPR.

Предмет дослідження: Змішані мережі для анонімізації користувачів.

Практичне значення одежраних результатів: Інтеграція в комунікаційні систими для забезпечення безпеки та анонімності користувачів.

РОЗДІЛ 1. НЕОБХІДНІСТЬ АНОНІМНОСТІ ПЕРСОНАЛЬНИХ ДАНИХ

1.1 Персональні дані і GDPR

Персональні дані – це будь-яка інформація, що стосується конкретної фізичної особи, за допомогою якої можна її ідентифікувати. У визначенні норм GDPR сюди відносяться як відомості, які користувач самостійно надав для того чи іншого веб-ресурсу (ім'я та прізвище, стать, email або номер телефону), так і дані, зібрані автоматично. Наприклад, це інформація про місцезнаходження, про пристрій (включаючи IP-адресу), операційну систему тощо [1].

Крім того, персональними даними в інтернеті вважаються відомості про переглянуті сторінки, пошукові запити, пости в соціальних мережах і всю ту інформацію, на основі якої можна визначити вподобання та інтереси користувача, його соціальний статус, релігійні переконання, політичні погляди тощо. Всю цю інформацію покликані захистити нові правила GDPR.

GDPR – це регламент (складається з 99 статей), який регулює відносини між тими, хто надає свої особисті дані (громадяни ЄС) і тими, хто ці дані збирає, обробляє та використовує у своїй роботі (інтернет-сервіси, веб-ресурси, комерційні та некомерційні компанії, організації).

Підхід до захисту персональних даних у GDPR базується на семи принципах, які були задокументовані ще в 1980 році (у «Керівництві про захист приватного життя та транскордонних потоків персональних даних») і схвалені як ЄС, так і США. У GDPR вони відображені в семи пунктах:

1. **Принцип законності, справедливості та прозорості**. Персональні дані повинні бути отримані законними та справедливими засобами за згодою суб'єкта даних.

2. **Обмеження мети.** Мета збору даних повинна бути вказана під час збору, і дані не повинні використовуватися для нічого іншого, окрім початкового наміру.

3. **Мінімізація даних.** Зібрані дані повинні відповідати заданій спочатку меті. Забороняється збирати дані в більшому обсязі, ніж це потрібно для досягнення мети.

4. **Точність.** Персональна інформація повинна бути точною, повною та актуальною, наскільки це необхідно для заданих цілей. Якщо такі дані будуть вважатися неточними, вони повинні бути стерті або виправлені (на запит користувача).

5. **Обмеження зберігання**. Дані зберігаються у формі, яка дозволяє ідентифікувати користувача не довше, ніж це необхідно для виконання цілей обробки інформації.

6. **Цілісність та конфіденційність.** Особисті дані повинні бути захищені гарантіями безпеки від таких ризиків, як втрата або несанкціонований доступ, знищення, використання, модифікація або розкриття даних.

7. **Підзвітність.** Контролер несе відповідальність і повинен бути готовим продемонструвати дотримання заходів, зазначених вище.

1.2 Анонімне спілкування

Підслуховувачі можуть повільно створювати профілі активності на основі метаданих спілкування, незалежно від того, чи зашифровані повідомлення, чи ні. Аналіз трафіку може виявити конфіденційну інформацію про учасників, яка не ґрунтується на змісті їхньої розмови. Особистість і зв'язок між відправником і одержувачем, а також частота спілкування можуть бути корисною інформацією для супротивника. Тому система, яка може приховати цю інформацію, є вкрай необхідною.

Анонімні комунікаційні мережі призначені для того, щоб запобігти спробам спостерігачів відстежувати метадані в спробі пов'язати особу з набором онлайн-транзакцій, тим самим ставлячи під загрозу конфіденційність цієї особи [7, 8, 9]. Мережі досягають певного рівня анонімності відповідно до ступеня від'єднання та неспостережуваності учасників. Терміни повинні бути чітко визначені до того, як концепція анонімної комунікації може бути реалізована на практиці. Pfitzmann & Hansen запропонували робоче визначення, яке використовують в методах перевірки на відповідність GDPR [6];

Анонімність - анонімність суб'єкта з точки зору зловмисника означає, що зловмисник не може достатньою мірою ідентифікувати суб'єкта в множині суб'єктів.

Якщо зловмисник не може виокремити суб'єкта з-поміж усіх можливих суб'єктів, то кажуть, що суб'єкт є анонімним. Анонімність захищає суб'єкта від сторонньої особи, яка намагається отримати особисту інформацію про нього, і в цьому сенсі поняття анонімності нерозривно пов'язане з супротивником.

Для комунікаційних систем (наприклад, чатів) відправник повідомлення є анонімним, якщо зловмисник не може визначити походження повідомлення. Хоча зловмисник часто може спостерігати, які повідомлення проходять через анонімну комунікаційну мережу, він не повинен бути в змозі співставити повідомлення з відправником.

Оскільки анонімні комунікаційні мережі значною мірою покладаються на концепції, визначені вище, їх можна використовувати як метрику безпеки. Пфіцманн і Хансен [10] розглядають анонімність з точки зору супротивника і стверджують, що системи не можуть бути анонімними, доки зловмисник не описаний і не охарактеризований. Таким чином, всі запропоновані системи захищають від певного типу супротивника, але їх дизайн іноді робить їх вразливими для інших

різновидів атак.

У літературі пропонується широкий спектр мережевих проектів і стратегій, орієнтованих на конкретні потреби в анонімному спілкуванні, і деякі з цих пропозицій були реалізовані [13]. Реалізації часто використовують такі системи, як мікси або цибулеві маршрутизатори, наприклад, анонімний ремейлер Mixminion [11] або мережа Tor [12] - сервіс з низькою затримкою, що використовується, наприклад, для анонімного перегляду веб-сторінок. У цьому тексті буде розглянуто і реалізовано один конкретний тип анонімної комунікаційної мережі: мікс-мережу.

1.3 Основна суперечність якості анонімності

Досягнення максимальної анонімності в Інтернеті це складне завдання через велику кулькість факторів які потрібно врахувати. Більшість сервісів надає лише частковий захист, і навіть у мережі Tor, яка вважається однією з найбезпечніших, ідентифікація користувача все ж можлива. Деякі VPN-провайдери також можуть відстежувати та зберігати історію користувачів, що порушує їхню приватність.

Інструменти анонімізації зазвичай призводять до зниження зручності користування через зменшену швидкість з'єднання та обмеження функціональності браузера. Рекомендації щодо підвищення безпеки можуть включати вимкнення потенційно небезпечних функцій, але це може зробити неможливим нормальне функціонування деяких веб-сайтів, що використовують JavaScript та Cookies [4].

Сам факт використання анонімізації може стати помітним та привернути увагу, що ускладнює доступ до деяких ресурсів. Більшість сайтів блокують доступ з IP-адрес Tor.

1.4 Міх сервери

У 1981 році Чаум представив сервер ретрансляціх, який можна було використовувати для надсилання анонімних електронних листів і який він назвав його мікс [14]. Мікс змінює зовнішній вигляд і розташування набору вхідних повідомлень перед тим, як відправити їх одержувачу. Повідомлення шифруються за допомогою криптографії з відкритим ключем, а потім переставляються, що робить відправника і одержувача не пов'язаними між собою. Зокрема, головною метою змішаних мереж є досягнення побітової незв'язності. Має бути неможливо пов'язати бітові патерни закодованого повідомлення, що надходить на сервер, з бітовими патернами повідомлення, що виходить з сервера [16].

Ця ідея не спрацьовує, коли супротивник перехоплює контроль над сервером. Тому замість того, щоб використовувати лише один мікс, часто використовують мережу міксів. Чаум запропонував з'єднати кілька міксів у фіксований шлях, який він назвав каскадом, через який має пройти кожне повідомлення. Перш ніж повідомлення буде відправлено через мережу, воно шифрується за допомогою відкритого ключа кожного міксу, таким чином створюючи багаторівневе шифрування. Кожен мікс у каскаді розшифровує один рівень шифрування за допомогою свого закритого ключа, отримує інформацію про місце призначення і надсилає повідомлення наступному міксу. Принцип багаторівневого шифрування також використовується в onion routing - техніці для анонімних додатків з низькою затримкою, таких як веб-браузер Tor [12]. Реалізації змішаних мереж зазвичай мають більшу затримкою, але вважаються більш безпечними [13].

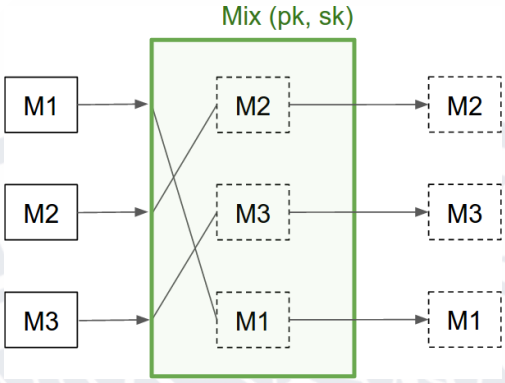

Рисунок 1.1 - Базовий мікс, що визначається відкритим ключем *pk* та закритим ключем *sk*.

Мікс змінює вигляд вхідних повідомлень за допомогою криптографії з відкритим ключем і змінює порядок повідомлень таким чином, щоб отримати побітову незв'язність вхідних і вихідних повідомлень. У мікс-мережі кожен мікс надсилає свої вихідні повідомлення іншому міксу [11].

Крім того, мікс, що використовує базове шифрування RSA, вразливий до атак тегування, в яких блоки зашифрованих текстів замінюються або дублюються, щоб знайти зв'язний шаблон з одним із відкритих текстів [15].

У літературі є багато статей, в яких обговорюються властивості безпеки міксів. Хоча мікс Чаума має деякі недоліки [39, 10], він використовується як основа для багатьох інших стратегій змішування, які були запропоновані для протидії різним недолікам. Однією з можливих варіацій міксу Чаума є зміна стратегії вибору вузлів. Мікс Чаума надсилає повідомлення через фіксований шлях вузлів, але в мережі з вільним маршрутом послідовність вузлів може бути будь-якою комбінацією міксів. Це дозволяє реалізувати різний ступінь випадковості: або початковий мікс обирає послідовність маршрутизації (вихідна маршрутизація), або кожен мікс визначає наступний мікс (маршрутизація "стрибками") [13].

Питання про те, коли мікс пересилає свій пакет наступному міксу, є важливим для поняття неспостережуваності. Мікс Чаума чекає доки

наберется n повідомлень, перш ніж відправити їх на наступний сервер, але це робить мережу вразливою до серії атак. Мало того, що порогове змивання призводить до високої затримки в системах з низьким навантаженням на трафік, цей метод вразливий до (n - 1) атаки, для якої зловмисник надсилає n - 1 фальшиве повідомлення і здатен скомпрометувати особу відправника і одержувача повідомлення, яке не було надіслано супротивником [16]. Флеш-стратегії тому працюють із затримкою повідомлень індивідуально або наборами. Хоча такі таймінгові мікси краще захищають від попередньої атаки, вони не забезпечують адекватної безпеки у випадку низького трафіку [13]. Щоб уникнути цього, останні реалізації, такі як Mixmaster [17] і Mixminion [11], поєднують порогові та часові процедури, а інші додають фіктивний трафік (підроблені повідомлення), щоб уникнути низького навантаження на трафік [18].

1.5 Аналогічні решення для набуття анонімності

Проксі-сервер - це посередник між клієнтом і кінцевим адресатом. У контексті анонімності можна виділити настуні види проксі-серверів [3]:

- HTTP-проксі-сервери: ці сервери обробляють виключно HTTP-трафік.

- SOCKS-проксі-сервери: на відміну від HTTP-проксі, вони передають всю інформацію без змін.

