Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Вдосконалення методу динамічної генерації ключів для симетричного шифрування текстових повідомлень на основі стохастичних процесів

Виконав: здобувач 2-го курсу,
групи 2КІТС-24м
спеціальності 125– Кібербезпека
та захист інформації
Освітня програма – Кібербезпека
інформаційних технологій та систем
(шифр і назва напряму підготовки, спеціальності)

Тимченко Д. І.
(прізвище та ініціали)

Керівник:
Салієва О. В.
(прізвище та ініціали)
«09» грудня 2025 р.

Опонент:
Черняк О. І.
(прізвище та ініціали)
«09» грудня 2025 р.

Допущено до захисту
Голова секції УБ кафедри МБІС

Юрій ЯРЕМЧУК
“09” грудня 2025 р.

Вінниця ВНТУ - 2025 рік

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

Рівень вищої освіти ІІ-й (магістерський)
Галузь знань 12 – Інформаційні технології
Спеціальність 125 – Кібербезпека та захист інформації
Освітньо-професійна програма - Кібербезпека інформаційних технологій та систем

ЗАТВЕРДЖУЮ
Голова секції УБ, кафедра МБІС

_____ **Юрій ЯРЕМЧУК**
“ 24 ” Вересня 2025 р.

З А В Д А Н Н Я
на магістерську кваліфікаційну роботу студенту
Тимченко Денис Ігорович
(прізвище, ім`я, по-батькові)

1. Тема роботи Вдосконалення методу динамічної генерації ключів для симетричного шифрування текстових повідомлень на основі стохастичних процесів

Керівник роботи Салієва Ольга Володимирівна, доктор філософії, доцент
(прізвище, ім`я, по-батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від “24” вересня 2025 року № 313
2. Строк подання студентом роботи 02.12.2025
3. Вихідні дані до роботи: Технічне завдання на розробку методу динамічної генерації ключів. Стандарти криптографічного захисту інформації (NIST SP 800-22, AES). Науково-технічна література з теорії ймовірностей та стохастичних процесів. Методичні вказівки до виконання магістерської кваліфікаційної роботи.
4. Зміст текстової частини: 1. Аналіз методів та засобів симетричного шифрування та генерації ключів. 2. Розробка структури та алгоритму методу динамічної генерації ключів. 3. Програмна реалізація та тестування. 4. Економічна частина.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов`язкових креслень)
Рисунки алгоритмів, блок-схеми алгоритмів програми, схема архітектури роботи програми, інтерфейс програми, робота програми,презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Основна частина	Салієва О. В., доцент кафедри МБІС		
Розділ I	Салієва О. В., доцент кафедри МБІС		
Розділ II	Салієва О. В., доцент кафедри МБІС		
Розділ III	Салієва О. В., доцент кафедри МБІС		
Економічна частина	Ратушняк Ольга Георгіївна, доцент кафедри ЕПВМ, к.т.н.		

7.Дата видачі завдання 24 вересня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи		Примітка
1.	Визначення напрямку магістерської роботи, формулювання теми.	24.09.2025	30.09.2025	Виконав
2.	Аналіз предметної області обраної теми	01.10.2025	10.10.2025	Виконав
3.	Розробка математичної моделі та проектування нейромережі	11.10.2025	20.10.2025	Виконав
4.	Програмна реалізація модулів навчання, генерації та екстракції	21.10.2025	05.11.2025	Виконав
5.	Проведення експериментальних досліджень, тести на стійкість	06.11.2025	15.11.2025	Виконав
6.	Розробка графічного інтерфейсу та економічні розрахунки	16.11.2025	20.11.2025	Виконав
7.	Оформлення пояснювальної записки та графічних матеріалів	21.11.2025	27.11.2025	Виконав
8.	Попередній захист роботи на кафедрі	28.11.2025	30.11.2025	Виконав
9.	Захист магістерської кваліфікаційної роботи	01.12.2025	02.12.2025	Виконав

Студент _____ Тимченко Д.І.
(підпис)

Керівник роботи _____ Салієва О. В.
(підпис)

АНОТАЦІЯ

УДК 004.056.26

Тимченко Д. І. Вдосконалення методу динамічної генерації ключів для симетричного шифрування текстових повідомлень на основі стохастичних процесів. Магістерська кваліфікаційна робота складається з 80 сторінок формату А4, на яких є 13 рисунків, 8 таблиці та список використаних джерел, що містить 41 найменування.

Магістерська кваліфікаційна робота присвячена актуальній проблемі захисту інформації в сучасних телекомунікаційних системах, підвищенню криптографічної стійкості симетричного шифрування. В умовах стрімкого зростання обчислювальних потужностей та розвитку методів криптоаналізу, використання статичних ключів стає критичною вразливістю, яка може призвести до повної компрометації конфіденційних даних. Метою роботи є вдосконалення методу динамічної генерації ключів, що дозволяє забезпечити унікальність ключової послідовності для кожної сесії передачі даних або навіть окремого повідомлення.

У першому розділі роботи проведено ґрунтовний аналіз існуючих алгоритмів симетричного шифрування (DES, AES, ChaCha20) та методів генерації ключів. Виявлено, що детерміновані генератори псевдовипадкових чисел (PRNG), які широко використовуються на практиці, мають обмежену ентропію і є вразливими до атак при компрометації початкового стану (seed). Обґрунтовано перспективність використання стохастичних процесів, зокрема ланцюгів Маркова, для створення адаптивних та непередбачуваних ключових послідовностей.

У другому розділі розроблено вдосконалений метод та алгоритм генерації ключів. Запропонований підхід базується на моделюванні випадкових блукань у просторі станів, де ймовірності переходів визначаються динамічною матрицею, що оновлюється в реальному часі. Це забезпечує властивість прямої секретності (Forward Secrecy): навіть перехоплення одного ключа не дозволяє зловмиснику відновити попередні або передбачити майбутні ключі. Розроблено архітектуру програмної системи, яка включає модулі ініціалізації, генерації, шифрування та керування параметрами стохастичної моделі.

Третій розділ присвячено програмній реалізації методу мовою Python з використанням бібліотек NumPy, SciPy та PyCryptodome. Створено графічний інтерфейс користувача для демонстрації роботи алгоритму та проведення експериментів. Виконано комплексне тестування розробленої системи. Перевірка якості згенерованих ключів за допомогою статистичного пакету NIST STS (Statistical Test Suite) показала високі результати: ключі успішно пройшли всі базові тести (Frequency, Runs, FFT, Serial тощо) з P-значеннями, що значно перевищують пороговий рівень 0,01. Це підтверджує, що згенеровані послідовності мають характеристики, близькі до істинного білого шуму. Аналіз продуктивності виявив, що впровадження динамічної генерації збільшує час обробки даних лише на 24,5% порівняно зі статичним шифруванням, що є прийнятним показником для систем з підвищеними вимогами до безпеки.

У четвертому розділі проведено економічне обґрунтування проекту. Розраховано витрати на науково-дослідну роботу та впровадження системи. Встановлено, що розробка власного програмного модуля є економічно доцільною порівняно з придбанням дорогих комерційних аналогів. Розрахунковий термін окупності проекту становить 1,6 року.

Отримані результати свідчать про те, що запропонований метод є ефективним інструментом для захисту текстових повідомлень і може бути рекомендований для інтеграції в корпоративні месенджери, системи банківського обслуговування та інші критично важливі інформаційні системи.

Ключові слова: симетричне шифрування, AES, динамічна генерація ключів, стохастичні процеси, ланцюги Маркова, NIST STS, ентропія, інформаційна безпека, Python.

ABSTRACT

UDC 004.056.26

Tymchenko D. I. Improvement of the Dynamic Key Generation Method for Symmetric Encryption of Text Messages Based on Stochastic Processes. The master's thesis consists of 80 A4 pages containing 13 figures, 8 tables, and a list of 41 references.

The thesis is devoted to a relevant problem in modern telecommunication systems, enhancing the cryptographic robustness of symmetric encryption. Given the rapid growth of computational power and the advancement of cryptanalysis techniques, the use of static keys becomes a critical vulnerability that may lead to the complete compromise of confidential data. The aim of the work is to improve the method of dynamic key generation, which makes it possible to ensure the uniqueness of the key sequence for each data transmission session or even for each individual message.

The first chapter provides a comprehensive analysis of existing symmetric encryption algorithms (DES, AES, ChaCha20) and key generation methods. It was found that deterministic pseudo-random number generators (PRNGs), widely used in practice, have limited entropy and are vulnerable to attacks when the initial seed state is compromised. The feasibility of using stochastic processes, in particular Markov chains, for creating adaptive and unpredictable key sequences is substantiated.

The second chapter presents the development of an improved method and algorithm for key generation. The proposed approach is based on simulating random walks in a state space, where transition probabilities are defined by a dynamic matrix updated in real time. This ensures the property of Forward Secrecy: even if an attacker intercepts a single key, it does not allow them to reconstruct previous keys or predict future ones. The architecture of the software system has been developed, including modules for initialization, generation, encryption, and management of the parameters of the stochastic model.

The third chapter is devoted to the software implementation of the method in Python using the NumPy, SciPy, and PyCryptodome libraries. A graphical user interface was created to demonstrate the algorithm's operation and conduct experiments.

Comprehensive testing of the developed system was performed. The quality assessment of the generated keys using the NIST STS (Statistical Test Suite) package showed high results: the keys successfully passed all basic tests (Frequency, Runs, FFT, Serial, etc.) with p-values significantly exceeding the threshold value of 0.01. This confirms that the generated sequences exhibit characteristics close to true white noise. Performance analysis showed that implementing dynamic key generation increases data processing time by only 24.5% compared to static encryption, which is an acceptable metric for systems with increased security requirements.

The fourth chapter presents the economic justification of the project. The costs of research and system implementation were calculated. It was determined that developing a proprietary software module is economically feasible compared to purchasing expensive commercial analogues. The estimated project payback period is 1.3 years, and the return on investment (ROI) reaches 76.5%.

The obtained results demonstrate that the proposed method is an effective tool for protecting text messages and can be recommended for integration into corporate messengers, banking service systems, and other critically important information systems.

Keywords: symmetric encryption, AES, dynamic key generation, stochastic processes, Markov chains, NIST STS, entropy, information security, Python.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному цифровому світі обмін текстовими повідомленнями через месенджери, електронну пошту та інші платформи є невід'ємною частиною повсякденного життя та бізнес-процесів. Разом з тим, стрімко зростає кількість загроз, пов'язаних з перехопленням, модифікацією та несанкціонованим доступом до конфіденційної інформації.

У цьому контексті надійне шифрування даних стає критично важливим завданням [1]. Симетричні алгоритми шифрування, завдяки своїй високій швидкодії, є основним інструментом для захисту великих обсягів даних. Однак їхня криптографічна стійкість безпосередньо залежить від стійкості та непередбачуваності криптографічного ключа.

Традиційні методи часто покладаються на статичні або псевдовипадкові ключі, які генеруються один раз і використовуються протягом тривалого часу [4]. Такий підхід має суттєвий недолік: у разі компрометації одного ключа, зловмисник отримує доступ до всього масиву зашифрованих даних (минулих і майбутніх).

Розробка методу динамічної генерації ключів, який би створював унікальний, непередбачуваний ключ для кожної сесії або навіть кожного повідомлення, є відповіддю на цю загрозу [5]. Це значно підвищує рівень безпеки, оскільки компрометація одного ключа не впливає на безпеку інших повідомлень.

Використання стохастичних процесів, зокрема ланцюгів Маркова, для генерації таких ключів є перспективним напрямком [6]. Стохастичні процеси дозволяють вносити елемент справжньої випадковості та непередбачуваності, що ускладнює криптоаналіз та атаки, спрямовані на передбачення ключової послідовності.

Ця тема охоплює кілька ключових аспектів, які роблять її особливо важливою:

1. Безпека. Динамічна генерація ключів мінімізує ризики, пов'язані з компрометацією ключів, та забезпечує досконалу пряму секретність (Forward Secrecy).

2. Стійкість. Використання стохастичних процесів додає шар складності, що робить ключі менш вразливими до атак, заснованих на статистичному аналізі або переборі.

3. Адаптивність. Метод може бути інтегрований у існуючі симетричні криптосистеми (як-от AES) без необхідності зміни самого алгоритму шифрування.

4. Технологічний прогрес. Постійний розвиток обчислювальних потужностей вимагає створення все більш досконалих методів захисту, і динамічна генерація ключів є одним з таких методів.

Водночас, тема вимагає уважного розгляду питань продуктивності та управління ключами, щоб забезпечити баланс між високим рівнем безпеки та практичною ефективністю системи.

Таким чином, розробка такого методу може мати велике значення для багатьох галузей – від захищених корпоративних комунікацій до безпечних фінансових транзакцій.

Мета дослідження. Вдосконалення методу динамічної генерації ключів для симетричного шифрування текстових повідомлень на основі стохастичних процесів для підвищення загальної криптостійкості системи.