- CGI-проксі або «анонімайзери»: це веб-сервери з формою, в яку клієнт вводить адресу необхідного сайту. Після цього відкривається сторінка запитаного ресурсу, але в адресному рядку браузера відображається адреса CGI-проксі. CGI-проксі можуть використовувати https для захисту з'єднання між собою і клієнтом.

Проксі-сервери мають кілька різновидів із власними

особливостями, але найчастіше для анонімізації використовуються SOCKS5. Однак, їхня надійність вважається обмеженою, оскільки вони не шифрують трафік і можуть легко бути деанонімізовані, навіть при використанні проксі-ланцюга [2].

VPN (Virtual Private Network) — це віртуальна приватна мережа, яка шифрує трафік між користувачем і VPN-сервером та змінює IP-адресу. Коли користувач підключається до VPN, створюється захищений канал між його комп'ютером і VPN-сервером, де дані надійно шифруються. Це заважає інтернет-провайдеру визначати місцезнаходження користувача та відстежувати відвідані ним веб-ресурси. Нова IP-адреса зазвичай видається з іншого міста або країни. VPN-сервіси дозволяють отримувати доступ до ресурсів, які заблоковані за географічними обмеженнями або рішенням влади. Використовуючи VPN, можна вільно відвідувати заблоковані сайти, завантаживши відповідний додаток на свій комп'ютер або мобільний пристрій [1].

VPN-сервіси також застосовуються для анонімізації, але головною проблемою залишається питання довіри до провайдера. Багато VPN-провайдерів заявляють, що не зберігають журнали активності, але перевірити це важко, і часто виявляється, що логування все ж відбувається. Крім того, у разі раптового відключення VPN-з'єднання весь трафік може потрапити в Інтернет напряму, що розкриває реальний IP-адресу користувача [2].

Висновки до 1 розділу

Забезпечення анонімності персональних даних є важливим аспектом сучасного цифрового середовища, особливо в контексті вимог GDPR. GDPR встановлює чіткі правила щодо збирання, обробки та зберігання персональних даних, забезпечуючи права

користувачів та відповідальність контролерів даних. Основними принципами GDPR є законність, справедливість, прозорість, обмеження мети, мінімізація даних, точність, обмеження зберігання, цілісність, конфіденційність та підзвітність.

Анонімне спілкування стає дедалі важливішим у боротьбі з підслуховуванням та аналізом трафіку, які можуть розкрити конфіденційну інформацію про користувачів. Мікс-сервери та інші анонімні комунікаційні мережі, такі як Tor, відіграють ключову роль у забезпеченні анонімності, але мають свої обмеження та вразливості. Для досягнення ефективної анонімності важливо враховувати тип загроз та можливості зловмисників.

Анонімізація в Інтернеті стикається з різними проблемами, такими як зниження швидкості з'єднання, обмеження функціональності та ризики деанонімізації. Інструменти, як-от проксі-сервери та VPN, надають певний рівень анонімності, але їх надійність та ефективність можуть бути обмеженими через різні вразливості та питання довіри до провайдерів.

РОЗДІЛ 2. ПРОЄКТУВАННЯ МЕРЕЖІ НА ОСНОВІ БАГАТОСТОРОННІХ ПРОТОКОЛІВ

2.1 План реалізації мережі

У цій роботі розглядається проектування та реалізація змішаної мережі на основі декількох багатосторонніх обчислювальних протоколів. Ширазі нещодавно запропонував новий протокол маршрутизації, який використовує перевірену функцію відображення для рандомізації наступного місця призначення кожного повідомлення в змішаній мережі. Крім того, запропонована топологія серверів у мережі дозволяє горизонтальне масштабування, відсутність якого створює серйозні проблеми з масштабуванням традиційних реалізацій міксів, таких як мікси з вільним маршрутом [20, 8]. Реалізації систем анонімного зв'язку, таких як Tor [12] і Atom [20], доводять, що горизонтальне масштабування мережі, або можливість збільшення ретрансляції повідомлень, має важливе значення для побудови масштабованої мережі анонімного зв'язку.

Для того, щоб виконати всі завдання в системі зв'язку, визначено три основні ролі в системі: ретранслятори, маршрутизатори та аудитори. Мікси відповідають за перемішування і повторне шифрування пакетів повідомлень і за обчислення доказів з нульовим знанням. Об'єкти маршрутизації встановлюють спільну випадковість, необхідну для протоколу маршрутизації, а аудитори виконують перевірку опублікованих значень. Суб'єкти можуть спілкуватися через дошку оголошень або загальнодоступну дошку повідомлень, з якої всі суб'єкти можуть читати і публікувати повідомлення.

Для реалізованої мережі змішування всі сервери виконують всі три ролі. Це означає, що будь-який сервер в системі може виконувати змішування, розраховувати нові маршрути і перевіряти значення на

дощці оголошень. Всі криптографічні примітиви і протоколи були реалізовані з нуля таким чином, щоб можна було досягти високого ступеня контролю в структурі коду і в розвитку мережі міксу.

Відповідно до [19], мікси структуровані відповідно до топології ‘Зірка”. Це означає, що мережа структурована на заздалегідь визначену кількість шарів, і кожна партія повідомлень передається в наступний стовпець після кожного періоду часу. На рисунку 2.2 показано, як структуровані мікси. Вхідні мікси (синього кольору) отримують зашифровані повідомлення від користувачів і надсилають їх до міксів у наступних стовпчиках (зеленим і червоним кольором). Хоча немає різниці між різними ролями міксу, є невелика різниця між вихідними міксами (червоним кольором). Ці мікси несуть додаткову відповідальність за розшифрування зашифрованих текстів. Вихідні мікси можуть надсилати свої повідомлення або брокеру повідомлень (показано сірим кольором), або безпосередньо одержувачам.

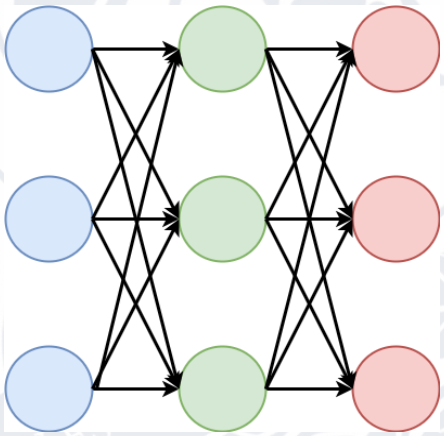

Рисунок 2.1 - Структура міксів з топологією ‘Зірка”

2.2 Розподілена генерація ключів Педерсена

Протоколи генерації ключів є основою криптосистеми. Протокол дозволяє учаснику згенерувати набір ключів, які можна використовувати для шифрування та дешифрування повідомлень. Існує кілька варіантів

генерації ключів для асиметричної криптографії. У першому випадку існує сторона, яка називається дилером, якій довіряють всі учасники. Довірений дилер генерує секретний і відкритий ключі та розподіляє їх між учасниками як секретні частки. Така ситуація малоймовірна в реальних умовах. У другому випадку дилеру не довіряють, і всі сторони повинні мати можливість перевірити правильність дій дилера. Такі протоколи називаються протоколами обміну секретними даними з можливістю перевірки (VSS), наприклад, VSS Фельдмана для гомоморфних криптосистем [21].

Існування третьої сторони, однак, означає, що можлива єдина точка відмови. Якщо дилер є корумпованим, не буде згенеровано жодного дійсного ключа, і, як наслідок, система буде заблокована. Третій варіант вирішує останню проблему шляхом виключення дилера з системи, а ключі генеруються повністю розподіленим способом. Учасники протоколів розподіленої генерації ключів спільно обчислюють криптографічну пару ключів. Замість того, щоб покладатися на третю сторону, учасники повинні спілкуватися один з одним, щоб отримати частку ключів.

Простим протоколом розподіленої генерації ключів є протокол, запропонований Педерсеном []. Як і схема VSS Фельдмана, він базується на пороговій схемі обміну секретами Шаміра [23]. Шамір розробив схему, в якій можна розділити дані на t частин таким чином, щоб було легко відновити дані шляхом компіляції частин, але складно компілювати дані, маючи не більше t - 1 частини. Схема поділу базується на поліноміальній інтерполяції, або на ідеї, що поліном степеня t - 1 визначається за t точками. Формально, щоб розділити секрет $s \in Z_p$ між n учасниками на частки $s_1 , \ldots , s_n$, вибираємо t - 1 випадкові коефіцієнти $a_1 , \ldots , a_{t-1}$ такі, що f є випадковим поліномом степеня t - 1:

$$f(z) = s + a_{i1}\, z + \ldots + a\, z_{it-1}^{t-1}$$

(2.1)

Потім секретні частки обчислюються як

$$s_i = f(i) \text{ для } \quad i = 1, \ldots, n$$

(2.2)

За Шаміром, для відновлення секрету *s* достатньо лише підмножини з принаймні *t* часток.

У схемі Шаміра дилер відповідає за обчислення полінома і розподіл часток. Протокол розподіленої генерації ключів Педерсена не вимагає участі третьої сторони і може бути легко використаний у поєднанні з пороговою криптосистемою, наприклад, пороговою ElGamal [24]. Як і схема Шаміра, протокол Педерсена базується на скінченному полі Z_p простого порядку p. Доведено, що протокол розподіленої генерації ключів Педерсена разом з криптосистемою ElGamal задовольняє припущенню Decisional Diffie-Hellman (DDH) [25]. Припущення DDH – це припущення про обчислювальну складність, яке стверджує, що при заданих g^a та g^b для $a, b \in Z_p$ вибрано рівномірно навмання, g^{ab} обчислювально не відрізняється від випадкової величини у Z_p .

Розподілена генерація ключів Педерсена (DKG) є синхронним протоколом, що означає, що кожен учасник повинен дочекатися завершення протоколу крок за кроком, перш ніж фактична пара ключів може бути побудована, але це дозволяє попередньо виконати багато обчислень в автономному режимі. Нещодавно були запропоновані більш масштабовані асинхронні протоколи DKG для роботи в реалістичних умовах, таких як Інтернет, які намагаються обмежити обчислювальні вузькі місця [26]. Однак, іншою проблемою багатосторонніх протоколів є їх складність і труднощі з реалізацією ефективних реалізацій. На відміну від [26], DKG є простим і досить легким для розуміння, і може бути легко замінений на більш складний протокол.

Реалізація протоколу розділена на три етапи: етап генерації параметрів домену, офлайн-фаза та онлайн-фаза. Протокол передбачає, що трійка параметрів домену *(p, q, g)* вже згенерована і є загальнодоступною через дошку оголошень. Будь-хто з учасників може попередньо обчислити ці параметри до запуску протоколу. Спочатку вибирається просте число *p* бітової довжини *b,* яке визначає порядок скінченного поля Z_p . Криптосистема буде шифрувати і розшифровувати повідомлень через Z_p . По-друге, генерується ще одне просте число *q* таке, що $q \mid p - 1$ і визначає підгрупу Z_q з Z_p . По-третє, генерується генератор *g* підгрупи Z_q. Генератор циклічної групи має таку властивість, що будь-який елемент Z_q можна обчислити за степенями *g*.

Алгоритм 1 – Генерація параметрів домену [38]:

Маючи довжину біт криптографічного ключа *b*, кожен M_i генерує трійку параметрів домену *(p, q, g)* таких, що

1.*p* - просте число з бітовою довжиною *b,* яке визначає скінченне поле Z_p простого порядку *p*.

2.*q* - велике просте число, таке що $q \mid p$ - 1.

3.*g* - генератор підгрупи простого порядку *q*. *g* - генератор циклічної групи, якщо всі елементи цієї групи можна побудувати за степенями *g*.

Для порогового значення *t* протокол Педерсена використовує схему секретного розподілу Шаміра з додатковою фазою перевірки. В автономній фазі кожен учасник генерує випадковий поліном степеня *t* - 1 і обчислює частки для кожного з інших учасників. Він також транслює набір верифікаційних значень X_{ik} , які доводять, що були розподілені правильні частки.

$$X_{ik} = g^{a_{ik}} \bmod p \text{ для } k = 0, \ldots, t - 1 \qquad (2.3)$$

Оскільки верифікаційні значення містять інформацію про таємне значення учасника та його частки, аудитор, який отримав частку, може перевірити її правильність, перевіривши, чи є вона дійсною:

$$g^{\bar{x}_{ij}} = \prod_{k=0}^{t} (X_{ik})^{j^k} \, mod \, p \qquad (2.4)$$

Якщо перевірка не проходить, на учасника подається скарга. Якщо принаймні *t* учасників подали скаргу на учасника, він публікує свою частку. Якщо перевірка знову не підтверджує опубліковану частку, учасника дискваліфікують. Це означає, що його частки не беруть участі в компіляції секретного і відкритого ключів.

Коли учасники перевірили всі інші частки, вони можуть компілювати криптографічні ключі, використовуючи частки і значення верифікації кваліфікованого набору учасників Q(*n)*. Відкритий ключ дорівнює добутку перших перевірочних значень всіх учасників. Він однаковий для всіх учасників.

$$y = \prod_{i \in Q} X_{i0} \, mod \, p \qquad (2.5)$$

Приватний ключ унікально визначається для кожного учасника як сума отриманих секретних часток.

$$x_i = \sum_{j \in Q} \bar{x}_{ij} \, mod \, q \qquad (2.6)$$

Секрет можна отримати лише за допомогою *t* + 1 частки. Тим не менш, криптосистема, реалізована для мікс-мережі, не потребує перекомпіляції секрету. Насправді, фактичний секретний ключ ніколи не компілюється. Це дозволяє повністю уникнути єдиної точки відмови. Для протоколу Педерсена були виявлені й інші проблеми. Дженнаро та ін. [27] пишуть, що DKG не може генерувати відкриті та секретні ключі з рівномірним розподілом, а потім показують, що противник, який скомпрометував лише два сервери, може впливати на генерацію криптографічних ключів. Дженнаро та ін. [29] пропонують розширення протоколу, яке додає ще один раунд комунікації, що ще більше збільшує обчислювальну вартість протоколу. Для виробничого середовища, однак, DKG Педерсена все ще

використовується через свою ефективність [24]. Таким чином, реалізація протоколу розподіленої генерації ключів зводиться до компромісу між безпекою та практичною ефективністю.

Згідно з [28], безпека протоколу розподіленої генерації ключів Педерсена є дещо незрозумілою. Протокол дозволяє зловмиснику мати невеликий контроль над відкритими ключами. Це пов'язано з тим, що кожен учасник може спостерігати за розподілом відкритих ключів, які генерує протокол. Тим не менш, Дженнаро та ін. вказують, що безпека, яку забезпечує DKG Педерсена, є "досить хорошою" [29], оскільки противник має лише незначні зміни, щоб керувати значенням відкритого ключа в множині поліноміального розміру і зламати дискретний логарифм.

Дженнаро та ін. [29] довели, що розподілена генерація ключів Педерсена разом з алгоритмом підпису Шнорра є безпечною, враховуючи доказ безпеки DKG Педерсена в моделі випадкового оракула. Крім того, підпис Шнорра є стійким до підробки, що унеможливлює підробку підписів шифротекстів без порушення протоколу. Отже, хоча DKG Педерсена має недоліки, він добре працює зі схемою підпису Шнорра і має перевагу масштабованості. Таким чином, можна вибирати більші криптографічні ключі без необхідності платити за це з точки зору ефективності.

Алгоритм 2 – розподілена генерація ключів [38]:

Мікс сервери M_i для $i = 1, \ldots, n$ мають набір параметрів домену Педерсена (p, q, g), їм задано поріг t, вони мають доступ до дошки оголошень **BB** та до всіх інших серверів через список серверних адрес.

Налаштування

1.Кожен учасник M_i генерує випадковий поліном f степеня t над Z_q такий, що:

$$f_i(z) = a_{i0} + a_{i1}z + \ldots + a_{it}z^t \qquad (2.7)$$

Зауважте, що секрет x_i , яким бажає поділитися M_i , входить до

складу полінома у вигляді $x_i = a_0$.

2.Кожен учасник передає на дошку оголошень **BB** набір перевірочних значень. Вони будуть використані для перевірки отриманих секретних акцій.

$$X_{ik} = g^{a_{ik}} \, mod \, p \text{ для } k = 0, \ldots, t-1 \qquad (2.8)$$

3.Кожен M_i обчислює секретні частки $\overline{x}_{ij} = f_i(j) \bmod p$ для всіх інших учасників $j = 1, \ldots, n$ і відправляє $\overline{x}_{ij}$ на M_j

Верифікація - вожен M_j верифікує частки, отримані від M_i , використовуючи значення верифікації M_i , опубліковані на дошці оголошень:

$$g^{\overline{x}_{ij}} = \prod_{k=0}^{t} (X_{ik})^{j^k} \, mod \, p \qquad (2.9)$$

M_j відхиляє частку, якщо ця перевірка не пройшла. Учасник M_i дискваліфікується, якщо принаймні t учасників відхилили його пропозицію на дошці оголошень.

Компіляція ключів - відкритий ключ y системи може бути скомпільований з використанням верифікаційних значень всіх кваліфікованих учасників Q:

$$y = \prod_{i \in Q} X_{i0} \, mod \, p \qquad (2.10)$$

Кожен учасник M_i формує свою секретну частку x_i наступним чином:

$$x_i = \sum_{j \in Q} \bar{x}_{ij} \, mod \, q \qquad (2.11)$$

для якого $\overline{x}_{ii} = f_i(i)$. Якщо учасника M_j дискваліфіковано, то $y_j = 1$ і $x_j = 0$. Тепер для правильного відновлення секрету потрібно лише $t + 1$ частки.

2.3 Криптосистема Threshold ElGamal

Криптосистема ElGamal [30] - це криптосистема з відкритим

ключем, розширена з протоколу Діффі-Хеллмана, яка базується на проблемі дискретного логарифму. Задача дискретного логарифмування постулює, що для секрету x і генератора g дуже легко обчислити $h = g^x$, але дуже важко знайти x *на* основі h. Більш конкретно, задача дискретного логарифмування є NP-важкою у скінченних полях Z_p для великих простих чисел p [23].

Криптосистема ElGamal є одночасно простою та ефективною. Для шифрування відкритого тексту m генерується випадкове число $r \in Z_p$ і створюється текст шифру ElGamal (c_1 , c_2) з

$$(c_1 , c_2) = (g^r , m \cdot y^r) \, mod \, p \qquad (2.12)$$

для y - відкритий ключ. Таким чином, зашифрований текст ElGamal завжди вдвічі довший за відкритий текст. Це важливо з точки зору навантаження на пам'ять системи з урахуванням розміру криптографічних ключів і простого порядку p.

Перевагою криптосистеми ElGamal, яка буде широко використовуватися для мікс-мережі, є її гомоморфна властивість [22]. Оскільки шифрування ElGamal використовує випадкові числа, шифрування одного і того ж відкритого тексту завжди буде різним, і зашифрований текст може бути знову зашифрований в дійсний шифрований текст ElGamal.

$$(c_1', c_2') = (c_1 \cdot g^8, c_2 \cdot y^8) \, mod \, p \; = (g^{r+8}, m \; \cdot y^{r+8}) \, mod \, p \qquad (2.13)$$

Повторне шифрування можна просто розшифрувати, використовуючи той самий секретний ключ. Для базової криптосистеми ElGamal $y = g^x$ для x секретний ключ. Це означає, що зашифрований текст ElGamal можна розшифрувати за допомогою:

$$m = c_2 \cdot c_1^{-x} \qquad (2.14)$$

Оскільки кожне обчислення буде виконуватися над Z_p , важливо вибрати достатньо велике просте число, щоб залишатися безпечним. В даний час довжина біта для асиметричної криптографії, яка використовує обмін ключами Діффі-Хеллмана, вважається безпечною на

рівні 1024 біт, але, згідно зі звітом NIST, для захисту урядової інформації дозволено використовувати тільки прості числа довжиною 2048 біт [13]. Тому важливо знайти масштабовані та ефективні алгоритми, які забезпечують безпеку і можуть бути обчислені за розумний час.

Для подальшого підвищення безпеки можна використовувати розподілену генерацію ключів, щоб зняти відповідальність однієї зі сторін за використання секретного ключа для розшифрування. Всі сторони розраховують секретну частку і уникають проблеми єдиної точки відмови. У цьому випадку нам знадобиться лише поріг t + 1 частка для розшифрування зашифрованого тексту. Але замість того, щоб вирішувати складну проблему, коли всі учасники повинні передати свою частку одній зі сторін, можна скористатися протоколом розподіленого дешифрування, щоб взагалі уникнути обчислення секрету. Секрет ніколи не обчислюється, хоча принаймні t + 1 учасник повинен надати стороні, що розшифровує, достатньо інформації, щоб ефективно розшифрувати зашифрований текст [17, 31].

Порогова криптосистема ElGamal - це розширення базової криптосистеми ElGamal, яке дозволяє розподілене дешифрування з використанням спільного доступу до секретів. Вона реалізована в поєднанні з DKG Педерсена і забезпечує елегантний спосіб реконструкції відкритого тексту без необхідності розкривати секрет.

Шифрування та перешифрування відбувається так само, як і в криптосистемі ElGamal. Щоб розшифрувати зашифрований текст, всі учасники M_i повинні обчислити і опублікувати частку розшифровки, використовуючи власну секретну частку і частину зашифрованого тексту:

$$d_j = c_1^{x_j} mod\ p \tag{2.15}$$

Оскільки секретні частки створюються з полінома, D може відновити відкритий текст, використовуючи інтерполяцію Лагранжа

частки розшифрування. Множина Q - це множина кваліфікованих учасників, тобто тих міксів, які не були дискваліфіковані в процесі генерації ключів [25].

$$m = c_2 \cdot \left(\prod_{j \in Q}^{n} d_j^{L_j(Q)} \right)^{-1} \quad mod\ p \tag{2.16}$$

for

$$L_i(Q) = \prod_{l \in Q \setminus \{j\}} \frac{l}{l-j}\ mod\ q \tag{2.17}$$

Кортьє та ін. [25] показують, що, хоча DKG Педерсена не завжди повертає рівномірно випадкові ключі, повністю розподілена система ElGamal у поєднанні з Pedersen DKG є семантично безпечною за припущенням DDH. Крім того, доведено, що криптосистема є безпечною і повною.

Брандт та ін. [31] пропонують сторонам не лише публікувати результати розшифрування, але й публікувати доказ з нульовим знанням, щоб перевірити, чи правильний дискретний логарифм. Брандт зосереджується на паралельному виконанні ∑-протоколу, що прискорює обчислення. Використовуючи евристику Фіата-Шаміра, ці інтерактивні доведення з нульовим знанням можна зробити неінтерактивними, щоб обмежити кількість раундів спілкування між учасниками. Додатковою перевагою цього є те, що учасник може опублікувати доведення на дошці оголошень, і будь-яка інша сторона може перевірити його правильність.

Щоб довести коректність дискретного логарифма неінтерактивним способом, припускається, що верифікатор *P* і будь-який верифікатор *V* знають *c* і *d* такі, що $d = c^x \bmod p$ для приватного значення *x, яке* знає тільки *P.* Далі виконуються наступні кроки:

1.*P* вибирає випадкове число *r* і обчислює $a = c^r \bmod p$.

2.*P* обчислює $h = H(a, c, d)$, де H - публічна хеш-функція.

3.*P* обчислює $b = r + hx \bmod p$ і публікує *(a, b)* на дошці оголошень.