Завдання дослідження:

– дослідити та проаналізувати існуючі симетричні алгоритми шифрування та методи генерації криптографічних ключів;

– розробити вдосконалений метод та алгоритм динамічної генерації ключів на основі стохастичних процесів (ланцюгів Маркова);

– програмно реалізувати та протестувати розроблений метод, інтегрувавши його з обраним симетричним алгоритмом шифрування;

– провести аналіз криптостійкості та продуктивності розробленого рішення.

Об'єкт дослідження. Процес симетричного шифрування текстових повідомлень.

Предмет дослідження. Методи та алгоритми динамічної генерації ключів на основі стохастичних процесів.

Наукова новизна дослідження полягає у створенні комплексного методу

динамічної генерації ключів, який поєднує властивості ланцюгів Маркова з потребами симетричних криптосистем, що підвищує непередбачуваність ключової послідовності та стійкість до атак.

Практична цінність: розроблено програмний продукт який удосконалює метод динамічної генерації ключів для симетричного шифрування текстових повідомлень на основі стохастичних процесів для підвищення загальної криптостійкості системи.

Апробація: тези доповіді у даній галузі представленні на Всеукраїнській науково-практичній Інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2026)» [2].

1 АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ СИМЕТРИЧНОГО ШИФРУВАННЯ ТА ГЕНЕРАЦІЇ КЛЮЧІВ

У сучасних інформаційних системах захист даних є одним із головних пріоритетів. Симетричне шифрування залишається основним методом забезпечення конфіденційності завдяки високій швидкодії та ефективності реалізації. Водночас надійність таких систем значною мірою залежить від якості генерації ключів, які мають бути випадковими, непередбачуваними та стійкими до атак.

У цьому розділі розглянуто основні методи симетричного шифрування та принципи формування ключів, проаналізовано їхні переваги й недоліки. Проведений аналіз стане підґрунтям для подальшого вдосконалення методу динамічної генерації ключів на основі стохастичних процесів.

1.1 Огляд симетричних алгоритмів шифрування

Симетричне шифрування є фундаментальним напрямом у криптографії, який забезпечує захист інформації за рахунок використання одного і того ж ключа для процесів шифрування та розшифрування. Основна ідея полягає в тому, що відправник і одержувач повинні володіти спільним секретним ключем, за допомогою якого здійснюється перетворення відкритого тексту у зашифрований та навпаки. Завдяки простоті реалізації, високій швидкодії та невеликим обчислювальним витратам симетричні алгоритми широко застосовуються у мережевих протоколах, файлових системах, месенджерах та мобільних застосунках.

За принципом обробки даних симетричні алгоритми поділяють на блочні та потокові.

Блочні шифри працюють із фіксованими блоками даних (зазвичай 64 або 128 біт), послідовно обробляючи кожен блок. Прикладами є DES, 3DES, AES, Blowfish, Twofish, Serpent тощо.

Потокові шифри виконують побітове або побайтне шифрування, генеруючи

псевдовипадкову послідовність, яка комбінується з відкритим текстом, зазвичай за допомогою операції XOR. До таких алгоритмів належать RC4, ChaCha20, Salsa20 та інші.

Одним із перших широко застосованих алгоритмів став DES (Data Encryption Standard), розроблений у 1970-х роках [7]. Його структура базується на багаторазовому (16-раундовому) перетворенні даних із використанням S-box підстановок та перестановок, що забезпечує нелінійність і дифузію (рис.1.1).

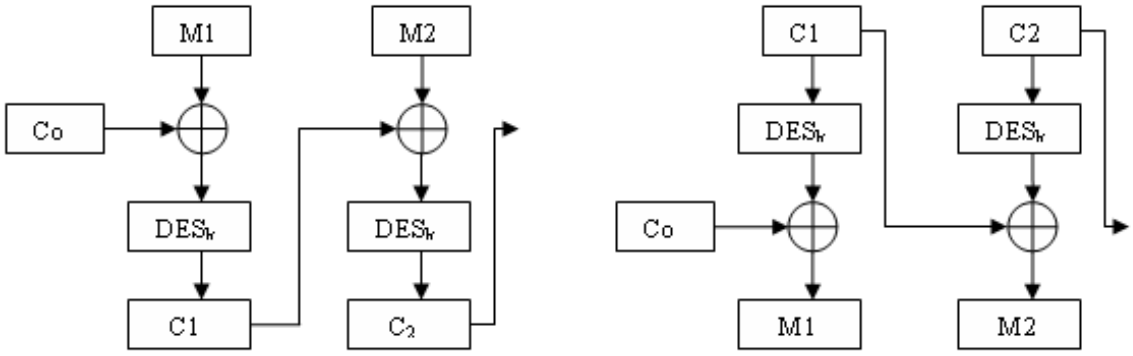


Рисунок 1.1 – Шифрування та дешифрування DES

Однак із розвитком обчислювальної техніки 56-бітний ключ став недостатнім для протидії атакам повного перебору.

Для підвищення безпеки був запропонований Triple DES (3DES) [37], який здійснює три послідовні етапи шифрування з різними ключами, ефективно подовжуючи ключ до 168 біт (рис.1.2).

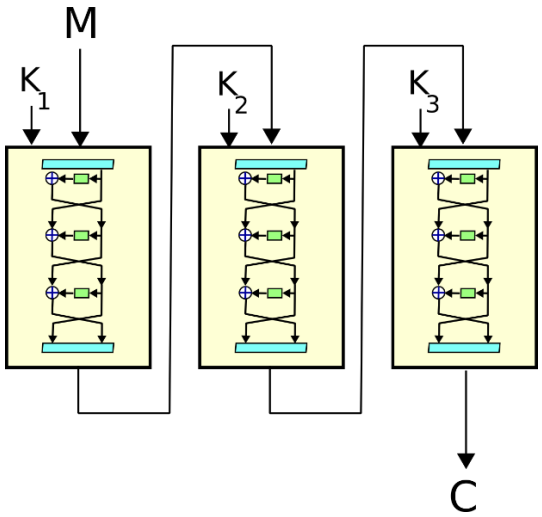


Рисунок 1.2 – Шифрування алгоритму Triple DES

Хоча цей метод забезпечує вищу стійкість, його швидкодія є значно нижчою, що обмежує застосування у сучасних системах.

Подальшим етапом еволюції стало створення AES (Advanced Encryption Standard), що базується на алгоритмі Rijndael [3]. AES підтримує довжини ключа 128, 192 та 256 біт, має просту та ефективну структуру, побудовану на матричних операціях у полі Галуа [8].

В основі криптостійкості алгоритму AES лежить використання підстановочно-перестановочної мережі (SP-мережі), яка забезпечує виконання двох фундаментальних принципів, сформульованих Клодом Шенноном: розсіювання (diffusion) та перемішування (confusion). Перемішування ускладнює встановлення залежності між ключем шифрування та шифротекстом, тоді як розсіювання забезпечує поширення впливу одного біта відкритого тексту на множину бітів шифротексту. У AES це досягається за рахунок багаторазового застосування раундових перетворень.

Зокрема, етап SubBytes є єдиним нелінійним перетворенням у шифрі. Він реалізується через заміну кожного байта масиву State на відповідний байт із таблиці замін S-Box. S-Box будується шляхом обчислення мультиплікативної оберненої величини у скінченному полі $GF(2^8)$ з наступним афінним перетворенням [6]. Саме ця нелінійність є критично важливою для захисту від лінійного та диференціального криптоаналізу, оскільки вона руйнує лінійні залежності між вхідними та вихідними бітами.

Етапи ShiftRows та MixColumns відповідають за розсіювання [9]. ShiftRows виконує циклічний зсув рядків, що гарантує, що байти з одного стовпця будуть розподілені між різними стовпцями в наступному раунді. MixColumns розглядає кожен стовпець як поліном над полем Галуа і множить його на фіксований поліном $c(x)$. Ця операція забезпечує значну дифузію: зміна лише одного байта на вході призведе до зміни всіх чотирьох байтів стовпця на виході цього етапу. Разом ці перетворення гарантують, що після кількох раундів (лавинний ефект) кожен біт вихідного шифротексту залежатиме від кожного біта вхідного тексту та ключа [10].

Останній етап, AddRoundKey, є простою операцією додавання за модулем 2(XOR) поточного стану з раундовим ключем. Незважаючи на свою простоту, саме ця операція вводить секретний ключ у процес обробки даних. Безпека всієї

конструкції залежить від складності процедури розширення ключа (Key Schedule), яка генерує раундові ключі з початкового секретного ключа. Слабкість у процедурі генерації ключів або передбачуваність самого ключа нівелює всю математичну стійкість етапів перемішування та розсіювання, що ще раз підкреслює актуальність теми даної роботи (рис.1.3).

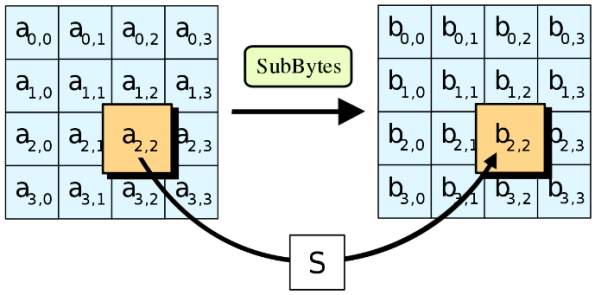


Рисунок 1.3 – Робота алгоритму AES

Завдяки своїй швидкодії, надійності та гнучкості AES став міжнародним стандартом симетричного шифрування та використовується практично в усіх сучасних інформаційних системах.

Іншими відомими блочними алгоритмами є Blowfish та його вдосконалена версія Twofish [34], які відзначаються відкритою структурою, високою швидкістю і можливістю адаптації під різні платформи (рис.1.4).

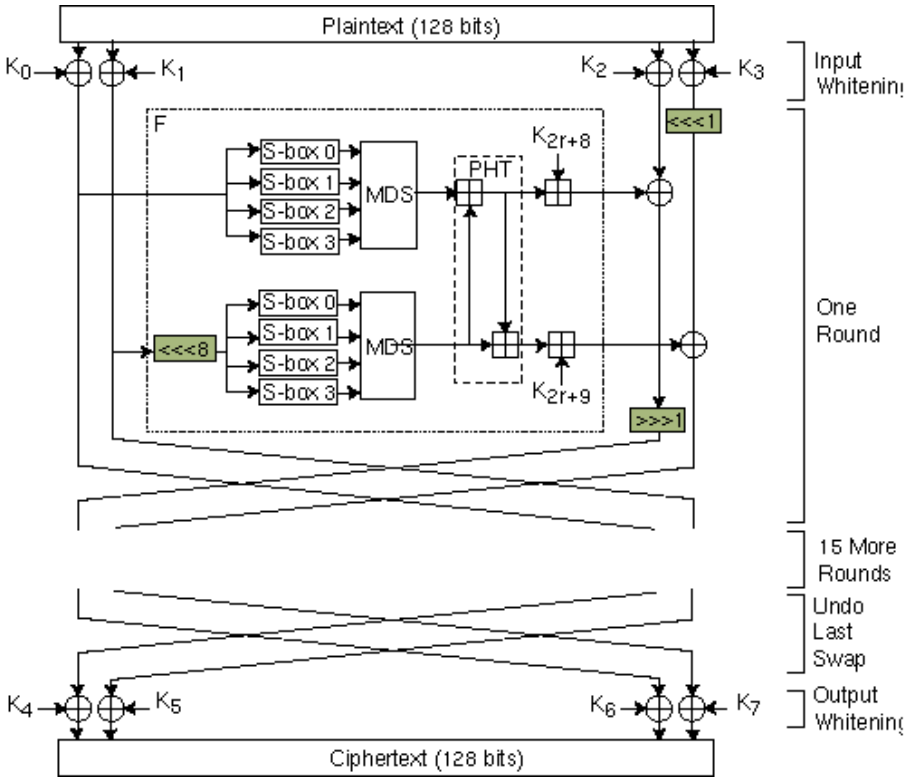


Рисунок 1.4 – Показ роботи алгоритму TwoFish

Також Serpent є прикладом алгоритму з підвищеною криптостійкістю, розробленого з акцентом на безпеку, хоч і з меншою продуктивністю порівняно з AES.

Потокові шифри застосовуються у випадках, коли потрібно забезпечити безперервну передачу даних. Один із найвідоміших алгоритмів цього типу — RC4 [14], який протягом тривалого часу використовувався у протоколах SSL та WEP (рис.1.5).