Використовуючи H, будь-який V може перевірити правильність дискретного логарифма P, обчисливши $h = H(a, c, d)$ і перевіривши, чи $c^b = ad^h$.

Якщо H - випадковий оракул, то V не може знайти x, окрім як розв'язавши задачу дискретного логарифмування.

Припущення DDH постулює, що для заданої циклічної групи Z_p , генератора g і $a = g^x$, $b = g^y$, знайти g^{xy} важко [29]. Крім того, доведено, що криптосистема є безпечною та повною.

Проблема з криптосистемою виникає, якщо зловмисник контролює декілька міксів. Зокрема, якщо скомпрометовано більше ніж $t/2$ міксів, зашифровані тексти можуть бути розшифровані. Потрібно знайти компроміс між збільшенням порогу, щоб не дати противнику розшифрувати зашифровані тексти, і достатньо низьким порогом, щоб не знизити обчислювальну ефективність. Якщо поріг t занадто великий, противник може просто заблокувати процес розшифрування, не відповідаючи на запити сторони, що розшифровує. Високий поріг означатиме, що для розшифрування повідомлення знадобляться майже всі ключі (включно з тими, що зберігаються у супротивника). Тоді зловмисник зможе зупинити подальшу роботу системи.

Алгоритм 3 – Порогова криптосистема ElGamal [25]:

Кожен мікс M_i має доступ до набору параметрів домену (p, q, g), відкритого ключа y та секретного ресурсу x_i . Сервер дешифрування D може бути як міксом, так і окремим сервером дешифрування.

Шифрування - відкритий текст m можна зашифрувати за допомогою шифру Ель-Гамаля, використовуючи випадкове число $r \in Z_p$ таке, що:

$$(c_1 , c_2) = (g^r , m - y)^r \mod p \qquad (2.18)$$

Повторне шифрування - шифротекст ElGamal (c_1 , c_2) можна перешифрувати за допомогою іншого випадкового $s \in Z_p$ такого, що:

$$(c_1', c_2') = (c_1 \cdot g^8, c_2 \cdot y^8) \; mod \; p \qquad (2.19)$$

Розшифрування - Щоб розшифрувати зашифрований текст (c_1 , c_2), D запитує частку розшифровки у всіх інших міксів M_j :

$$d_j = c_1^{x_j} mod\ p \tag{2.20}$$

M_j повинен довести правильність частки дешифрування, опублікувавши доказ дискретного логарифма з нульовим знанням [19].

Відновлення - Коли всі мікси надіслали частку розшифровки, D може відновити відкритий текст, використовуючи інтерполяцію Лагранжа розшифровки частки. Множина Q - це множина кваліфікованих учасників, тобто тих міксів, які не були дискваліфіковані в процесі генерації ключів.

$$m = c_2 \cdot \left(\prod_{j \in Q}^{n} d_j^{L_j(Q)} \right)^{-1} \ mod\ p \tag{2.21}$$

for

$$L_i(Q) = \prod_{l \in Q \setminus \{j\}} \frac{l}{l - j}\ mod\ q \tag{2.22}$$

2.4 Перевірка коректності перемішування

Видалення центрального елемента в протоколі дозволяє уникнути єдиної точки збою. Зловмисник більше не має можливості корумпувати дилера і таким чином розголошувати секрети. Замість цього йому потрібно зламати кілька серверів, перш ніж він отримає будь-яку цінну інформацію.

Основною функцією міксу залишається зробити так, щоб набір вхідних повідомлень не відрізнявся від набору вихідних повідомлень, тому, якщо противник контролює мікс, він може просто замінити повідомлення в міксі так, щоб ніхто про це не дізнався. У такому випадку наступні сервери несвідомо обробляють пошкоджену партію,

дозволяючи зловмиснику відстежувати його повідомлення по всій системі разом з правильним повідомленням, тим самим компрометуючи особи відправників. Тому нам потрібно мати можливість перевірити, чи правильно мікс перетасував і повторно зашифрував свою партію повідомлень. Це можна зробити за допомогою тестів з нульовим знанням.

Іншими словами, нам потрібно довести, що для перешифрування кожного зашифрованого тексту використовується певний набір випадкових чисел, і що певна перестановка використовується для перемішування колекції перешифрованих текстів. Формально, для набору вхідних даних зашифрованих текстів (u_1 , . . . , u_n) та набору вихідних зашифрованих текстів ($\acute{u}_1$,…, $\acute{u}_2$), то перевірювач *P* повинен довести верифікатору *V*, що він знає перестановку π та випадкові числа (r_1 , ... , r_n) такі, що$u_i' = u_{\pi(i)} \cdot E_{pk}(1, r_{\pi(i)})$, де E_{pk} - функція шифрування ElGamal з відкритим ключем *pk*.

Різні версії цього доведення з нульовим знанням були опубліковані в [32] та [33]. Ці доведення є неінтерактивними, що означає, що існує мінімальна кількість комунікації між тим, хто доводить і тим, хто перевіряє. Доведення з нульовим знанням зазвичай покладаються на декілька раундів спілкування між доведенням і верифікатором на основі схеми "зобов'язання-виклик-відповідь". Верифікатор вибирає випадковий виклик *w,* який повинен використати верифікатор для генерації доказу. Доказ надсилається верифікатору, і відповідь перевіряється. Однак, використовуючи евристику Фіата-Шаміра, комунікацію можна скоротити, замінивши схему "зобов'язання-виклик-відповідь" на одноразундове хешування підпису [28, 29]. Перевіряючий вибирає випадковий виклик і обчислює відповідь на основі оцінки публічної хеш-функції H, до якої може отримати доступ будь-який верифікатор. Після цього будь-який верифікатор може обчислити той самий хеш і може перевірити правильність доведення. Це називається

неінтерактивним доведенням з нульовим знанням, яке, як відомо, є безпечним у моделі випадкового оракула [9]. Для зниження затримо, слід використовувати неінтерактивне доведення з нульовим рівнем знань для реалізації мікс-мережі. Аргументи з нульовою невідомістю в загальному випадку обчислювально складні і несуть велике навантаження на мережу [33]. Вікстром [32] та Байєр і Грот [33] нещодавно запропонували неінтерактивні доведення коректності перемішування з нульовим знанням, і обидва доведення виявилися майже однаково ефективними [33]. Ми вирішили скористатися неінтерактивним доведенням з нульовим знанням з [32], оскільки воно є більш зрозумілим з двох. Крім того, доведення розділено на дві частини: автономну та інтерактивну фази. Під час офлайн фази, доказ P бере на себе зобов'язання щодо перестановки π і доводить знання π і випадкових чисел (r_1 , ..., r_n), які будуть використані для перемішування і повторного шифрування вхідної партії зашифрованих текстів. Під час онлайн-етапу P доводить, що та ж сама перестановка π і рандомізатори (r_1 , . . . , r_n) використовуються для перемішування і повторного шифрування набору з n шифротекстів ElGamal. Протокол Вікстрема [32] цікавий тим, що він розбиває доказ з нульовим знанням на дві дуже схожі частини. Протокол був реалізований у Verificatum, який є власною розробкою Вікстрема, і в UniCrypt [35]. Хаенні та ін. [35, 29] провели ретельне дослідження доведення Вікстрема. Ці роботи зробили доведення більш зручним для інтеграції в реалізацію. Огляд протоколу можна знайти в Алгоритмі 3. Загальна ідея доведення полягає в тому, що доведенню спочатку потрібно зафіксувати перестановку π і випадкові числа (r_1 , ..., r_n), які будуть використані при перетасуванні і повторному шифруванні вхідної партії зашифрованих текстів. Доказувач також повинен довести знання π і (r_1 , . . . , r_n), наприклад, верифікатор переконаний, що зобов'язання складено правильно. Після цього верифікатор перетасовує і повторно шифрує зашифровані тексти, публікує їх на

дошці оголошень і доводить знання перестановки і випадкових чисел, використаних для створення вихідної партії. Оскільки доказ перетасування не є інтерактивним, будь-який верифікатор може перевірити правильність доказу і відповідність вихідного пакету зобов'язанням.

Алгоритм 4 – перемішуваня з нульовим знанням [32]:

Розглянемо версифікатора P та верифікатора V . P бажає взяти на себе зобов'язання виконати перестановку π і довести знання π та випадкових чисел $(r_1 , \ldots , r_n)$.

1.V відправляє випадкове число $z \in Z_p$ до P.

$$P \leftarrow \quad z \tag{2.23}$$

P обчислює $e_1 , \ldots , e_n$ такі, що для генератора псевдовипадкових чисел R, $(e_1 , \ldots , e_n) = R(z)$

2.P записує зобов'язання в матрицю перестановок, використовуючи протокол, описаний в Алгоритмі 3.3, і доводить, що зобов'язання складено коректно.

3.P доводить знання про правильність тасування за допомогою Алгоритму 3.3.

4.V може перевірити, чи узгоджується зобов'язання з вихідними даними пакету, перевіряючи, чи рівні публічні значення офлайн та онлайн фаз є рівними [35].

Формально, програмі P спочатку потрібно опублікувати зобов'язання для матриці перестановок A, використовуючи стовпчикові зобов'язання Педерсена. У цьому сенсі P опосередковано повідомляє про використання певної перестановки π і певного набору випадкових чисел $(r_1 , \ldots , r_n)$ які перетасовують та перешифровують зашифровані тексти $(c_1 , \ldots , c_n)$. Зобов'язання Педерсена $(a_1 , \ldots , a_n)$ обчислюються за допомогою набору генераторів $g, g_1 , \ldots , g_n$ з Z_p таких, що

$$(a_1, \ldots, a_n) = \left(g^{r1} g_{\pi^{-1}(1)}, \ldots, g^{r_n} g_{\pi^{-1}(n)}\right) \tag{2.24}$$

Потім P обчислює узагальнене зобов'язання Педерсена $a =$

$\prod_i^n a_i^{e_i}$ і публікує a.

Потім P повинен довести, що він знає π і (r_1 , ... , r_n) так, що зобов'язання складено коректно за допомогою ∑-протоколу [35, 34]. Деталі обчислень наведено в Алгоритмі 3.3. Публічне значення a буде використовуватися для перевірки відповідності вихідних шифротекстів зобов'язанням.

Алгоритм 5 – Зобов'язання щодо матриці перестановок [32]:

Розглянемо перестановщика P та верифікатора V . P бажає взяти на себе зобов'язання щодо перестановки π та випадкових чисел (r_1 , . . . , r_n). Розглянемо g, g_1 , . . . , g_n генератори Z_p .

1.P обчислює конусоподібні Педерсеновські зобов'язання для всіх зашифрованих текстів (u_1 , . . . , u_n).

$$(a_1, \dots, a_n) = \left(g^{r1} g_{\pi^{-1}(1)}, \dots, g^{r_n} g_{\pi^{-1}(n)}\right) \tag{2.25}$$

P потрібно довести знання π і (r_1 , . . . , r_n). P отримав випадкові числа (e_1 , . . . , e_n) з V.

2.P обчислює публічну сумму.

$$\text{a} = \prod_i^n a_i^{e_i} = Com\left(\acute{e}, \sum_i^n e_i\, r_i\right) \text{ для } \acute{e} = \left(e_{\pi(1)}, \dots, e_{\pi(n)}\right) \tag{2.26}$$

Використовує хеш-функцію H, P виконує ∑-протокол для підтведження знання $\acute{e}$ та $\sum_i^n e_i\, r_i$ виконуючи наступні кроки.

3.P вибирає випадкові задачі (w_1 , ... , w_n , w_{n+1}).

4.P обчислює $t = g^{w_n+1} g_1^{w_1} \dots g_n^{w_n}$.

5.P обчислює c = H(a, t).

6.P обчислює s = (s_1 , ... , s_n , s_{n+1}) таким чином, що

$$s = \left(w_1 + c \cdot e_1, \dots, w_n + c \cdot e_n, w_{n+1} + c \sum_i^n e_i\, r_i\right) \tag{2.27}$$

7.P передає (t, s) до V .

$$(t,\, s) \rightarrow V \tag{2.28}$$

8.V обчислює c = H(a, t) і приймає доведення, якщо

$$t \cdot a^c = g^{s_{n+1}} g_1^{s_1} \dots g_n^{s_n} \quad (2.29)$$

Маючи перетасовані та зашифровані тексти ($\acute{u}_1, \dots, \acute{u}_n$), доказуючий повинен переконати будь-якого верифікатора, що перестановка π та випадкові числа ($r_1, \dots, r)_n$ були використані і не змінені супротивникома. Спочатку P обчислює значення $u = \prod_i^n u_i^{e_i}$ та $\acute{u} = \prod_i^n \acute{u}_i^{e_{\pi(i)}}$, і публікує $\acute{u} - u^{-1}$. Це публічне значення можна порівняти з публічним значенням Алгоритму 3.3 на рівність, щоб перестановка і випадкові числа узгоджуються із зобов'язанням. Далі P виконує ∑-протокол для доведення знання π і ($r_1, \dots, r_n$). Використовуючи евристику Фіата-Шаміра [28], сильна хеш-функція замінює базову структуру зобов'язання-виклик-відповідь інтерактивного доведення з нульовим знанням. Це працює тому, що хеш містить інформацію про перетасовані повторні шифрування. Формально, хеш c містить інформацію про $\acute{u} = \pi(u \cdot E_{pk}(1, \sum_i^n e_i r_i))$[32].

Використовуючи доведення перестановок з нульовим знанням, запропоноване Вікстремом [32], пересновщик прив'язаний до певної перестановки π та набору випадкових чисел ($r_1, \dots, r_n$), які використовуються для повторного шифрування вхідної партії шифротекстів (зобов'язання Педерсена є обчислювально-зобов'язуючими [32]).

Алгоритм 6 – доведення правильності перемішування [32]:

Розглянемо перестановщика P та верифікатор V. P отримав від V випадкові числа ($e_1, \dots, e_n$). ElGamal шифротексти ($u_1, \dots, u_n$) та перешифровки ($\acute{u}_1, \dots, \acute{u}_n$). P хоче довести з нульовими знаннями, що він знає перестановку π і випадкові числа (r_1 0 , ... , r_n) такі, що $\acute{u}_\iota = u_{\pi(i)} \cdot E_{pk}(1, r_{\pi(i)})$, де E_{pk} - функція шифрування ElGamal з відкритим ключем pk.

1.P обчислює u та u^r такі, що

$$u = \prod_i^n u_i^{e_i} \text{ та } \acute{u} = \prod_i^n \acute{u}_i^{e_{\pi(i)}} \quad (2.30)$$

Значення $\acute{u} \cdot u^{-1}$ є публічними.

2.*P* вибирає випадкове завдання *w* і обчислює

$$t = E_{pk}(1, w) \tag{2.31}$$

3.Використовуючи публічну хеш-функцію H, *P* обчислює

$$c = \mathrm{H}(\acute{u} \cdot u^{-1}, t) \tag{2.32}$$

4.*P* обчислює

$$s = w + c \cdot \sum_{i}^{n} e_i r_i \tag{2.33}$$

5.*P* відправляє (*t, s*) до *V* .

$$(t, s) \rightarrow V \tag{2.34}$$

6.*V* обчислює $c = \mathrm{H}(\acute{u} \cdot u^{-1}, t)$ і приймає доведення, якщо

$$t \cdot (\acute{u} \cdot u^{-1})^c = E_{pk}(1, s) \tag{2.35}$$

Неінтерактивне доведення з нульовим знанням правильності перемішування було представлено в [32] та [33]. Обидва доведення були детально протестовані у [32, 33, 30], і відмінності між ними були мінімальними [33]. Для нашої мікс-мережі було реалізовано узгоджене із зобов'язаннями доведення тасування за Вікстремом [32]. Він розділяє неінтерактивне доведення з нульовим знанням на офлайн і онлайн фазу, причому власне доведення і перевірка оголошуються в онлайн фазі.

2.5 Схема підпису

Пакети підписуються за допомогою порогового розширення алгоритму підпису Шнорра [36]. Пороговий варіант алгоритму, який добре працює з протоколом розподіленої генерації ключів Педерсена, був запропонований Дженнаро та ін. [24]. Пороговий алгоритм підпису Шнорра разом з DKG Педерсена виявився безпечним у моделі випадкового оракула, яка передбачає, що існує функція *H,* яка виробляє дійсно випадковий вихід на основі входу, який вона ніколи

не бачила раніше. Крім того, реалізація Verificatum від Вікстorма використовує алгоритм підпису Шнорра, щоб зробити порогову криптосистему ElGamal захищеною від активних супротивників. Це означає, що алгоритм підпису Шнорра є гарним вибором у поєднанні з нашим вибором DKG та протоколів шифрування/розшифрування.

Порогова версія схеми підпису Шнорра, запропонована в [24], працює наступним чином. Маючи набір параметрів домену Педерсена (*p, q, g)*, всі *n* учасників вже володіють секретною часткою x_i та відкритим ключем *y*. Для того, щоб підписати повідомлення *m*, учасники запускають протокол розподіленої генерації ключів Педерсена, щоб спільно згенерувати ще один секретний ключ *k* та відкрите значення $r = g^k \bmod p$. Кожен P_i обчислює виклик $c = H(m, r)$, і транслює $s_i = k_i + cx_i$. Кожен s_i тепер можна перевірити, чи виконується$g^{s_i} = r_i y_i^c$. Як і у випадку з DKG Педерсена, кожен s_i публічно обчислюється, якщо перевірка не вдається. Тоді підписом *m є кортеж* $(c, s_1 + \ldots + s_n)$. Верифікацію підпису може виконати будь-який учасник P_i , перевіривши, чи виконується $g^s = ry^c$.

Безпека схеми підпису випливає зі стійкості DKG Педерсена. Порогові підписи Шнорра також неможливо підробити за припущенням про дискретний логарифм [29]. Крім того, найкращий сценарій схеми підпису є досить ефективним, оскільки для підписання повідомлення потрібен лише один раунд комунікації. Це означає, що ця схема підпису лінійно масштабується з кількістю повідомлень, що надсилаються через систему.

2.6 Багатостороння маршрутизація

Після того, як мікс виконає своє завдання - повторне шифрування та перемішування вхідних повідомлень, інші мікси повинні мати можливість отримувати ці повідомлення через протокол маршрутизації.

Багатосторонній протокол маршрутизації, реалізований в поточній системі і описаний в [19], повністю розподіляє вибір наступного маршруту по всій мережі. Протокол обмежує можливість для підслуховувача передбачити наступний маршрут. У цьому випадку сервер не можна змусити вибрати конкретний маршрут, якому надає перевагу противник. Мікси досягають випадковості маршрутизації, використовуючи схему зобов'язань для публікації випадкового числа, яке перетворюється за допомогою хеш-функції в пункт призначення наступного міксу. Оскільки кожен мікс має брати участь у відправленні випадкового числа, випадковість наступного маршруту гарантується, якщо хоча б один з міксів є чесним. Протокол маршрутизації використовує топологію “Зірка” для структурування міксів.

Ключова ідея протоколу полягає в тому, що маршрути проходять тільки від одного рівня до іншого. Таким чином, тільки мікси в поточному шарі відповідають за змішування, повторне шифрування і обчислення доказів нульового знання, що знімає частину обчислювального навантаження, з яким повинна справлятися мікс-мережа з топологією вільних маршрутів. Щоб гарантувати коректність протоколу, в нього включено декілька перевірочних дій. Все, крім набору чисел для повторного шифрування та перестановки шаффлів, публікується на дошці оголошень. Кожен аудитор може звернутися до опублікованих записів на дошці оголошень і перевірити, чи кожне повідомлення було надіслано та отримано за правильним маршрутом. Аудитори також можуть перевірити правильність процесу змішування на основі неінтерактивного тесту з нульовим рівнем знання, який доводить, що партія була повторно зашифрована і перемішана правильно. Як пояснювалося раніше, всі мікси беруть на себе роль аудитора.

Після того, як мікс виконує дві основні задачі перетворення вхідних шифротекстів, які повторно шифруються, перемішуються і підписуються,

обчислюється неінтерактивний доказ з нульовим знанням. Потім мікс публікує вхідні і вихідні шифротексти, підписи, наступний маршрут для кожного повідомлення і доказ з нульовим знанням на дошці оголошень. Наступні мікси отримують повідомлення, і ці мікси перевіряють, чи всі опубліковані значення правильні.

Коли мікс M_{ij} хоче знайти наступний маршрут для вихідного повідомлення $ć_k$, йому спочатку потрібно попрацювати разом з усіма іншими міксами. M_{ij} надсилає кожному учаснику запит на публікацію випадкового числа. Учасники відповідають, генеруючи рівномірно випадкове число r і публікуючи SHA-256-зобов'язання на дошці оголошень. Як тільки всі учасники підтвердили свої зобов'язання за допомогою випадкових чисел, M_{ij} просить відкрити зобов'язання. Учасники відкривають r, а M_{ij} обчислює і публікує R_{ij} таким чином, що

$$R_{ij} = \sum_{n-1} r \ mod\ n \tag{2.36}$$

M_{ij} використовує функції permute та map для пошуку наступного маршруту пакету повідомлень. Функція permute переставляє індекси повідомлень, використовуючи хеш R_{ij} , і видає список переставлених індексів повідомлень за алгоритмом, описаним в [19]. У цій реалізації використовується хеш-функція SHA-256 [13]. Список переставлених індексів потім присвоюється наступному міксу за допомогою функції map на основі ємності міксів у наступному стовпчику.

Алгоритм 7 – Багатостороння маршрутизація:

Змішана мережа розміром n має топологію “Зірка”, позначену через T. $M_{ij} \in$ T значення s містить список перешифрованих шифротекстів$(ć_1, \dots ć_s)$, переставлені під перестановкою π. M_{ij} володіє парою маршрутних ключів (pk, sk_{ij}) і має доступ до дошки оголошень BB. M_{ij} бажає визначити наступні мікси $M_{a_1 j+1,\dots,} M_{a_s j+1}$ для кожного шифротексту $(ć_1, \dots ć_s)$.

1.M_{ij} підписує вхідні та вихідні пакети секретом і транслює як шифротексти, так і їхні підписи.

$$(c_1, \dots, c_s), (c\acute{}_1, \dots, c\acute{}_s) \rightarrow BB$$
$$Sign_{sk_{ij}}(c_1, \dots, c_s), Sign_{sk_{ij}}(c\acute{}_1, \dots, c\acute{}_s) \rightarrow BB \qquad (2.37)$$

2.M_{ij} надсилає запит на маршрут усім іншим міксам і чекає, поки інші мікси опублікують свої зобов'язання.