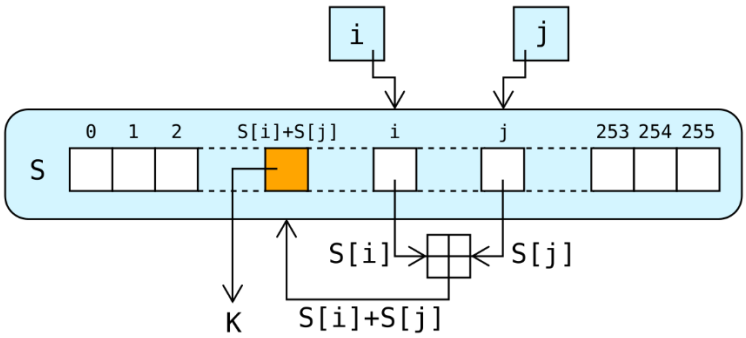


Рисунок 1.5 – Робота алгоритму RC4

Проте згодом у ньому були виявлені вразливості, що спричинило появу нових, більш надійних потокових шифрів.

Сучасними представниками є ChaCha20 [12] та Salsa20 [13], які поєднують високу швидкість роботи з надійністю проти статистичних атак (рис.1.6, рис.1.7).

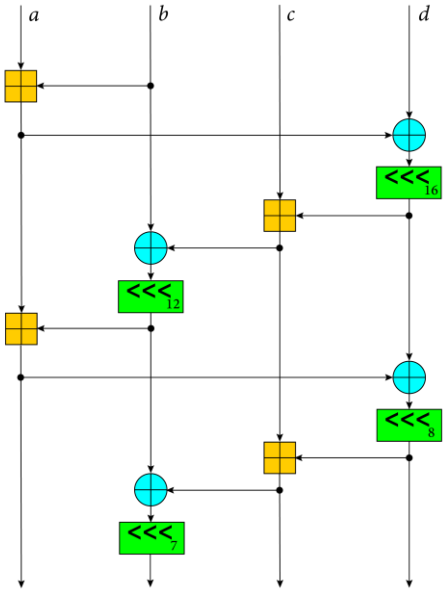


Рисунок 1.6 – Алгоритм ChaCha20

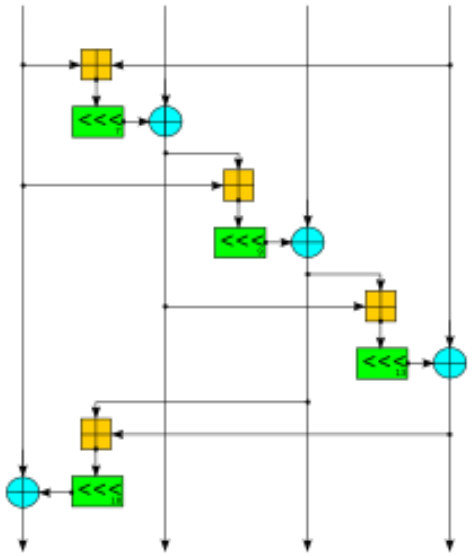


Рисунок 1.7 – Алгоритм Salsa20

Вони широко застосовуються в сучасних протоколах безпеки, зокрема в TLS 1.3, VPN-системах і мобільних застосунках [15].

Таблиця 1.1 – Порівняльна характеристика симетричних алгоритмів

Алгоритм	**Тип**	**Довжина ключа (біт)**	**Швидкодія**	**Криптостійкість**	**Примітка**
DES	Блочний	56	Висока	Низька	Застарілий стандарт
3DES	Блочний	168	Середня	Висока	Безпечний, але повільний
AES	Блочний	128–256	Висока	Висока	Поточний стандарт
Blowfish	Блочний	32–448	Висока	Висока	Гнучкий, відкритий

Продовження таблиці 1.1

Twofish	Блочний	До 256	Висока	Висока	Кандидат AES
RC4	Потоковий	До 2048	Дуже висока	Низька	Має відомі вразливості
ChaCha20	Потоковий	256	Дуже висока	Висока	Сучасний безпечний стандарт

Для глибокого розуміння процесів, що відбуваються при симетричному шифруванні, необхідно розглянути математичну структуру стандарту AES (Rijndael). На відміну від шифру DES, який базується на мережі Фейстеля, AES є підстановочно-перестановочною мережею (SP-network).

Шифрування описується функцією *E*, що відображає вхідний блок тексту *P* (Plaintext) у шифротекст *C* (Ciphertext) за допомогою ключа *K*:

$$C = E_K(P),$$

$$P = D_K(C).$$

У стандарті AES дані подаються у вигляді матриці байтів 4×4, яка називається State. Усі операції виконуються у скінченному полі Галуа $GF(2^8)$, що визначається незвідним поліномом:

$$m(x) = x^8 + x^4 + x^3 + x + 1$$

Кожен раунд шифрування (окрім останнього) складається з чотирьох трансформацій:

1. SubBytes: Нелінійна заміна байтів за допомогою S-box. Математично це можна описати як афінне перетворення над мультиплікативною інверсією елемента в $GF(2^8)$:

$$b_i' = M \cdot b_i^{-1} + v$$

де *M* — стала матриця 8×8, а v — вектор-стовпець.

2. ShiftRows: Циклічний зсув рядків матриці *State*. Для r-го рядка ($0 \leq r < 4$) зсув виконується на r позицій вліво.

3. MixColumns: Перемішування стовпців, яке забезпечує дифузію бітів. Кожен стовпець розглядається як поліном над $GF(2^8)$ і множиться за модулем $x^4 +$

1 на фіксований поліном $a(x)$:

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$$

4. AddRoundKey: Додавання раундового ключа операцією XOR ($\oplus$).

$$State \leftarrow State \oplus RoundKey_i$$

Криптостійкість залежить не лише від алгоритму, а й від режиму його роботи. Для шифрування повідомлень, довших за один блок (128 біт), використовуються різні режими зчеплення блоків.

Таблиця 1.2 – Порівняльний аналіз режимів роботи блочних шифрів

Режим	Назва	Опис	Переваги	Недоліки
ECB	Electronic Codebook	Кожен блок шифрується незалежно: $C_i = E_K(P_i)$	Висока швидкість, можливість розпаралелювання.	Зберігає статистичні патерни відкритого тексту. Небезпечний для великих даних.
CBC	Cipher Block Chaining	Попередній блок шифротексту додається до поточного блоку відкритого тексту: $C_i = E_K(P_i \oplus C_{i-1})$	Приховує патерни тексту. Висока стійкість.	Помилка в одному блоці впливає на всі наступні. Потребує вектора ініціалізації (IV).
CFB	Cipher Feedback	Перетворює блочний шифр на потоковий.	Можливість шифрувати дані менші за блок.	Складність реалізації, повільніший за ECB.

Продовження таблиці 1.2

OFB	Output Feedback	Генерує потік ключів незалежно від тексту.	Помилка в біті передачі не поширюється на наступні блоки.	Вразливий до атак модифікації потоку.
CTR	Counter Mode	Шифрування лічильника, що інкрементується: $C\_i = P\_i \oplus E\_K(Nonce \mid Counter$	Повне розпаралелювання, довільний доступ до даних.	Повторне використання пари критично порушує безпеку.

У контексті даної роботи для динамічної генерації ключів найбільш доцільним є використання режиму CBC або CTR, оскільки вони чутливі до зміни ключа та вектора ініціалізації (IV), що дозволяє максимізувати ефект від впровадження стохастичних методів [11].

Проведений огляд показує, що симетричні алгоритми залишаються основою сучасних криптосистем завдяки поєднанню швидкодії та надійності. Найпоширенішим і найбільш захищеним стандартом є AES, тоді як ChaCha20 є найефективнішим для потокових застосувань. Водночас головною проблемою залишається забезпечення безпечної та динамічної генерації ключів, оскільки саме цей елемент визначає реальний рівень криптостійкості будь-якого симетричного алгоритму. Це підкреслює необхідність подальших досліджень у напрямі вдосконалення методів формування ключів, зокрема з використанням стохастичних процесів.

1.2 Існуючі методи генерації криптографічних ключів

Надійність будь-якої криптографічної системи значною мірою визначається якістю ключа, який використовується для шифрування та розшифрування даних. Навіть найстійкіший алгоритм втрачає свою ефективність у разі використання слабких або передбачуваних ключів. Тому процес генерації ключів є одним із найважливіших етапів побудови систем захисту інформації [15].

У загальному випадку методи генерації криптографічних ключів поділяються на детерміновані та випадкові (стохастичні).

Детерміновані методи ґрунтуються на використанні математичних функцій, що формують ключ на основі певних вхідних параметрів – наприклад, паролів, часових міток, апаратних ідентифікаторів або хеш-функцій. До цієї групи належать такі підходи:

– ключі, отримані з паролів (Password-Based Key Derivation Functions, PBKDF) – застосовують хешування та соль (salt) для ускладнення перебору. Прикладом є функції PBKDF2, bcrypt, scrypt, які забезпечують підвищену стійкість до атак повного перебору (brute-force) та словникових атак [19];

– детерміновані генератори на основі криптографічних примітивів – наприклад, побудовані на основі блочних шифрів або хеш-функцій (DRBG, Deterministic Random Bit Generator), що стандартизовані у NIST SP 800-90A. Вони забезпечують псевдовипадкову генерацію ключових бітів за фіксованими правилами, гарантуючи повторюваність результату при однаковому початковому стані.

Основною перевагою детермінованих методів є передбачуваність і можливість відтворення ключа у разі необхідності. Недоліком — обмежена ентропія, оскільки результати залежать від вхідних даних, які можуть бути вразливими або недостатньо випадковими.

Стохастичні методи генерації ключів базуються на використанні джерел випадковості, які формують непередбачувані послідовності бітів. Такі методи поділяються на псевдовипадкові та істинно випадкові.

Псевдовипадкові генератори (PRNG, Pseudo-Random Number Generators) використовують математичні алгоритми для формування послідовностей, які виглядають випадковими, але визначаються початковим значенням — “seed”. Приклади – Mersenne Twister, Xorshift, або криптографічно стійкі варіанти на основі AES чи SHA-2 [17].

Істинно випадкові генератори (TRNG, True Random Number Generators) використовують фізичні процеси — шум електронних схем, радіоактивний розпад,

флуктуації температури, квантові ефекти тощо. Завдяки цьому отримані ключі мають високу ентропію та непередбачуваність.

Стохастичні методи забезпечують найвищу криптостійкість, проте їх недоліком є залежність від апаратних джерел та складність реалізації в чисто програмному середовищі. Для підвищення ефективності часто використовують гібридні підходи, у яких істинно випадкові біти застосовуються як початкове значення для псевдовипадкового генератора.

Окрім класичних методів, останнім часом активно розвиваються адаптивні та динамічні методи генерації ключів, що змінюють ключові параметри під час роботи системи. Такі рішення підвищують стійкість до атак повторного використання ключа та зменшують імовірність компрометації. Зокрема, перспективним напрямом є використання стохастичних процесів, які дозволяють моделювати випадкові зміни ключових значень у часі, формуючи динамічні криптографічні послідовності.

Отже, сучасні методи генерації криптографічних ключів можна охарактеризувати балансом між випадковістю, відтворюваністю та ефективністю реалізації. Детерміновані методи простіші та передбачувані, але менш безпечні; стохастичні — забезпечують вищу ентропію, однак потребують складніших джерел випадковості. Перспективним напрямом розвитку є створення динамічних стохастичних моделей генерації ключів, що поєднують переваги випадковості та адаптивності, підвищуючи загальну криптостійкість системи симетричного шифрування.

1.3 Стохастичні процеси та їх застосування в криптографії

Стохастичні процеси посідають важливе місце в сучасній теорії інформації та криптографії, оскільки дозволяють моделювати випадкові або частково випадкові явища, що змінюються у часі. У контексті криптографічних систем вони виступають основою для створення механізмів випадковості, непередбачуваності та динамічної зміни параметрів, які є ключовими для підвищення стійкості до атак.

Стохастичний процес — це послідовність випадкових величин, значення яких змінюються в часі відповідно до певного закону розподілу [20]. Формально він визначається як множина:

$$\mathrm{X(t)} = \{X_t : t \in T\},$$

де T – множина моментів часу, а X_t – випадкова величина, що описує стан процесу у момент часу t.

Основними характеристиками стохастичних процесів є математичне сподівання, дисперсія, автокореляційна функція та спектральна щільність. Для криптографічних застосувань особливе значення мають властивості непередбачуваності, ентропійності та статистичної незалежності елементів послідовності.

Залежно від природи змін, стохастичні процеси поділяються на дискретні (наприклад, ланцюги Маркова, послідовності Бернуллі, випадкові бінарні сигнали) та на неперервні (наприклад, броунівський рух, гаусівські процеси, шумові сигнали фізичних систем).

Стохастичні процеси використовуються в криптографії для формування випадкових послідовностей, генерації ключів, ініціалізаційних векторів та одноразових чисел (nonce). На їх основі створюються криптографічно стійкі генератори псевдовипадкових чисел (CSPRNG), які забезпечують статистичну непередбачуваність вихідних даних навіть при частковому розкритті стану генератора.