3.Отримавши повідомлення про запит маршруту, мікс реагує на нього, генеруючи випадкове число $r \in Z_p$ і транслюючи зобов'язання $C(r)$.

$$C(r) \rightarrow BB \qquad (2.38)$$

4.Як тільки всі мікси опублікували свої зобов'язання, M_{ij} просить інші мікси відкрити свої зобов'язання. M_{ij} публікує R_{ij} таким чином:

$$R_{ij} = \sum_{n-1} r \ mod \ n \qquad (2.39)$$

5.M_{ij} переставляє індекси вихідних повідомлень, щоб отримати список переставлених індексів $(c_1 *, \dots, c_s *)$ за допомогою публічної хеш-функції **H**.

$$(i_1 *, \dots, i_s *) = permute((1, \dots, s), R_{ij}, H) \qquad (2.40)$$

6.M_{ij} відображає переставлені індекси $(i_1 *, \dots, i_s *)$ у список маршрутів $(a_1, \dots, a_s)$, які відповідають пунктам призначення наступних серверів $M_{a_1 j+1}, \dots, M_{a_s j+1}$.

$$(a_1, \dots, a_s) = map(i_1 *, \dots, i_s *) \qquad (2.41)$$

Перевірка - перевірка може здійснюватися іншими міксами під час маршрутизації або після того, як повідомлення досягло вихідного мікса. Для перевірки правильності виконання протоколу маршрутизації аудитор перевіряє, чи був шифротекст c_k коректно отриманий M_a kj+1, використовуючи записи, опубліковані на дошці оголошень. Для перевірки коректності маршрутизації можуть бути використані функції permute і map. Аудитор також перевіряє підписи і

кожен з неінтерактивних доказів нульового знання для перевірки правильності перемішування і повторного шифрування.

Висновки до 2 розділу

В даному розділі були визначені методи та протоколи які будуть використані в реалізації змішаної мережі. Описані алгоритими для реалізації багатосторонніх протоколів.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ МЕРЕЖІ

3.1 Вибір мови програмування та бібліотек

Для реалізації мережі було обрано мову програмування Python 3. Python - це об'єктно-орієнтована мова програмування високого рівня загального призначення з відкритим вихідним кодом. Її перевагами перед іншими мовами є:

- масштабованість
- продуктивність
- безпека
- кросс-платформеність

Були використані такі бібліотеки:

- **NumPy (Numerical Python)** — це бібліотека для наукових обчислень на Python. Вона надає підтримку для великих багатовимірних масивів і матриць, а також велику колекцію високорівневих математичних функцій для роботи з цими масивами. NumPy є основою для багатьох інших бібліотек для наукових обчислень, таких як SciPy та pandas [40].
- **SimPy (Simulation in Python)** — це бібліотека для моделювання подій. Вона дозволяє створювати дискретно-подійні симуляції, які корисні для моделювання та аналізу складних систем, таких як системи черг, мережі та інші подібні процеси[41].
- **Math** — це стандартний модуль Python для математичних операцій. Включає в себе численні математичні функції, такі як тригонометричні функції, логарифми, факторіали тощо.
- **Pandas** — це бібліотека для обробки та аналізу даних. Вона надає структуру даних і функції для роботи з даними, особливо для маніпуляції з таблицями (DataFrame) і часовими рядами. Pandas широко використовується в аналізі даних, наукових дослідженнях і фінансових додатках [5].

3.2 Тестування мережі

Мета тестування - виміряти якість мікс-мережі на основі продуктивності, масштабованості та відмовостійкості. Перша серія експериментів буде тестувати розподілену генерацію ключів, розподілене дешифрування та доказ нульового знання з точки зору продуктивності та масштабованості. Ці протоколи є відмовостійкими за своєю конструкцією. Продуктивність вимірюється часом, який потрібен всім учасникам для завершення роботи протоколу. Масштабованість розподіленої генерації ключів Педерсена вимірюється часовою складністю для різних конфігурацій учасників і розмірів ключів. Масштабованість порогового дешифрування ElGamal буде протестовано на основі кількості повідомлень, що надсилаються через мережу, та кількості серверів у системі. Неінтерактивний доказ нульового знання буде протестовано на основі кількості перетасованих повідомлень.

Тестування проводится на одному комп’ютері, процссор 6 ядер 3 ГГц 32 ГБ ОЗП з ОС Windows. Усі мікси симульовані за допомогою бібліотеки simpy.

3.2.1 Розподілена генерація ключів

Протокол розподіленої генерації ключів Педерсена було реалізовано для мікс-мережі на основі оригінальної статті Торбена Педерсена [22] та реалізацій в [27, 25]. Загальна ідея протоколу полягає в тому, щоб перекласти відповідальність за генерацію пари ключів з довіреної сторони на всіх учасників, які бажають взяти участь у дешифруванні. Метою цих експериментів є вимірювання продуктивності та масштабованості протоколу в контексті мікс-мережі.

Тести вимірюють масштабованість протоколу на основі різних змінних: кількості міксів і довжини ключового біта. Оскільки першим

завданням міксів є спільне створення пари ключів, протокол можна тестувати ізольовано, без запуску інших протоколів під час тестування.

Тестування зі змінною кількістю міксів. У цьому тесті ми вимірюємо, як кількість міксів протоколу впливає на продуктивність системи. Хоча теоретична здійсненність протоколів багатосторонніх обчислень була доведена, ці протоколи було дуже важко використовувати на практиці. Вузьким місцем залишається потреба в комунікації та розрив у продуктивності між безпекою та практичною реалізацією [37]. Результати тестування наведені в таблиці 3.1.

Тестова конфігурація:

- 6 ядерний коп'ютер з 32 ГБ Оперативної пам'яті
- ОС Windows
- Довжина ключа 1024 біт
- Топологія "Зірка"
- Кількість міксів 9 – 200

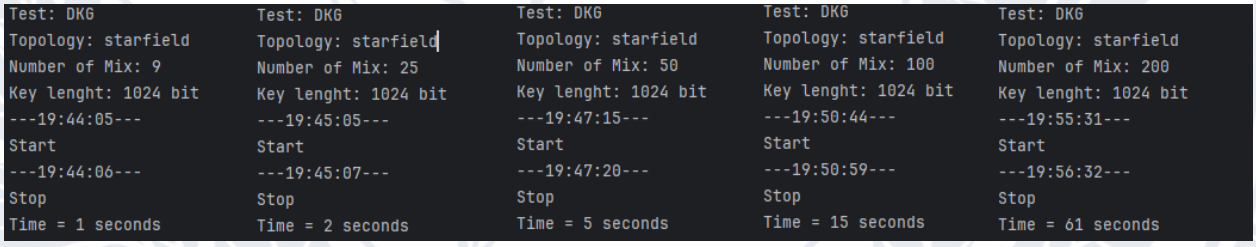

Рисунок 3.1 – Результати тестування зі змінною кількістю учасників

Таблиця 3.1 – Час затрачений на генерацію ключів залежно від кількості учасників

Кількісь учасників	Час(с)
9	1
25	2
50	5
100	15
200	61

Тести з протоколом Педерсена показують, що при постійному розмірі ключа 1024 біт, продуктивність протоколу сильно залежить від кількості міксів. Час затрачений на генерацію ключів при 9-100 міксах зростає пропорційно, однак якщо міксів більше часова складність значно зростає. Це не дивно, враховуючи кількість комунікацій, необхідних для кожного з учасників. Протокол Педерсена все ще залишається одним з найпростіших і практично здійсненних протоколів, який використовується в багатьох інших реалізаціях.

Тестування зі змінною довжиною ключа. Довжина біт криптографічного ключа b використовується для генерації параметрів домену Педерсена. Ці параметри визначають скінченне поле Z_p простого порядку p з b бітами. Таким чином, довжина бітів ключа впливає на швидкість генерації випадкових чисел під час роботи протоколу, а також на навантаження на пам'ять самої системи. Більша довжина біта ускладнює зловмиснику підбір секрету грубою силою, але це також створює обчислювальні обмеження та обмеження пам'яті для системи. Результати тестування наведені в Таблиці 3.2.

Тестова конфігурація:

- 6 ядерний коп’ютер з 32 ГБ Оперативної пам’яті
- ОС Windows
- Довжина ключа 512-8192 біт
- Топологія “Зірка”
- Кількість міксів 100

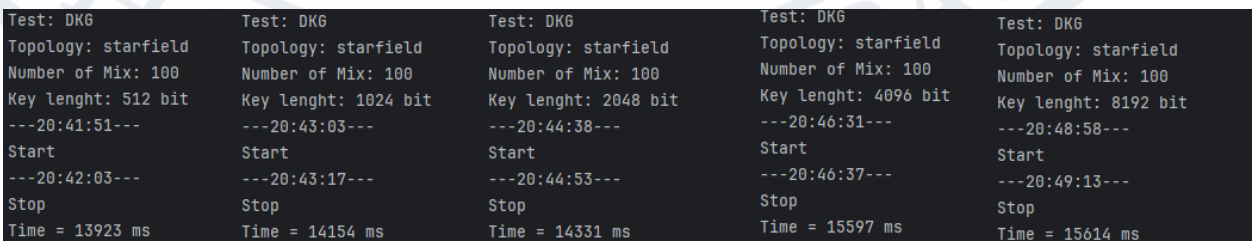

Рисунок 3.2 – Результати тестування з зміною довжини ключа

Таблиця 3.2 – Час генерації ключів в залежності від довжини

Розмір ключа (біт)	Час (мс)
512	13923
1024	14154
2048	14331
4096	15597
8192	15614

Довжина ключа майже не впливає на час генерації.

3.2.2 Криптосистема Threshhold ElGamal

Реалізація криптосистеми ElGamal базується на оригінальній статті Тахера Ель-Гамаля [25] та наступних публікаціях про порогове розширення криптосистеми з розподіленим дешифруванням, викладених в [16, 24, 20]. Ідея, що лежить в основі Протокол дешифрування полягає в тому, що набір секретних часток можна відновити за допомогою інтерполяції Лагранжа. Крім того, жодна з часток не передається в ефір.

Через комунікаційну складність протоколу, цікаво виміряти продуктивність та масштабованість дешифрувальної частини протоколу. Розшифровка - єдина частина протоколу, яка вимагає раундів зв'язку, тому експерименти проводилися тільки з розподіленою частиною розшифровки. Під час кожного раунду дешифрування учасник, який розшифровує, повинен запросити у всіх інших учасників частку дешифрування, яка є експонентою частини зашифрованого тексту і секретної частки учасника. Якщо учасник був дискваліфікований під час протоколу DKG, його секретна частка дорівнює 0. Як тільки сторона, що розшифровує, отримала частки від усіх учасників, вона може відновити відкритий текст за допомогою інтерполяції Лагранжа.

Експерименти з розподіленим протоколом дешифрування проводилися для сторони, що дешифрує, яка відновлює зашифрований текст по одному разу. До уваги бралися: кількість повідомлень, які потрібно відновити стороні, що розшифровує, і розмір мережі.

Тестування зі змінною кількістю повідомлень. Сервер дешифрування відновлює по одному зашифрованому тексту за раз, і за результатами експерименту ми бачимо, що кожна з реконструкцій займає приблизно однакову кількість часу. Хоча розшифровка великої кількості повідомлень займає багато часу, дешифрувальник буде готовий за скінченний час. Результати наведені в таблиці 3.3.

Тестова конфігурація:

- 6 ядерний коп’ютер з 32 ГБ Оперативної пам’яті
- ОС Windows
- Довжина ключа 2048 біт
- Топологія “Зірка”
- Кількість міксів 9
- Кількість повідомлень 100-1500

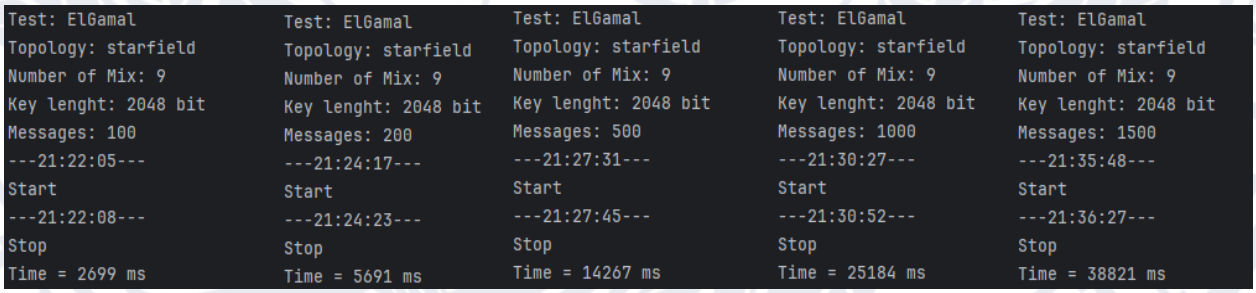

Рисунок 3.3 – Результати тесту зі змною кількості поведомлень

Таблиця 3.3 – Час дешифрування в залежності від кількості повідомленнь.

Кількість повідомленнь	Час (мс)
100	2699
200	5691
500	14267

Продовження таблиці 3.3

1000	25184
1500	38821

З цього тесту можна зробити висновок, що складність протоколу є лінійною по відношенню до кількості повідомлень, які потрібно розшифрувати.

Тестування зі змінним розміром мережі. Результати наведені в таблиці 3.4.

Тестова конфігурація:

- 6 ядерний коп'ютер з 32 ГБ Оперативної пам'яті
- ОС Windows
- Довжина ключа 2048 біт
- Топологія "Зірка"
- Кількість міксів 9-50
- Кількість повідомлень 100

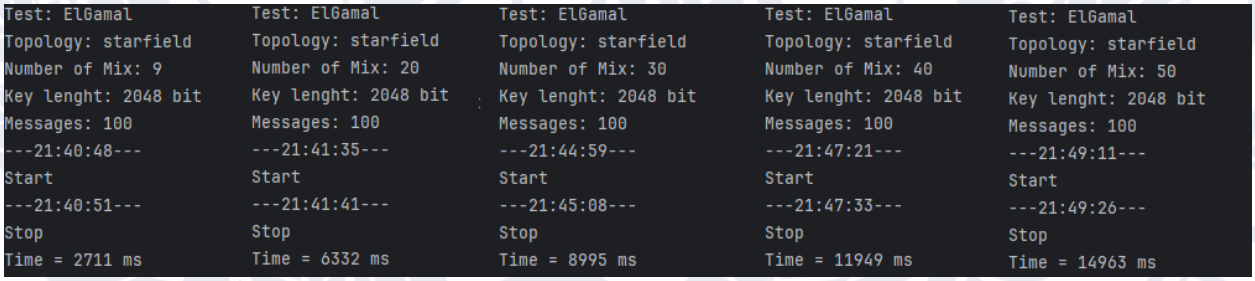

Рисунок 3.4 – Результати тестування зі змінною кількістю міксів

Таблиця 3.4 – Час дешифрування в залежності від розміру мережі

Кількість міксів	Час(мс)
9	2711
20	6332
30	8995
40	11949
50	14963

Часова складність протоколу лінійно залежить від розміру мережі.

3.2.3 Топологія ‘Зірка”

Метою цих експериментів є вимірювання затримки системи та вимірювання стійкості системи, тобто наскільки добре система працює під час (часткового) збою. З метою підвищення продуктивності, кількість міксів *M* у кожному стовпчику буде перевірено до і після атаки. З метою підвищення анонімності враховується кількість шарів *L*. Таким чином, налаштування міксів будуть позначатися як *M/L*. На рисунку 3.5 показано, як ці налаштування структурують мережу міксів.

Кількість шарів у мережі має бути щонайменше *log*(*N*), де *N* – кількість повідомлень, що надсилаються через мережу [33]. Кількість повідомлень може досягати 100 000, тому для всіх змішаних мереж буде використовуватися щонайменше 5 шарів. Вимірювані затримки - це мінімальні затримки системи, засновані на першому повідомленні, яке виходить.

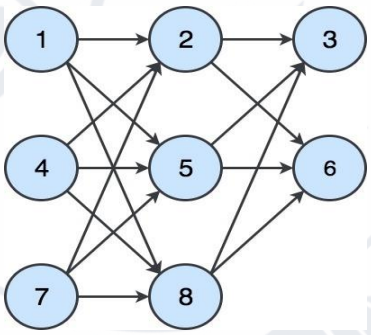

Рисунок 3.5 - Структура мережі змішування 8 міксами та 3 шарами.

На рисунку 3.6 показано результати порівняння протоколу багатосторонньої маршрутизації з використанням топології ‘Зірка” та топології з вільним маршрутом. Топологія з вільним маршрутом вимагає обчислення набагато більшої кількості доказів нульового знання під час кожного переходу, що робить її повільнішою, ніж топологія ‘Зірка”, а також не дуже добре масштабується при збільшенні кількості міксів у системі.

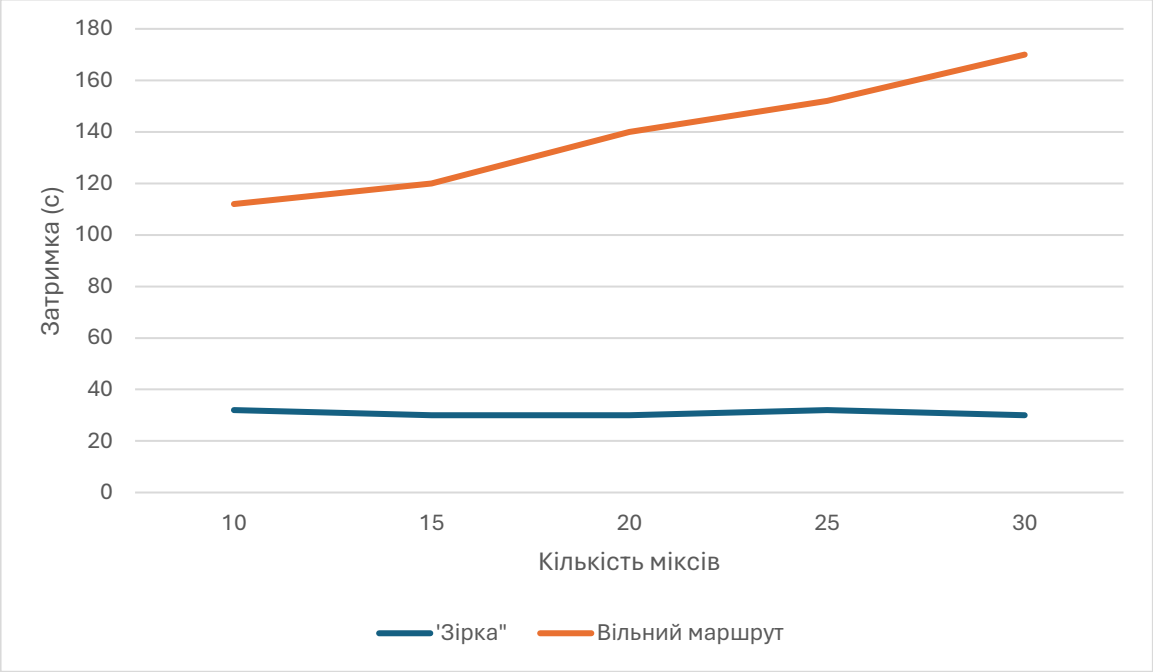

Рисунок 3.6 Тест затримки в мережі з вільним маршрутом та топологією “Зірка”

Продуктивність топології з вільним маршрутом гірша, ніж у топології “Зірка”.

3.3 Оптимізація мережі

Хоча анонімність є найважливішим аспектом змішаної мережі, вона також повинна утримувати затримку в розумних межах. Тести проведені в цій роботі, підтверджують, що більший ступінь безпеки у вигляді більших криптографічних ключів, кількості серверів і доказів з нульовим знанням обернено пропорційний продуктивності мережі. Однак, згідно з рекомендаціями NIST, розмір криптографічного ключа повинен бути не менше 1024 біт, а краще 2048 біт [13], що означає, що системи повинні бути в змозі впоратися з підвищеною затримкою, яку тягне за собою більша довжина ключа. Експерименти показують, що протокол розподіленої генерації ключів Педерсена бажано масштабувати навіть для 2048 біт. Завдяки своїй простоті, він є корисним протоколом для реалізації змішаних мереж, а також може бути використаний для

генерації ключів підпису.

Можливою оптимізацією продуктивності є спроба виконати більшу частину обчислень в автономному режимі. Для цього зручна топологія ‘Зірка”, оскільки мікси, як правило, мають виконувати менше обчислень, ніж, наприклад, мікси з вільним маршрутом. Отже, мікси у топології ‘Зірка” мають більше часу простою, який можна витратити на обчислення в автономному режимі. Наприклад, параметри домену Педерсена можуть бути згенеровані одним із міксів, який має вільну пам'ять, і опубліковані на дошці оголошень. Також можна заздалегідь обчислити значення перевірки Педерсена, секретні частки та ключ підпису. В такому випадку складність протоколу може бути зведена до кількості комунікацій, необхідних для верифікації та обміну секретами.

Поряд з продуктивністю, важливим показником якості є мереж є масштабованість. Мірою масштабованості, запропонованою в [20], є горизонтальна масштабованість. Мережа може масштабуватися горизонтально, якщо додавання нових серверів до мережі збільшує загальну пропускну здатність системи [20]. На противагу горизонтальному масштабуванню існує вертикальне масштабування, яке означає, що мережа може масштабуватися лише за рахунок збільшення потужності існуючих серверів. Багато систем анонімного зв'язку не допускають горизонтального масштабування, і такі системи, як мікс-каскад, страждають від проблем масштабованості через цей факт [20, 19].

На відміну від каскаду міксів, топологія ‘Зірка” підтримує горизонтальне масштабування [19]. Розширення системи є масштабованим, але воно також розподіляє ту саму кількість повідомлень між більшою кількістю міксів. Проблема, з якою ми стикаємося в цьому випадку, полягає в тому, що мікс тепер повинен працювати з меншими наборами анонімності. Рішенням є збільшення кількості повідомлень, що надсилаються через систему, але це зазвичай

збільшує загальну затримку системи. Однак той факт, що топологія ‘Зірка” є горизонтально масштабованою, а також її здатність збільшувати кількість повідомлень без підвищення продуктивності дозволяє надсилати більші набори анонімності через більші мережі, тим самим підвищуючи анонімність системи.

Інші топології міксів менш здатні на це. Хоча каскадним мережам потрібно обчислювати лише один доказ нульового знання на кожному кроці маршрутизації, каскади не допускають горизонтального масштабування, що робить їх дуже неефективними. З іншого боку, мережі міксів з вільним маршрутом легше масштабуються, ніж каскад міксів, але не є ефективними через необхідність великої кількості обчислень нульового знання та зв'язку між усіма іншими міксами в системі [13]. Топологія ‘Зірка” забезпечує баланс між топологією вільного маршруту та каскадом міксів. З одного боку, вона є більш масштабованою, ніж каскадний мікс, а з іншого боку, досягає кращої продуктивності, ніж топологія з вільними маршрутами.

Висновки до розділу 3

В цьому розділі було проведено тестування продуктивності мережі в різних умовах. Проведені тести з різними розмірами ключа, кількістю міксів та надісланих повідомлень. Тестування показало високі показники продуктивносі та масштабованості системи. Визначено оптимальну конфігурацію. Оптимальним рішенням буде використання топології “Зікра” що суттєво знизисть затримки в мережі та використання ключа довжиною 2048 біт згідно з рекомендаціями NIST[13], що не вплине на продуктивність мережі.

ВИСНОВКИ

У цій роботі запропонована реалізація мікс-мережі на основі багатосторонніх криптографічних алгоритмів з акцентом на пошук балансу між анонімністю, масштабованістю та продуктивністю.

Змішана мережа була реалізована з вбудованими гарантіями цілісності маршрутизації, яка не потребує довіреної третьої сторони, піддається перевірці та здатна відновлюватися після збоїв.