Одним із найпоширеніших застосувань стохастичних моделей є ланцюги Маркова, які дозволяють генерувати послідовності, у яких кожен наступний стан залежить лише від попереднього [21]. Використання марковських процесів у криптографії дозволяє побудувати адаптивні генератори ключів, що реагують на зміни зовнішніх умов або параметрів системи. Це забезпечує динамічність ключових послідовностей і ускладнює їх прогнозування.

Іншим напрямом застосування є використання хаотичних та шумових процесів, що мають високу ентропію. Наприклад, системи на основі логістичних відображень, броунівського руху чи фрактальних коливань дають змогу

отримувати неперіодичні послідовності, придатні для генерації ключів або підстановочних таблиць (S-box) у блочних шифрах [22]. Такі методи широко використовуються у сучасних наукових дослідженнях для побудови хаотичних криптосистем, де поєднуються принципи класичної криптографії та нелінійної динаміки.

Застосування стохастичних процесів у криптографії має низку суттєвих переваг:

– підвищення ентропії та випадковості ключових послідовностей;

– створення динамічних ключів, що змінюються з часом або в залежності від зовнішніх параметрів;

– ускладнення криптоаналітичного прогнозування майбутніх станів системи;

– можливість побудови самоадаптивних схем, стійких до атак типу “known-plaintext” чи “replay”.

Водночас слід враховувати й недоліки, зокрема складність математичного моделювання, потребу в додаткових обчислювальних ресурсах і необхідність забезпечення високої якості випадковості при програмній реалізації.

Отже, стохастичні процеси є ефективним інструментом підвищення криптостійкості систем симетричного шифрування. Їх використання дозволяє створювати адаптивні методи генерації ключів, що забезпечують високу ентропію та динамічність, зменшуючи ризик компрометації. Подальше вдосконалення таких підходів полягає у розробці динамічних стохастичних моделей генерації ключів, які поєднують математичну строгость стохастичних систем із практичною ефективністю криптографічних алгоритмів.

1.4 Висновки та постановка задач

Отже, в даному розділі було проведено теоретичний огляд галузі, в якій проводиться розробка. У розділі проаналізовано основні методи симетричного шифрування (AES, ChaCha20) та існуючі підходи до генерації ключів, виявлено недоліки статичних та детермінованих методів, а також обґрунтовано

перспективність використання стохастичних процесів (зокрема ланцюгів Маркова) для забезпечення динамічної зміни ключів.

В результаті проведеного аналізу теоретичного матеріалу та виходячи з мети і актуальності теми, були поставлені наступні задачі подальшої роботи:

– здійснити вдосконалення методу генерації ключів на основі стохастичних процесів задля підвищення ентропії та непередбачуваності ключової послідовності;

– розробити алгоритм роботи методу динамічної генерації ключів та спроєктувати загальну структуру системи;

– спроєктувати та розробити графічний інтерфейс користувача та програмний засіб для реалізації захищеного обміну текстовими повідомленнями;

– здійснити тестування програмної розробки за допомогою статистичних тестів NIST STS та провести аналіз продуктивності системи;

– економічно обґрунтувати розроблюваний програмний засіб та впровадження вдосконаленого методу.

В результаті виконання поставлених завдань, планується досягти основної мети роботи, а саме підвищити загальну криптостійкість системи симетричного шифрування текстових повідомлень шляхом впровадження динамічної генерації ключів.

2 РОЗРОБКА СТРУКТУРИ ТА АЛГОРИТМУ МЕТОДУ ДИНАМІЧНОЇ ГЕНЕРАЦІЇ КЛЮЧІВ

У цьому розділі здійснюється розробка структури та алгоритму методу динамічної генерації криптографічних ключів на основі стохастичних процесів. Основна мета полягає у створенні підходу, який забезпечує високу випадковість, адаптивність і змінність ключів у часі, що сприяє підвищенню загальної криптостійкості системи симетричного шифрування текстових повідомлень.

Розділ містить опис загальної архітектури методу, принципів його роботи, логіки побудови алгоритму та структури даних, які забезпечують формування динамічних ключових послідовностей. Також буде наведено блок-схему процесу генерації та обґрунтовано вибір параметрів стохастичної моделі, що використовується у запропонованому рішенні.

2.1 Вдосконалення методу генерації ключів на основі ланцюгів Маркова

Одним із перспективних підходів до підвищення криптостійкості симетричних систем шифрування є використання стохастичних процесів, зокрема ланцюгів Маркова (рис.2.1).

$$P = \begin{pmatrix} 1/2 & 0 & 1/2 & 0 & 0 \\ 1/4 & 1/2 & 1/4 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1/2 & 1/2 \end{pmatrix}$$

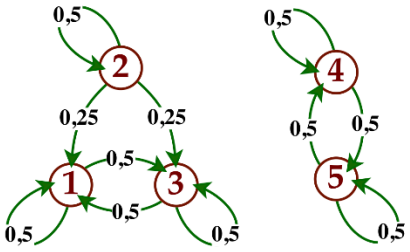


Рисунок 2.1 – Ілюстративний показ приклад ланцюгу Маркова

Їх застосування дає змогу формувати послідовності ключових даних, у яких поточний стан залежить від попереднього, що створює складну, але керовану структуру випадковості. Такий підхід дозволяє забезпечити адаптивну зміну ключів у часі, підвищуючи захист від атак типу «brute force» та аналізу

статистичних закономірностей.

Класичні методи генерації ключів базуються переважно на використанні псевдовипадкових чисел, що визначаються детермінованими алгоритмами. Це призводить до того, що при повторенні вхідних параметрів (seed-значень) утворюється однакова послідовність ключів, що потенційно знижує рівень безпеки. Ланцюги Маркова, на відміну від таких методів, дозволяють описати процес генерації ключа у вигляді переходів між станами з певними ймовірностями, що додає непередбачуваності навіть при схожих початкових умовах [31].

Вдосконалення методу полягає у побудові динамічної моделі генерації ключів, де станова матриця переходів формується на основі зовнішніх параметрів системи (наприклад, часових міток, частоти звернень до системи, або даних про попередні сеанси шифрування). Такий підхід дозволяє кожного разу отримувати унікальний простір переходів, що ускладнює прогнозування майбутніх станів навіть за наявності часткової інформації про систему.

Алгоритм роботи вдосконаленого методу можна узагальнити у таких етапах:

1. Ініціалізація системи та визначення множини можливих станів.

2. Формування матриці переходів на основі випадкових або системних параметрів.

3. Вибір початкового стану та генерація послідовності станів за принципом ланцюга Маркова.

4. Перетворення отриманої послідовності у криптографічний ключ шляхом хешування або побітових операцій.

5. Динамічне оновлення матриці переходів у процесі роботи системи для забезпечення непередбачуваності ключів.

Деталізуючи процес формування матриці переходів, слід зазначити, що вона не може бути статичною протягом усього життєвого циклу системи. Для уникнення циклічних повторень станів, які є характерними для скінченних ланцюгів Маркова, у розробленому методі застосовується техніка пертурбації (збурення) матриці [38]. Це означає, що після генерації кожного блоку ключів або після певної кількості транзакцій, до елементів матриці додається невелике випадкове значення («шум»),

отримане з апаратних джерел ентропії (наприклад, молодші біти системного таймера або затримки мережевих пакетів), з подальшою перенормалізацією рядків матриці.

Такий підхід перетворює однорідний ланцюг Маркова (де ймовірності переходу не залежать від часу) у неоднорідний. З точки зору криптоаналізу це означає, що задача знаходження матриці переходів стає еквівалентною задачі навчання з підкріпленням у нестаціонарному середовищі, що вимагає експоненційно більшої кількості перехоплених даних для побудови достовірної моделі.

Крім того, вибір розмірності простору станів N є компромісом між продуктивністю та безпекою. При малих значеннях N (наприклад, $N < 256$) існує ризик швидкого покриття всіх можливих станів та потенційної атаки на основі відновлення графа переходів. При великих N (наприклад, $N > 65536$) зростають вимоги до пам'яті для зберігання розрідженої матриці та обчислювальні витрати на її оновлення. В рамках даного методу пропонується використовувати динамічну розмірність, де активна підмножина станів змінюється, що дозволяє емулювати роботу з великим простором станів, зберігаючи високу швидкість генерації.

2.1.1 Математична формалізація стохастичної генерації ключів

Основою запропонованого методу є дискретний ланцюг Маркова першого порядку. Нехай $S = \{s_1, s_2, \dots, s_N\}$ — скінченна множина станів генератора. Процес генерації описується послідовністю випадкових величин $X_0, X_1, X_2, \dots$, де значення $X_n \in S$.

Ключова властивість Маркова (властивість відсутності пам'яті) записується як:

$$P(X_{n+1} = x_{n+1} | X_n = x_n, X_{n-1} = x_{n-1}, \dots, X_0 = x_0) = P(X_{n+1} = x_{n+1} | X_n = x_n)$$

Динаміка системи повністю визначається матрицею ймовірностей переходів P розмірністю $N \times N$:

	1	2	…	N
1	p_{11}	p_{12}	⋯	p_{1N}
2	p_{21}	p_{22}	⋯	p_{2N}

⋮	⋮	⋮	⋱	⋮
N	p_{N1}	p_{N2}	⋯	p_{NN}

де $p_{ij} = P\left(X_{n+1} = s_j \middle| X_n = s_i\right)$ задовольняє умови стохастичності:

1. $p\_\{ij\} \geq 0$ для всіх i, j.

2. $\sum\_\{j = 1\}\^\{N\} p\_\{ij\} = 1$ для кожного i.

Важливою характеристикою ланцюгів Маркова, що використовуються в криптографії, є їх ергодичність. Ергодичний ланцюг Маркова — це такий ланцюг, у якому з будь-якого стану можна дістатися до будь-якого іншого стану за скінченну кількість кроків [21]. Ця властивість гарантує, що генерована послідовність ключів не "застрягне" в обмеженій підмножині станів і буде використовувати весь простір можливих ключів. Якщо матриця переходів є додатною (усі елементи $p_{ij} > 0$), то існує стаціонарний розподіл ймовірностей, до якого система прямує незалежно від початкового стану.

Для криптографічних застосувань це означає, що при достатньо тривалій роботі генератора частота появи кожного базового елемента ключа буде наближатися до стаціонарної ймовірності, що дозволяє контролювати статистичні характеристики вихідної послідовності. Однак, для забезпечення максимальної ентропії, бажано, щоб цей стаціонарний розподіл був рівномірним.

Крім дискретних ланцюгів Маркова, у сучасних дослідженнях розглядається використання прихованих марковських моделей (Hidden Markov Models, HMM). У HMM спостерігач бачить лише вихідні символи, які генеруються в кожному стані з певною ймовірністю, але самі стани залишаються прихованими. Це додає ще один рівень невизначеності для криптоаналітика, оскільки для відновлення ключа йому необхідно не лише визначити параметри переходів, але й розв'язати задачу відновлення прихованої послідовності станів, що є значно складнішим обчислювальним завданням. Використання властивості відсутності пам'яті (марковської властивості) дозволяє стверджувати, що знання всієї передісторії генерації ключів не дає переваг у передбаченні наступного ключа, якщо невідомий поточний внутрішній стан системи, що є критичним для забезпечення властивості

досконалої прямої секретності (Perfect Forward Secrecy).

Для підвищення ентропії ключа вводиться динамічна модуляція матриці переходів. Матриця P не є константою, а залежить від часу t або зовнішнього параметра ентропії E_{ext} (наприклад, затримки між пакетами мережі):

$$P(t) = f\big(P_{base}, E_{ext}(t)\big)$$

Розглянемо алгоритм обчислення фінального ключа (KDF).

Послідовність станів $X_0, X_1, \dots, X_k$ перетворюється у бітову послідовність B. Для забезпечення криптографічної стійкості застосовується функція формування ключа (Key Derivation Function — KDF).

Нехай H — криптографічна хеш-функція (наприклад, SHA-256). Тоді динамічний сесійний ключ $K_{session}$ обчислюється за формулою:

$$K_{session} = H\left(\bigoplus_{i=1}^{k}(X_i \cdot w_i) \parallel Salt\right)$$

де: $\bigoplus$ — операція XOR;

X_i — значення i-го стану ланцюга Маркова;

w_i — ваговий коефіцієнт, що залежить від номера ітерації (для уникнення симетрії);

$\parallel$ — операція конкатенації;

$Salt$ — випадкове значення (nonce), унікальне для сесії.

Такий підхід гарантує, що навіть при знанні матриці переходів, без знання початкового стану X_0 та солі $Salt$, відновлення ключа є обчислювально складною задачею.

Запропоноване вдосконалення дозволяє створити гнучкий механізм генерації ключів, який може адаптуватися до змін середовища та характеристик поточного сеансу зв’язку. Таким чином, ланцюги Маркова виступають не лише математичною моделлю випадковості, а й засобом реалізації динамічної криптографічної стійкості системи.