Тести проведені в цій роботі показали, що поєднання багатосторонніх протоколів і топології “Зірка” пропонує компроміс між масштабованістю, анонімністю та продуктивністю.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. Лмешко А.В. , Новіченко Є.О. та Недавній А.В. Безпека даних в Україні за допомогою використання технології VPN. ITSynergy. 2 (Груд 2022), 28–42. DOI:https://doi.org/10.53920/ITS-2022-2-3.
2. Карабан Д. Р., Жаровський Р. О. "Методи забезпечення анонімності в інтернеті." Матеріали XI науково-технічної конференції „Інформаційні моделі, системи та технології “ (2023): 237-237.
3. Каланча А. А., В. А. Світличний. "Засоби забезпечення анонімності в мережі." (2021).
4. Карабан Д. Р., Жаровський Р.О. "Аналіз проблем забезпечення анонімності користувачів при використанні мережі інтернет." Матеріали XII Міжнародної науково-практичної конференції молодих учених та студентів „Актуальні задачі сучасних технологій “ (2023): 456-456.
5. Карабін Р. М., Литвиненко Я. В. Огляд бібліотек машинного навчання для мови Python //Матеріали VIII науково-технічної конференції „Інформаційні моделі, системи та технології “. – 2020. – С. 165-165.
6. AXEL EKDAHL LÍDIA NYMAN “A Methodology to Validate Compliance to the GDPR” (2018): 17-19.
7. George Danezis and Claudia Diaz. “A Survey of Anonymous Communication Channels”. In: Computer Communications 33 (2010).
8. Fatemeh Shirazi et al. “A Survey on Routing in Anonymous Communication Protocols”. In: CoRR abs/1608.05538 (2016).
9. Krishna Sampigethaya. “A Survey on Mix Networks and Their Secure Applications”. In: Proceedings of the IEEE 94.12 (2006).
10. Andreas Pfitzmann and Marit Hansen. A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management. v0.34. Aug. 2010.
11. George Danezis, Roger Dingledine, and Nick Mathewson. “Mixminion: Design

of a Type III Anonymous Remailer Protocol". In: 2003 IEEE Symposium on Security and Privacy (S&P 2003), 11-14 May 2003, Berkeley, CA, USA. 2003, pp. 2–15

12. Roger Dingledine, Nick Mathewson, and Paul F. Syverson. "Tor: The Second Generation Onion Router". In: Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA. 2004, pp. 303–320.
13. Elaine Barker. NIST Special Publication 800-57: Recommendation for Key Management - Part 1: General(Revised). Tech. rep. 2016.
14. David Chaum. "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms". In: Commun. ACM 24.2 (1981), pp. 84–88.
15. George Danezis. Designing and attacking anonymous communication systems. Tech. rep. University of Cambridge, Computer Laboratory, 2004.
16. George Danezis and Claudia Diaz. "A Survey of Anonymous Communication Channels". In: Computer Communications 33 (2010).
17. Ulf M¨oller et al. "Mixmaster Protocol - Version 2". In: IETF Internet Draft (2005).
18. Andreas Pfitzmann, Birgit Pfitzmann, and Michael Waidner. "ISDN-MIXes: Untraceable Communication with Very Small Bandwidth Overhead". In: In Proceedings of the GI/ITG Conference on Communication in Distributed Systems. Springer-Verlag, 1991, pp. 451–463
19. Fatemeh Shirazi et al. "Multiparty Routing: Secure Routing for Mix Networks". In: ttps://arxiv.org/submit/1975326 (2017).
20. Albert Kwon et al. "Atom: Scalable Anonymity Resistant to Traffic Analysis". In: CoRR abs/1612.07841 (2016).
21. Paul Feldman. "A Practical Scheme for Non-interactive Verifiable Secret Sharing". In: Proceedings of the 28th Annual Symposium on Foundations of Computer Science. SFCS '87. Washington, DC, USA: IEEE Computer Society,

1987, pp. 427–438. isbn: 0-8186-0807-2.

22. Torben Pryds Pedersen. “A Threshold Cryptosystem Without a Trusted Party”. In: Proceedings of the 10th Annual International Conference on Theory and Application of Cryptographic Techniques. EUROCRYPT’91. Brighton, UK: Springer-Verlag, 1991, pp. 522–526. isbn: 3-540-54620-0.

23. Adi Shamir. “How to Share a Secret”. In: Commun. ACM 22.11 (Nov. 1979), pp. 612–613. issn: 0001-0782.

24. Felix Brandt. “Efficient Cryptographic Protocol Design Based on Distributed El Gamal Encryption”. In: Information Security and Cryptology - ICISC 2005, 8th International Conference, Seoul, Korea, December 1-2, 2005, Revised Selected Papers. 2005, pp. 32–47.

25. Veronique Cortier et al. “Distributed ElGamal `a La Pedersen: Application to Helios”. In: Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society. WPES ’13. Berlin, Germany: ACM, 2013, pp. 131–142. isbn: 978-1-4503-2485-4

26. Aniket Kate, Yizhou Huang, and Ian Goldberg. “Distributed Key Generation in the Wild”. In: IACR Cryptology ePrint Archive 2012 (2012).

27. Rosario Gennaro et al. “Secure Distributed Key Generation for DiscreteLog Based Cryptosystems”. In: Advances in Cryptology - EUROCRYPT ’99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding. 1999, pp. 295– 310

28. Paul Feldman. “A Practical Scheme for Non-interactive Verifiable Secret Sharing”. In: Proceedings of the 28th Annual Symposium on Foundations of Computer Science. SFCS ’87. Washington, DC, USA: IEEE Computer Society, 1987, pp. 427–438. isbn: 0-8186-0807-2.

29. Rosario Gennaro et al. “Secure Distributed Key Generation for Discrete-Log Based Cryptosystems”. In: J. Cryptol. 20.1 (Jan. 2007), pp. 51–83. issn: 0933-

2790.

30. Taher El Gamal. “A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms”. In: Proceedings of CRYPTO 84 on Advances in Cryptology. Santa Barbara, California, USA: Springer-Verlag New York, Inc., 1985, pp. 10–18.

31. Felix Brandt. “Efficient Cryptographic Protocol Design Based on Distributed El Gamal Encryption”. In: Information Security and Cryptology - ICISC 2005, 8th International Conference, Seoul, Korea, December 1-2, 2005, Revised Selected Papers. 2005, pp. 32–47.

32. Douglas Wikstr¨om. “A Commitment-Consistent Proof of a Shuffle”. In: Information Security and Privacy, 14th Australasian Conference, ACISP 2009, Brisbane, Australia, July 1-3, 2009, Proceedings. 2009, pp. 407–421.

33. Stephanie Bayer and Jens Groth. “Efficient Zero-Knowledge Argument for Correctness of a Shuffle”. In: Advances in Cryptology – EUROCRYPT 2012: 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings. Ed. by David Pointcheval and Thomas Johansson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 263–280.

34. Nigel Smart. Cryptography: An Introduction. McGraw-Hill, 2003.

35. Philipp Locher and Rolf Haenni. “A Lightweight Implementation of a Shuffle Proof for Electronic Voting Systems”. In: 44. Jahrestagung der Gesellschaft f¨ur Informatik, Informatik 2014, Big Data - Komplexit¨at meistern, 22.-26. September 2014 in Stuttgart, Deutschland. 2014, pp. 1391–1400.

36. Claus-Peter Schnorr. “Efficient Identification and Signatures for Smart Cards”. In: Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology. CRYPTO ’89. London, UK, UK: Springer-Verlag, 1990, pp. 239–252. isbn: 3-540-97317-6.

37. Yehuda Lindell and Benny Pinkas. “Privacy Preserving Data Mining”. In:

Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology. CRYPTO '00. London, UK, UK: Springer-Verlag, 2000, pp. 36–54.

38. Rosario Gennaro et al. Revisiting the Distributed Key Generation for Discrete-Log Based Cryptosystems. 2003.

39. Fatemeh Shirazi et al. "Multiparty Routing: Secure Routing for Mix Networks". In: ttps://arxiv.org/submit/1975326 (2017).

40. Офіційний сайт NumPY. URL: https://numpy.org/ (Дата доступу 01.06.2024)

41. Офіційний сайт SimPY. URL: Офіційний сайт NumPY. URL: https://numpy.org/ (Дата доступу 01.06.2024)

Додаток А

```
import pandas as pd
import simpy
import numpy as np
import math
import sympy
import hashlib
from sympy import mod_inverse
from flask import Flask, request, jsonify
from flask_socketio import SocketIO, emit

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

# Ініціалізація параметрів
def initialize_params():
    p = generate_large_prime()  # Велике просте число
    q = (p - 1) // 2  # q теж просте число
    g = find_primitive_root(p)
    return p, q, g

# Генерація великого простого числа
def generate_large_prime():
    return sympy.nextprime(np.random.randint(2**10, 2**11))

# Пошук первісного кореня
def find_primitive_root(p):
    for g in range(2, p):
        if pow(g, (p-1)//2, p) != 1:
```

```
        return g

# Distributed Key Generation (DKG) Педерсена
def dkg_pedersen(p, q, g):
    private_shares = [np.random.randint(1, q) for _ in range(num_servers)]
    public_shares = [pow(g, share, p) for share in private_shares]
    return private_shares, public_shares

# Threshold ElGamal Encryption
def elgamal_encrypt(p, g, public_key, message):
    k = np.random.randint(1, p-1)
    c1 = pow(g, k, p)
    c2 = (message * pow(public_key, k, p)) % p
    return c1, c2

def elgamal_decrypt(p, c1, c2, private_share):
    shared_secret = pow(c1, private_share, p)
    shared_secret_inv = mod_inverse(shared_secret, p)
    message = (c2 * shared_secret_inv) % p
    return message

# Підпис Шнорра
def schnorr_sign(p, q, g, private_key, message):
    k = np.random.randint(1, q)
    r = pow(g, k, p)
    e = hash_func(message, r) % q
    s = (k - private_key * e) % q
    return r, s

def schnorr_verify(p, q, g, public_key, message, signature):
```

```
    r, s = signature
    e = hash_func(message, r) % q
    v1 = pow(g, s, p) * pow(public_key, e, p) % p
    return v1 == r

# Хеш-функція
def hash_func(message, r):
    return int.from_bytes(hashlib.sha256((str(message) +
str(r)).encode()).digest(), byteorder='big')

# Симуляція мережі з використанням SimPy
def mixnet_simulation(env, num_servers, messages):
    p, q, g = initialize_params()
    private_shares, public_shares = dkg_pedersen(p, q, g)
    public_key = np.prod(public_shares) % p

    # Симуляція передачі повідомлень через мікснет
    for i in range(num_servers):
        env.process(mix_server(env, i, messages[i], private_shares[i],
public_key, p, g))

def mix_server(env, server_id, message, private_share, public_key, p, g):
    while True:
        yield env.timeout(np.random.exponential(1))
        c1, c2 = elgamal_encrypt(p, g, public_key, message)
        decrypted_message = elgamal_decrypt(p, c1, c2, private_share)
        print(f"Server {server_id} processed message: {decrypted_message}")
        socketio.emit('message_processed', {'server_id': server_id, 'message':
decrypted_message})
```

```
# Запуск
if __name__ == "__main__":
    num_servers = 5
    messages = [np.random.randint(1, 100) for _ in range(num_servers)]
    env = simpy.Environment()
    mixnet_simulation(env, num_servers, messages)

    @app.route('/send_message', methods=['POST'])
    def send_message():
        message = request.json.get('message')
        messages.append(int(message))
        return jsonify({'status': 'Message received', 'message': message})

    @socketio.on('connect')
    def handle_connect():
        print('Client connected')

    @socketio.on('disconnect')
    def handle_disconnect():
        print('Client disconnected')

    @socketio.on('send_message')
    def handle_send_message(data):
        message = int(data['message'])
        messages.append(message)
        env.process(mix_server(env, len(messages) - 1, message,
private_shares[len(messages) - 1], public_key, p, g))

    env.run(until=100)
    socketio.run(app, host='0.0.0.0', port=5000)
```