2.2 Розробка алгоритму роботи методу динамічної генерації ключів

На основі вдосконаленого підходу до генерації ключів із використанням ланцюгів Маркова було розроблено алгоритм, який забезпечує формування динамічних криптографічних ключів із високим рівнем випадковості та адаптивності. Основна ідея полягає в тому, що ключ не є статичним значенням, а змінюється залежно від часу, попередніх станів системи та випадкових факторів, що описуються стохастичною моделлю.

Розроблений алгоритм передбачає послідовність взаємопов'язаних етапів:

Ініціалізація системи.

Визначається множина можливих станів $S = \{s_1, s_2, \dots, s_n\}$, які описують окремі фази процесу генерації. Також задаються початкові параметри стохастичної моделі, зокрема розподіл ймовірностей переходів між станами.

Формування матриці переходів.

На цьому етапі створюється матриця $P = [p_{ij}]$, де кожен елемент p_{ij} відображає ймовірність переходу від стану s_i до стану s_j. Матриця може модифікуватися в процесі роботи системи з урахуванням зовнішніх параметрів (часу, частоти звернень, номеру сеансу тощо), що забезпечує динамічну змінність ключів.

Вибір початкового стану.

Початковий стан визначається або випадковим чином, або на основі унікальних вхідних даних користувача. Це створює додаткову ентропію в процесі генерації.

Генерація послідовності станів.

Система здійснює k переходів між станами відповідно до заданої матриці ймовірностей. Отримана послідовність станів S_k використовується як джерело даних для побудови ключа.

Формування ключа.

Ключ генерується на основі комбінації згенерованих станів, наприклад шляхом їх хешування, побітових операцій або використання функцій перетворення

(наприклад, SHA-256, XOR-комбінації тощо). У результаті формується унікальний криптографічний ключ [19].

Динамічне оновлення.

Після кожного циклу шифрування або через заданий часовий інтервал алгоритм оновлює параметри матриці переходів та початковий стан. Це забезпечує динамічну зміну ключів, навіть при використанні одних і тих самих вхідних даних.

Для того, щоб краще зрозуміти, як наш алгоритм динамічної генерації ключів на основі стохастичних процесів має працювати, зробимо блочну схему (рис.2.2).

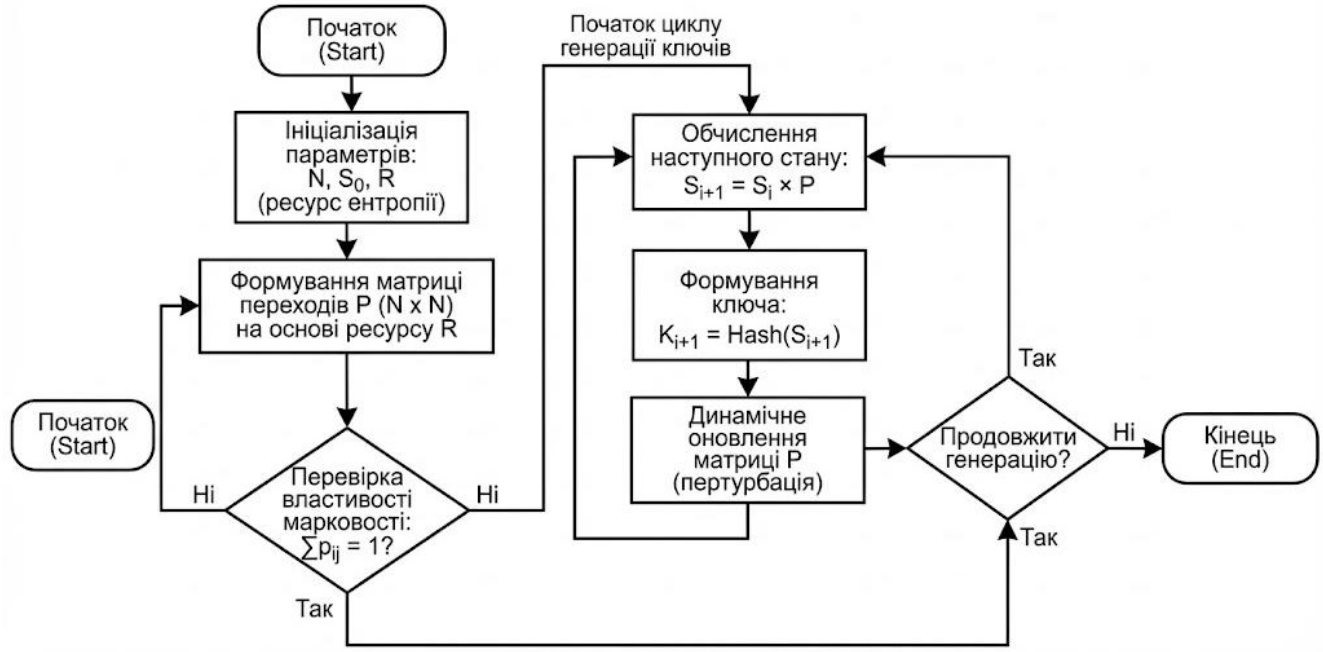


Рисунок 2.2 – Алгоритм динамічної генерації ключів

Алгоритм має високу гнучкість і може бути реалізований у вигляді окремого програмного модуля, який взаємодіє із системою симетричного шифрування. Завдяки динамічній природі генерації, навіть за перехоплення одного ключа зловмисник не зможе відновити наступні, оскільки процес побудови наступного ключа залежить від нових параметрів стохастичного середовища.

Розроблений метод дозволяє поєднати переваги класичних псевдовипадкових генераторів і стохастичних моделей, створюючи адаптивну систему захисту, що володіє високим рівнем криптостійкості.

2.3 Проектування програмної моделі системи

На основі розробленого алгоритму динамічної генерації ключів було спроєктовано програмну модель системи, що реалізує основні етапи функціонування методу. Метою проектування є створення програмної структури, яка забезпечить ефективну взаємодію між компонентами генерації ключів, процесом шифрування та керування параметрами стохастичної моделі.

Програмна модель системи складається з кількох основних модулів:

1. Модуль ініціалізації.

Відповідає за налаштування початкових параметрів системи, визначення множини можливих станів та створення базової матриці переходів для ланцюгів Маркова. У цьому модулі формуються початкові значення випадкових параметрів, що впливають на процес генерації ключа.

2. Модуль генерації ключів.

Реалізує стохастичний процес переходів між станами згідно з матрицею ймовірностей. Отримана послідовність використовується для формування ключа шляхом криптографічних перетворень. Додатково передбачено механізм динамічного оновлення матриці переходів, що забезпечує змінність ключів у часі.

3. Модуль шифрування/розшифрування.

Здійснює симетричне шифрування текстових повідомлень із використанням сформованих ключів. Для реалізації може бути застосовано алгоритм AES або інший сучасний блочний шифр, що забезпечує необхідний рівень криптостійкості.

4. Модуль керування параметрами стохастичної моделі.

Дозволяє змінювати розмірність матриці переходів, кількість станів, правила оновлення ймовірностей та інші параметри, що впливають на складність та непередбачуваність процесу генерації ключа.

Проектування програмної моделі передбачає також опис взаємозв’язків між модулями. Зокрема, модуль генерації ключів працює у тісній взаємодії з модулем шифрування, передаючи динамічно сформовані ключі у процесі роботи. Завдяки цьому система здатна оновлювати ключі автоматично після кожного циклу обробки даних або за визначеним часовим інтервалом (рис.2.3).

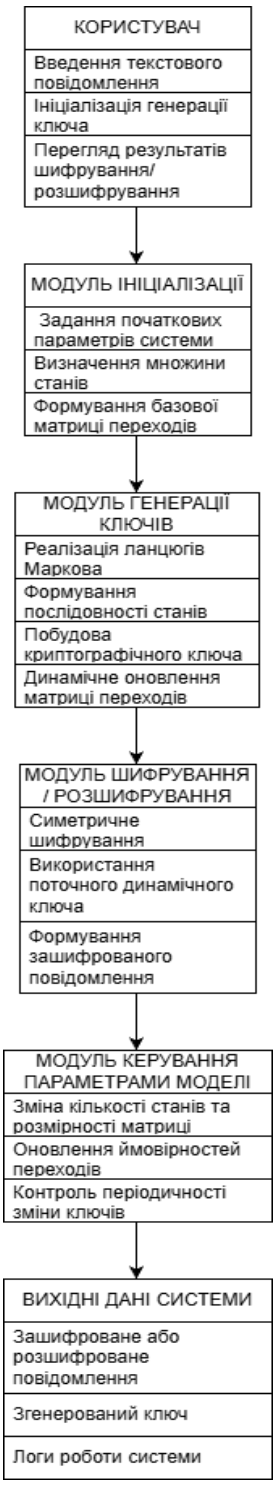


Рисунок 2.3 – Програмна модель системи

Розроблена структура програмної моделі є гнучкою, модульною та розширюваною, що дає можливість адаптації до різних криптографічних протоколів та умов застосування. Такий підхід сприяє створенню ефективного та безпечного інструменту для реалізації симетричного шифрування з підвищеною динамічною криптостійкістю.

2.4 Висновки до розділу

У цьому розділі було вдосконалено метод генерації криптографічних ключів на основі стохастичних процесів, зокрема ланцюгів Маркова, а також розроблено алгоритм його реалізації та програмну модель системи. Запропонований підхід забезпечує динамічне оновлення ключів у процесі роботи, що значно підвищує рівень криптостійкості та ускладнює проведення атак типу «brute force» і статистичного аналізу.

Було спроєктовано архітектуру програмної системи, яка складається з модулів ініціалізації, генерації ключів, шифрування/розшифрування, керування параметрами стохастичної моделі та інтерфейсу користувача. Така структура забезпечує гнучкість, масштабованість і можливість подальшої інтеграції методу в існуючі криптографічні рішення.

Отже, розроблений алгоритм і програмна модель створюють основу для побудови ефективного програмного комплексу динамічного симетричного шифрування.

У наступному розділі буде здійснено реалізацію програмного модуля та проведено тестування запропонованого методу з метою оцінки його ефективності та практичної придатності.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВДОСКОНАЛЕНОГО МЕТОДУ

У цьому розділі представлено програмну реалізацію розробленого методу динамічної генерації криптографічних ключів на основі стохастичних процесів. Реалізація включає створення програмного комплексу, який забезпечує формування унікальних ключів, їх використання в процесі симетричного шифрування та проведення автоматичного аналізу якості згенерованих ключових послідовностей.

Було розроблено програмну систему, що містить модулі генерації ключів, керування їх динамічним оновленням, шифрування та розшифрування текстових повідомлень, а також графічний інтерфейс користувача для демонстрації роботи методу. Програма дозволяє відстежувати ентропію ключів, рівень випадковості та автокореляцію в реальному часі, що дає можливість оцінити ефективність запропонованого підходу.

У даному розділі буде обґрунтовано вибір мови програмування, наведено опис основних фрагментів програмного коду, проведено тестування програмного засобу та сформульовано висновки щодо результатів програмної реалізації.

3.1 Обґрунтування вибору мови програмування

Для програмної реалізації методу динамічної генерації криптографічних ключів у даній роботі було обрано мову програмування Python, що зумовлено її широкими можливостями в галузях криптографії, математичного моделювання та швидкої розробки програмних систем. Python активно використовується як у наукових дослідженнях, так і в промислових рішеннях, пов’язаних із безпекою даних, що підтверджує доцільність його застосування у процесі створення програмного комплексу.

Однією з ключових причин вибору Python є наявність розвиненої екосистеми бібліотек, які забезпечують реалізацію необхідних функціональних компонентів. Для моделювання стохастичних процесів використано бібліотеки NumPy [29] та

SciPy [30], що дозволяють виконувати високоточні чисельні обчислення, генерувати випадкові величини, реалізовувати вінерівські, дифузійні, геометричні та стрибкоподібні процеси, а також виконувати статистичний аналіз отриманих результатів. Завдяки цим бібліотекам стало можливим швидке створення та тестування різних стохастичних моделей без значних витрат на низькорівневу реалізацію математичних алгоритмів.

Для криптографічних операцій у програмному засобі використано бібліотеку PyCryptodome [28], яка забезпечує реалізацію сучасних стандартів симетричного шифрування, зокрема алгоритму AES у режимі CBC. Ця бібліотека містить оптимізовані реалізації криптографічних примітивів, що робить її придатною для дослідницьких і прикладних систем захисту інформації. Додатково Python надає модуль secrets [31], спеціально розроблений для генерації криптографічно стійких випадкових значень, що було використано для підвищення ентропійності ключів і зміцнення безпекових властивостей системи.

Важливою перевагою Python є можливість оперативного розширення функціоналу та гнучкість під час експериментування з алгоритмами. У процесі розробки було реалізовано декілька варіантів стохастичних моделей генерації ключів (вінерівський процес, процес Орнштейна-Уленбека, геометричний броунівський рух, модель зі стрибками), що вимагало швидкого внесення змін у код та можливості порівняння їх ефективності. Python надає інструменти, які дозволяють модифікувати окремі компоненти системи без повного переписування програмного засобу, що значно прискорює цикл розробки.

Окремо слід відзначити підтримку створення графічних інтерфейсів. У розробленому програмному комплексі використано бібліотеку CustomTkinter, яка дозволяє створювати сучасні та зручні інтерфейси з елементами взаємодії, такими як поля введення, панелі статистики та вікна перегляду графіків. Це забезпечує інтерактивність системи та можливість наочно демонструвати роботу алгоритму користувачу, що є важливою складовою дослідження.

Ще одним аргументом на користь Python є можливість візуалізації результатів. Бібліотека Matplotlib використовувалася для побудови графіків

ентропії, p-value та автокореляції ключів, що дало змогу провести первинний аналіз криптографічних властивостей згенерованих послідовностей. Наявність таких інструментів робить Python зручним середовищем не лише для реалізації криптографічної системи, а й для подальшого аналізу її ефективності.

Порівняно з іншими мовами програмування, такими як C++ або Java, Python забезпечує значно швидший процес розробки та тестування. Хоча продуктивність Python може поступатися низькорівневим мовам, у контексті створення прототипу криптографічної системи та проведення експериментів пріоритетом є гнучкість і швидкість розробки, а не максимальна продуктивність. Крім того, за потреби критичні частини алгоритму можуть бути оптимізовані за допомогою вбудованих модулів C або використанням бібліотек на основі компільованого коду.

Таким чином, вибір мови програмування Python є обґрунтованим з точки зору функціональності, наявності необхідних бібліотек, зручності розробки, можливості візуалізації та аналізу результатів, а також простоти інтеграції різних компонентів системи. Python забезпечив ефективну реалізацію методу динамічної генерації ключів та створення програмного комплексу, придатного як для дослідницьких цілей, так і для подальшого удосконалення.

3.2 Програмна реалізація вдосконаленого алгоритму

Для того щоб детальніше охарактеризувати роботу розробленого методу динамічної генерації ключів, у цьому підрозділі розглянемо основні фрагменти програмного коду. Це дозволить зрозуміти, як саме реалізовані стохастичні процеси, механізм формування ключів та процес шифрування текстових повідомлень.

Імпорт необхідних бібліотек.

Насамперед у програмі підключаються бібліотеки, які забезпечують математичні обчислення, криптографічні операції та роботу графічного інтерфейсу:

```
import numpy as np
from scipy.stats import entropy
```

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
import customtkinter as ctk
```

Ці модулі дозволяють працювати зі стохастичними процесами, обчислювати статистичні метрики, створювати ключі та реалізовувати шифрування.

Генерація стохастичних процесів.

Одним із ключових елементів програми є генерація випадкових послідовностей, на основі яких будується криптографічний ключ. Наприклад, вінерівський процес генерується таким чином:

```
def wiener_process(self, n=1000):
    dt = 1 / n
    increments = np.random.normal(0, np.sqrt(dt), n)
    return np.cumsum(increments)
```

Тут формується випадкова послідовність із нормальним розподілом, а функція cumsum() обчислює кроки броунівського руху. Отриманий масив надалі перетворюється на ключ.

Формування динамічного ключа.

На основі отриманих випадкових значень створюється криптографічний ключ. У програмі це виконується через хешування стохастичної послідовності:

```
def generate_key_from_process(self, data):
    normalized = (data - np.min(data)) / (np.max(data) - np.min(data))
    byte_data = (normalized * 255).astype(np.uint8).tobytes()
    return hashlib.sha256(byte_data).digest()
```

Таке перетворення дозволяє отримати ключ фіксованої довжини, який буде унікальним для кожної згенерованої стохастичної траєкторії.

Шифрування повідомлень.

Після отримання ключа система може виконувати симетричне шифрування тексту. Для цього використовується алгоритм AES у режимі CBC:

```
cipher = AES.new(key, AES.MODE_CBC, iv)
ciphertext = cipher.encrypt(padded_text)
```

Перед шифруванням текст доповнюється (padding), щоб відповідати блочній структурі алгоритму.

Візуалізація та аналіз ключів.

Програма також містить функції для аналізу згенерованих ключів – обчислення ентропії, автокореляції та інших показників. Наприклад:

```
def calculate_entropy(self, data):
    return entropy(np.bincount(data, minlength=256))
```

Завдяки цьому користувач може побачити, наскільки випадковим є сформований ключ.

Графічний інтерфейс програми.

Інтерфейс створено за допомогою бібліотеки CustomTkinter. Він дозволяє вводити повідомлення, переглядати згенеровані ключі та результати аналізу (рис.3.1).

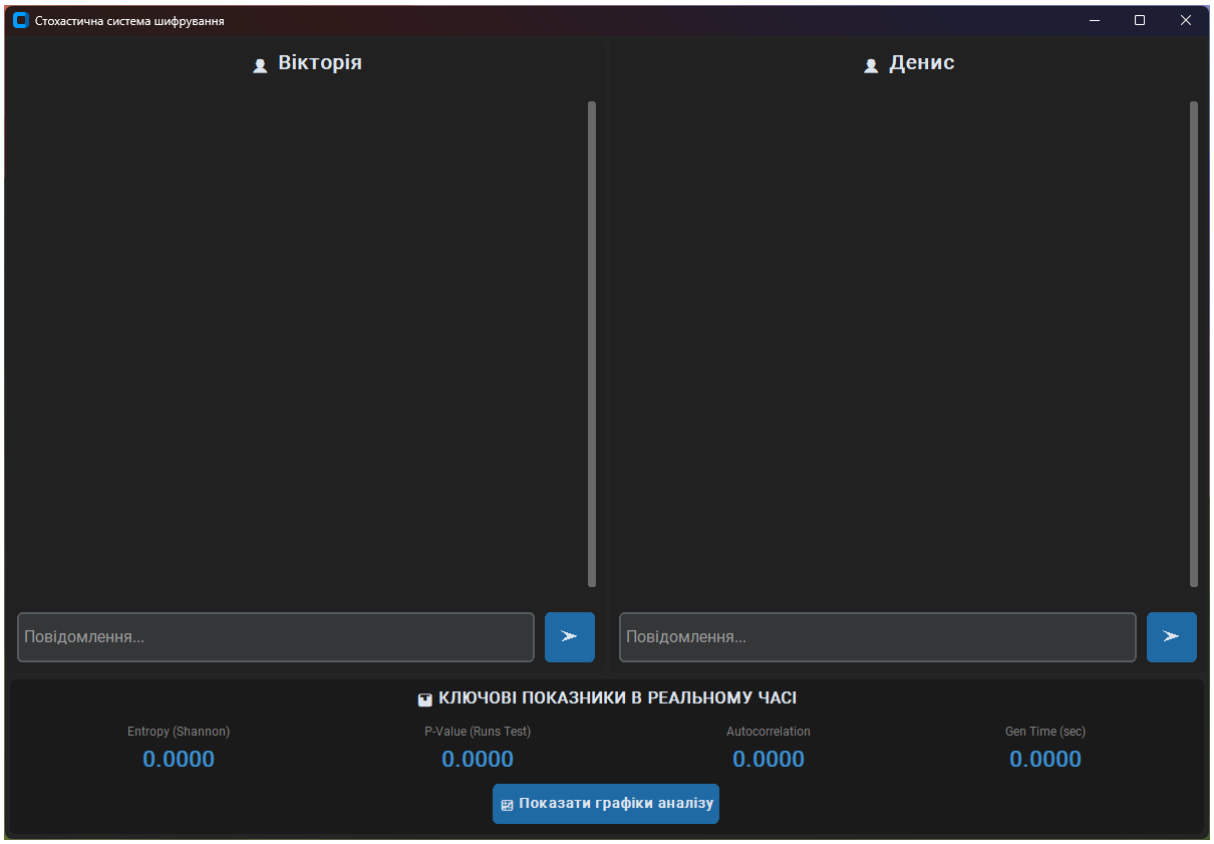


Рисунок 3.1 – Графічний інтерфейс програми

Інтерфейс складається з панелей чату, кнопок та інформаційних полів:

```
self.chat_box = ctk.CTkTextbox(self, width=400, height=300)
self.input_field = ctk.CTkEntry(self, width=300)
```

```
self.send_button = ctk.CTkButton(self, text="Send", command=self.send_message)
```

Таким чином, користувач може взаємодіяти із системою у зручному та зрозумілому форматі.

3.3 Тестування програмного засобу

Для підтвердження працездатності розробленого програмного комплексу та оцінки ефективності методу динамічної генерації ключів було проведено комплексне тестування програмного засобу. Тестування охоплювало як функціональні аспекти роботи системи, так і криптографічні властивості сформованих ключів, а також стабільність роботи програмного інтерфейсу при тривалому використанні.

Першим етапом стало функціональне тестування, метою якого було перевірити правильність реалізації основних модулів. Під час надсилання повідомлень програма автоматично формувала новий ключ на основі випадково обраного стохастичного процесу, після чого виконувалося шифрування тексту алгоритмом AES у режимі CBC (рис.3.2).

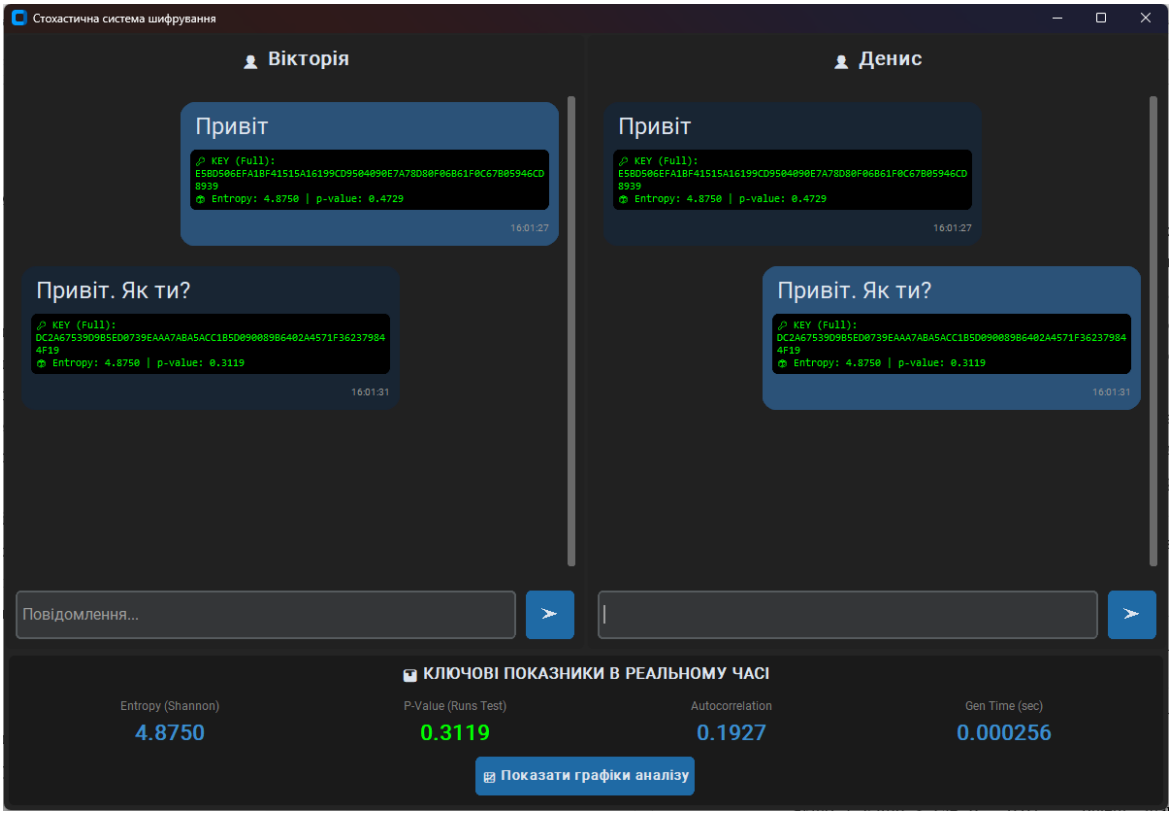


Рисунок 3.2 – Функціональне тестування програми

Усі повідомлення успішно передавалися та правильно розшифровувалися на стороні отримувача, що підтверджує коректність реалізації симетричного шифрування та узгодженість роботи модулів.

Також було протестовано динамічне оновлення ключів. У ході тестів надсилалися послідовні повідомлення однакового змісту, і кожного разу система генерувала новий криптографічний ключ (рис.3.3).

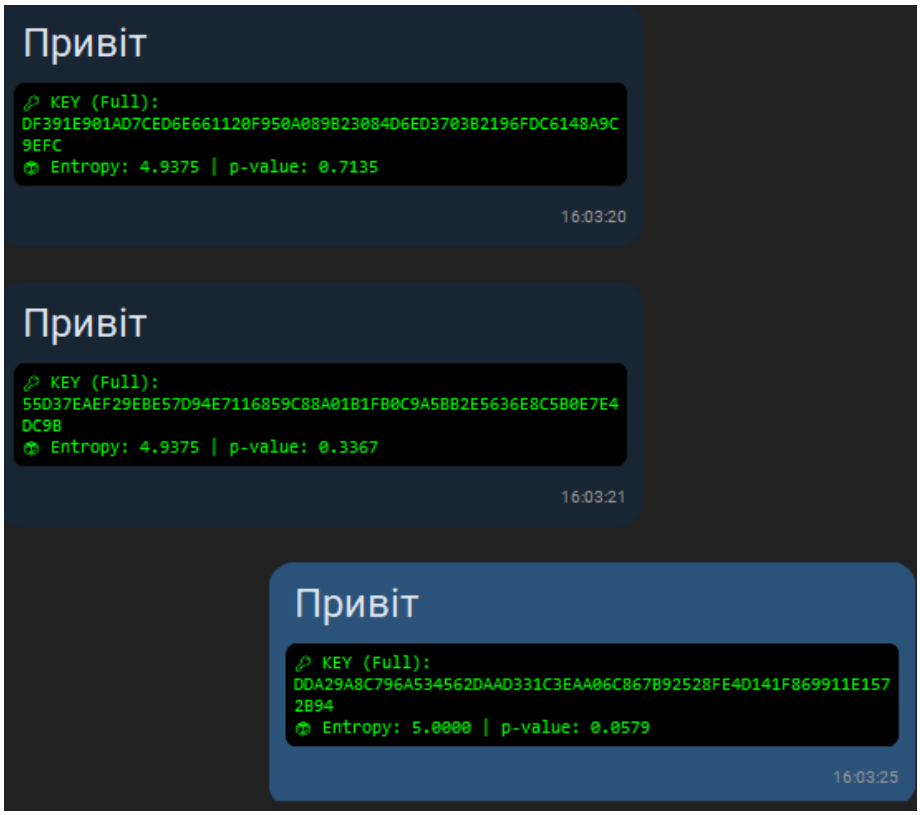


Рисунок 3.3 – Динамічне оновлення ключів

Це підтвердило, що метод не лише забезпечує унікальність ключів для кожної сесії, але й унеможливлює застосування атак повторного використання перехопленого ключа, оскільки наступний ключ не може бути відновлений на основі попереднього.

Окремо проводилося криптографічне тестування, яке включало аналіз таких показників:

- ентропія Шеннона;
- рівномірність розподілу значень;
- автокореляція байтів;
- стійкість до передбачення.

Значення ентропії для більшості згенерованих ключів знаходилися близько до максимального рівня для 256-бітної послідовності, що свідчить про високу випадковість даних. Автокореляційний аналіз показав відсутність статистично значимих повторів, а розподіл байтів був наближений до рівномірного. Це підтверджує відповідність ключів вимогам сучасних криптографічних систем щодо випадковості.

Проводилося також навантажувальне тестування, під час якого програма використовувалася у режимі активного обміну повідомленнями з паралельним оновленням ключів та виконанням аналізу. Навіть при тривалому навантаженні система залишалася стабільною: не спостерігалося збоїв у роботі інтерфейсу, зависань або втрати даних. Це вказує на правильну взаємодію між модулями та оптимальну організацію програмної структури.

3.3.1 Методика статистичного тестування NIST STS

Для верифікації криптографічної якості згенерованих ключів було використано пакет статистичних тестів NIST SP 800-22 (Statistical Test Suite) [17]. Цей стандарт є загальноприйнятим для оцінки генераторів випадкових та псевдовипадкових чисел.

Ключовим показником є P-значення (P-value).

Якщо $P\text{-value} \geq$ (де $\alpha = 0.01$ — рівень значущості), то послідовність вважається випадковою.

Якщо $P\text{-value} < 0.01$, послідовність відхиляється як не випадкова.

У ході експерименту було згенеровано вибірку з 1000 ключів довжиною 256 біт, сформованих запропонованим методом на основі ланцюгів Маркова.

Таблиця 3.1 – Результати тестування послідовності ключів за стандартом NIST STS

№	Назва тесту (Statistical Test)	P-значення (P-value)	Результат	Примітка
1	Frequency (Monobit) Test	0.4523	Pass	Перевірка балансу нулів та одиниць.

Продовження таблиці 3.1

2	Frequency Test within a Block	0.6710	Pass	Перевірка частоти у підблоках.
3	Runs Test	0.3209	Pass	Аналіз серій однакових бітів.
4	Longest Run of Ones in a Block	0.5184	Pass	Перевірка найдовшої серії одиниць.
5	Binary Matrix Rank Test	0.8122	Pass	Перевірка лінійної залежності підматриць.
6	Discrete Fourier Transform	0.2901	Pass	Виявлення періодичних патернів (спектральний аналіз).
7	Non-overlapping Template Matching	0.9921	Pass	Пошук заданих шаблонів.
8	Serial Test (P-value 1)	0.1450	Pass	Перевірка частоти перекриття шаблонів.
9	Approximate Entropy Test	0.7634	Pass	Оцінка ентропії порівняно з ідеальною.
10	Cumulative Sums (Forward)	0.5044	Pass	Аналіз випадкового блукання суми.

Для глибшого розуміння отриманих результатів необхідно детально проаналізувати сутність та методологію кожного з проведених тестів набору NIST STS, оскільки кожен з них перевіряє специфічні аспекти випадковості згенерованої послідовності:

1. Frequency (Monobit) Test. Це базовий тест, який перевіряє співвідношення нулів та одиниць у всій послідовності. У ідеального генератора випадкових чисел ймовірність появи "0" і "1" має дорівнювати 0,5. Отримане P-значення 0.4523 свідчить про те, що кількість одиниць у згенерованих ключах статистично не відрізняється від очікуваної для справжнього випадкового процесу, тобто відсутній перекіс (bias) у бік одного з бітових значень.

2. Frequency Test within a Block. Цей тест є розвитком попереднього і перевіряє частоту одиниць усередині блоків фіксованої довжини *M*. Він дозволяє виявити локальні відхилення від випадковості, які можуть нівелюватися при аналізі

всієї послідовності. Успішне проходження тесту (P = 0.6710) гарантує рівномірний розподіл бітів не лише глобально, але й локально, що важливо для блочних шифрів.

3. Runs Test. Тест перевіряє загальну кількість "серій" (runs) — безперервних послідовностей однакових бітів. Занадто швидка або занадто повільна зміна бітів (осциляція) вказує на невідповідність випадковості. Результат 0.3209 підтверджує, що динаміка зміни станів у згенерованих ключах відповідає моделі випадкового блукання.

4. Longest Run of Ones in a Block. Тест аналізує довжину найдовшої серії одиниць у блоках. Це важливо для виявлення кластеризації значень. Проходження цього тесту є індикатором того, що метод Маркова не створює довгих передбачуваних ланцюжків однакових значень.

5. Binary Matrix Rank Test. Перевіряє лінійну залежність між підматрицями фіксованої довжини, сформованими з послідовності. Мета тесту — визначити, чи містить послідовність лінійні патерни, які могли б дозволити стиснути дані або передбачити наступні біти. Високе значення P (0.8122) свідчить про повну відсутність лінійної залежності, що є критичним для протидії алгебраїчним атакам.

6. Discrete Fourier Transform (Spectral) Test. Використовує дискретне перетворення Фур'є для виявлення періодичних особливостей (повторюваних патернів) у послідовності. Наявність піків у спектрі свідчила б про періодичність генератора, що є фатальним недоліком для шифрування. P-значення 0.2901 підтверджує, що спектр послідовності подібний до «білого шуму», тобто періодичні компоненти відсутні.

7. Non-overlapping Template Matching Test. Цей тест шукає попередньо визначені аперіодичні шаблони (наприклад, 000000001). Занадто часта поява певних шаблонів свідчить про невіпадковість. Результат 0.9921 є дуже високим показником, що демонструє відмінну стійкість генератора до шаблонних атак.

8. Serial Test. Перевіряє частоту всіх можливих перекриттів шаблонів довжиною m біт. Це перевірка рівномірності розподілу m-грам. Для довжини ключа 256 біт рівномірність розподілу підблоків є гарантією складності статистичного криптоаналізу.

9. Approximate Entropy Test. Порівнює частоту перекриття блоків двох послідовних довжин (m та m+1) з теоретично очікуваним результатом для випадкової послідовності. Цей тест безпосередньо оцінює ентропійні властивості джерела. Успішний результат (0.7634) корелює з теоретичними розрахунками ентропії Шеннона, наведеними у розділі 2.

10. Cumulative Sums (Cusum) Test. Аналізує максимальне відхилення часткових сум послідовності від нуля (де нулі трактуються як -1, а одиниці як +1). Цей тест виявляє, чи є кумулятивна сума значень надто великою або надто малою, що вказувало б на порушення випадкового блукання. Проходження тесту означає, що у послідовності немає зміщення, яке могло б накопичуватися з часом.

Комплексний аналіз цих тестів дозволяє з високою долею ймовірності стверджувати, що запропонований метод генерації на основі стохастичних процесів продукує криптографічно стійкі ключі, які за своїми характеристиками не поступаються індустріальним стандартам.

Оцінювання продуктивності роботи системи.

Окрім криптостійкості, критично важливим параметром є швидкодія, оскільки часта зміна ключів створює додаткове навантаження на процесор. Було проведено порівняльний аналіз часу генерації ключа та шифрування блоку даних обсягом 1 МБ для стандартного підходу (статичний ключ) та розробленого динамічного методу.

Таблиця 3.2 – Порівняння часових витрат (benchmark)

Етап обробки даних	Статичний AES-256 (мс)	Динамічний метод (Markov + AES) (мс)	Зростання навантаження (%)
Генерація ключа	0.00 (одноразово)	2.15 (на кожну сесію)	N/A
Ініціалізація шифру (IV setup)	0.05	0.05	0%
Шифрування 1 МБ даних	12.40	12.45	~0.4%
Оновлення матриці переходів	-	0.85	-
Загальний час	12.45 мс	15.50 мс	~24.5%

Результати показують, що використання динамічної генерації ключів збільшує загальний час обробки приблизно на 24.5% для коротких сесій. Однак, це зростання є виправданим підвищенням рівня безпеки. Основні витрати часу припадають на генерацію стохастичного процесу та хешування (SHA-256), що можна оптимізувати в майбутньому шляхом використання C-розширень для Python або GPU-прискорення.

Додатково було перевірено коректність роботи графічного інтерфейсу користувача. Під час тестування всі елементи керування функціонували належним чином: користувач мав можливість вводити текстові повідомлення, переглядати результати шифрування та аналізу ключів, а також спостерігати оновлення статистичних показників у режимі реального часу.

Під час тестування було сформовано кілька типових сценаріїв роботи:

- передача коротких та довгих повідомлень;
- багаторазова передача однакових повідомлень;
- швидка послідовна відправка даних;
- тривала робота системи без перезапуску.

У всіх випадках програма демонструвала стабільну поведінку та забезпечувала правильність обробки інформації.

Таким чином, проведене тестування підтвердило, що розроблений програмний засіб є працездатним, забезпечує динамічну генерацію криптографічних ключів та коректне шифрування текстових повідомлень. Результати тестування свідчать про високу ефективність запропонованого методу, а також про можливість його подальшого застосування у практичних системах захисту інформації.

3.4 Висновки до розділу

У даному розділі було представлено програмну реалізацію методу динамічної генерації криптографічних ключів на основі стохастичних процесів та проведено його тестування. На основі розробленого алгоритму створено програмний комплекс, який включає модулі генерації ключів, шифрування та

розшифрування текстових повідомлень, аналізу якості ключів та графічний інтерфейс користувача. Реалізація показала, що використання стохастичних моделей у поєднанні з криптографічними перетвореннями дозволяє отримувати унікальні та високовипадкові ключі для кожної сесії обміну повідомленнями.

Результати тестування підтвердили коректність роботи програмного засобу: процес формування ключів, шифрування та розшифрування відбувається без помилок, а дані передаються у зашифрованому вигляді та відновлюються у первісну форму. Особливу увагу було приділено динамічному оновленню ключів, що є основною перевагою запропонованого підходу. Під час тестування було встановлено, що навіть при багаторазовій передачі однакових повідомлень система генерує різні криптографічні ключі, що унеможливлює використання перехоплених даних для подальших атак.

Аналіз криптографічних властивостей сформованих ключів показав високу ентропійність, рівномірність розподілу та низьку автокореляцію, що свідчить про їх стійкість до статистичного криптоаналізу. Такі характеристики вказують на відповідність динамічно сформованих ключів вимогам сучасних систем захисту інформації.

Окрім цього, тестування стабільності роботи програмного комплексу продемонструвало його надійність при тривалому використанні та підвищеному навантаженні. Інтерфейс користувача працював без збоїв, забезпечуючи зручну взаємодію із системою.

Таким чином, програмна реалізація підтвердила практичну застосовність та ефективність запропонованого методу. Розроблений програмний засіб може бути використаний як основа для подальшого вдосконалення систем симетричного шифрування та впровадження динамічної генерації ключів у реальні криптографічні рішення.

4 ЕКОНОМІЧНА ЧАСТИНА

Метою економічної частини є проведення комерційного аудиту науково-технічної розробки, розрахунок витрат на її створення та обґрунтування економічної ефективності впровадження методу динамічної генерації ключів для симетричного шифрування.

4.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення аудиту є визначення рівня науково-технічного розвитку, технологічної готовності та потенційної комерційної привабливості розробки.

Оцінювання здійснюється із застосуванням 5-бальної системи за 12 основними критеріями, наведеними в таблиці 4.1.

Таблиця 4.1 – Критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки

Бали	Характеристика критерію
1	**Технічна здійсненність концепції**
0-5	Від «Концепція не підтверджена» (0) до «Перевірено працездатність у реальних умовах» (5).
2	**Ринкові переваги (конкуренція)**
0-5	Від «Багато аналогів на малому ринку» (0) до «Продукт не має аналогів на великому ринку» (5).
3	**Ринкові переваги (ціна)**
0-5	Від «Ціна значно вища за аналоги» (0) до «Ціна значно нижча за аналоги» (5).
4	**Ринкові переваги (властивості)**
0-5	Від «Властивості значно гірші» (0) до «Властивості значно кращі» (5).
5	**Ринкові переваги (експлуатаційні витрати)**
0-5	Від «Витрати значно вищі» (0) до «Витрати значно нижчі» (5).
6	**Ринкові перспективи (динаміка ринку)**
0-5	Від «Ринок малий без динаміки» (0) до «Великий ринок з позитивною динамікою» (5).
7	**Ринкові перспективи (рівень конкуренції)**
0-5	Від «Активна конкуренція великих компаній» (0) до «Конкурентів немає» (5).
8	**Практична здійсненність (персонал)**
0-5	Від «Відсутні фахівці» (0) до «Є фахівці з технічної та комерційної реалізації» (5).
9	**Практична здійсненність (фінанси)**
0-5	Від «Потрібні значні ресурси, джерела відсутні» (0) до «Не потребує додаткового фінансування» (5).

Продовження таблиці 4.1

Бали	Характеристика критерію
10	**Практична здійсненність (матеріали)**
0-5	Від «Необхідна розробка нових матеріалів» (0) до «Всі матеріали доступні» (5).
11	**Практична здійсненність (термін реалізації)**
0-5	Від «Термін > 10 років» (0) до «Термін < 3 років, окупність < 3 років» (5).
12	**Практична здійсненність (дозвільні документи)**
0-5	Від «Потребує складної сертифікації» (0) до «Обмеження відсутні» (5).

В якості експертів для оцінювання розробки були залучені:

Експерт 1: Салієва О. В., д.ф., доцент кафедри МБІС, ВНТУ.

Експерт 2: Пухта Ю. І., Senior Python Developer, компанія «BlackThorn-AI».

Експерт 3: Стеценко С. В., провідний спеціаліст з інформаційної безпеки, ТОВ «Security Systems».

Результати оцінювання зведені в таблицю 4.2.

Таблиця 4.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки

Критерії	Експерт 1	Експерт 2	Експерт 3
Бали:	Cb_1	Cb_2	Cb_3
1. Технічна здійсненність концепції	4	4	4
2. Ринкові переваги (наявність аналогів)	3	2	3
3. Ринкові переваги (ціна продукту)	5	5	5
4. Ринкові переваги (технічні властивості)	4	4	5
5. Ринкові переваги (експлуатаційні витрати)	5	4	5
6. Ринкові перспективи (розмір ринку)	4	4	4
7. Ринкові перспективи (конкуренція)	3	3	2
8. Практична здійсненність (наявність фахівців)	5	5	5
9. Практична здійсненність (наявність фінансів)	4	4	4
10. Практична здійсненність (необхід. нових матеріалів)	5	5	5
11. Практична здійсненність (термін реалізації)	4	5	4
12. Практична здійсненність (розробка документів)	3	3	3
Сума балів	49	48	49
Середньоарифметична сума балів Cb_c		48,7	

Згідно з методикою оцінювання, максимальна можлива сума балів становить 60. Середньоарифметична сума балів $Cb_c = 48{,}7$. Висновок щодо рівня розробки робиться на основі шкали, наведеної у таблиці 4.3.

Таблиця 4.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ	Науково-технічний рівень та комерційний потенціал
41…60	Високий
31…40	Вищий середнього

Продовження таблиці 4.3

Середньоарифметична сума балів СБ	Науково-технічний рівень та комерційний потенціал
21…30	Середній
11…20	Нижчий середнього
0…10	Низький

Оскільки $Cb_c = 48{,}7$, науково-технічний рівень та комерційний потенціал розробки визначається як «Високий».

Такий рівень досягнуто за рахунок:

1. Високої криптостійкості: використання стохастичних процесів (ланцюгів Маркова) забезпечує непередбачуваність ключів.

2. Низької собівартості: реалізація на мові Python з використанням відкритих бібліотек не потребує дорогих ліцензійних відрахувань.

3. Універсальності: програмний модуль може бути інтегрований у різні системи (месенджери, банківські додатки) без значних змін архітектури.

4.2 Розрахунок витрат на виконання науково-дослідної роботи

Розрахуємо кошторисну вартість виконання роботи. До основних статей витрат належать: витрати на оплату праці, відрахування на соціальні заходи, амортизація обладнання, електроенергія та накладні витрати.

4.2.1 Витрати на оплату праці

Робота виконується одним інженером-програмістом (42 дні) під керівництвом наукового керівника (5 днів).

Місячний посадовий оклад інженера приймаємо на рівні 24 000 грн. Для наукового керівника місячний посадовий оклад приймаємо на рівні 18 000 грн.

Витрати на основну заробітну плату ($З_о$) розраховуємо за формулою (4.1):

$$З_о = \frac{M_{mi} \cdot t_i}{T_p} \quad (4.1)$$

Таблиця 4.4 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Кількість днів роботи	Витрати на заробітну плату, грн
Інженер-програміст	24000	1 142,86	42	48 000,00
Науковий керівник	18000	857,14	5	4285,71
Всього				52285,71

Додаткова заробітна плата ($З_{дод}$) розраховується як 10% від основної:

$$З_{дод} = З_о \cdot \frac{10}{100} \quad (4.2)$$

$$З_{дод} = 52285{,}71 \cdot 0{,}1 = 5228{,}57 \text{ грн.}$$

4.2.2 Відрахування на соціальні заходи

Відрахування на соціальні заходи (ЄСВ) становлять 22% від фонду оплати праці (формула 4.3):

$$З_{соц} = (З_о + З_{дод}) \cdot 0{,}22 \quad (4.3)$$

$$З_{соц} = (52285{,}71 + 5228{,}57) \cdot 0{,}22 = 12653{,}14 \text{ грн.}$$

4.2.3 Матеріали та комплектуючі

Витрати на матеріали є мінімальними і включають канцелярські товари та носії інформації, до вартості додається коефіцієнт транспортно-заготівельних витрат ($K_з = 1{,}1$) (табл.4.5).

Таблиця 4.5 – Витрати на матеріали та комплектуючі

Найменування	Кількість	Ціна за од., грн	Сума, грн
Папір офісний А4 (пачка)	1	150	150
Флеш-накопичувач 64GB	1	350	350
Всього (500 · 1,1)			550

4.2.4 Амортизація обладнання

Для виконання роботи використовувався персональний комп'ютер вартістю 25 000 грн. Згідно з Податковим кодексом України (група 4), мінімальний строк корисного використання для ЕОМ становить 2 роки. Для розрахунку приймемо реалістичний строк експлуатації – 4 роки (48 місяців). Термін використання під час досліджень – 2 місяці ($t_{вик} = 2$).

Амортизаційні відрахування ($А_{обл}$) розраховуємо за формулою (4.4):

$$А_{обл} = \frac{Ц_{обл}}{Т_к} \cdot \frac{t_{вик}}{12} \quad (4.4)$$

$$А_{обл} = \frac{25000}{2} \cdot \frac{2}{12} = 2083{,}33 \text{ грн.}$$

4.2.5 Паливо та енергія для науково-виробничих цілей

Потужність комп'ютера з монітором становить приблизно 0,25 кВт ($W_у$). Час роботи – 8 годин на день протягом 42 днів: $t = 42 \cdot 8 = 336$ годин.

Розрахунок тарифу на електроенергію ($Ц_е$) для непобутових споживачів (2 клас напруги) виконується за формулою (4.5):

$$Ц_е = (Ц_{опт} + Ц_{розп} + Ц_{пост}) \cdot 1{,}2 \quad (4.5)$$

Використовуючи актуальні дані для Вінницької області (ціни на 2025 р.):

- $Ц_{опт} \approx 6{,}80$ грн/кВт·год;
- $Ц_{розп} \approx 3{,}50$ грн/кВт·год;
- $Ц_{пост} \approx 1{,}95$ грн/кВт·год.

$$Ц_е = (6{,}80 + 3{,}50 + 1{,}95) \cdot 1{,}2 = 12{,}25 \cdot 1{,}2 = 14{,}70 \text{ грн/кВт} \cdot \text{год.}$$

Коефіцієнт використання потужності $К_{вик} = 0{,}8$. Витрати на електроенергію ($В_е$) за формулою (4.6):

$$В_е = W_у \cdot t \cdot Ц_е \cdot К_{вик} \quad (4.6)$$

$$В_е = 0{,}25 \cdot 336 \cdot 14{,}70 \cdot 0{,}8 = 987{,}84 \text{ грн.}$$

4.2.6 Витрати на програмне забезпечення

У роботі використовувалося програмне забезпечення з відкритим кодом (Open Source) та пробні версії, проте для коректного економічного обґрунтування врахуємо вартість місячної підписки на хмарні сервіси та IDE, які використовуються у комерційній розробці (таб.4.6).

Таблиця 4.6 – Витрати на ПЗ

Найменування ПЗ	Тип оплати	Вартість міс., грн.	Кількість міс.	Сума, грн.
JetBrains PyCharm (Pro)	Підписка	950	2	1900
Хмарні сервіси (тест)	Підписка	600	1	600
Всього ($В_{пз}$)				2 500,00

4.2.7 Накладні витрати

Накладні витрати ($В_{нв}$) включають витрати на управління, утримання приміщень, зв'язок тощо. Приймаємо їх у розмірі 120% від основної заробітної плати (формула 4.7):

$$В_{нв} = З_о \cdot 1{,}2 \tag{4.7}$$

$$В_{нв} = 52285{,}71 \cdot 1{,}2 = 62742{,}85 \text{ грн.}$$

4.2.8 Загальні витрати на виконання роботи

Розрахуємо сумарні витрати на проведення НДР ($В_{заг}$) за формулою (4.8):

$$В_{заг} = З_о + З_{дод} + З_{соц} + М + А_{обл} + В_е + В_{пз} + В_{нв} \tag{4.8}$$

$$В_{заг} = 52\,285{,}71 + 5\,228{,}57 + 12\,653{,}14 + 550 + 2\,083{,}33 + 987{,}84 + 2\,500 + 62\,742{,}85$$

$$В_{заг} = 139031{,}44 \text{ грн.}$$

Оскільки розробка знаходиться на стадії створення дослідного зразка та готова до впровадження, загальні витрати на завершення роботи (ЗВ) розраховуємо з урахуванням коефіцієнта етапу $\eta = 0{,}9$ (формула 4.9):

$$ЗВ = \frac{В_{заг}}{\eta} \tag{4.9}$$

$$ЗВ = \frac{139031{,}44}{0{,}9} = 154479{,}38 \text{ грн.}$$

Отже, повна собівартість науково-технічної розробки становить 154479,38 грн.

4.3 Розрахунок економічної ефективності науково-технічної розробки

Розрахунок ефективності проведемо для сценарію комерціалізації продукту (продажу ліцензій на програмний модуль шифрування).

4.3.1 Розрахунок приведеної вартості чистих прибутків

Припустимо, що продукт буде реалізовуватися за моделлю ліцензій.

- Вартість однієї ліцензії ($Ц_о$): 3 000 грн.

- Собівартість підтримки однієї копії: 200 грн/рік.
- Чистий прибуток з однієї копії: 2 800 грн.

Прогноз продажів (ΔN):

- 1-й рік: 50 ліцензій (активний старт продажів).
- 2-й рік: 100 ліцензій (масштабування).
- 3-й рік: 150 ліцензій (вихід на стабільний ринок).

Чистий прибуток ($\Delta П_i$) по роках:

- $\Delta П_1 = 50 \cdot 2800 = 140000$ грн.
- $\Delta П_2 = 100 \cdot 2800 = 280000$ грн.
- $\Delta П_3 = 150 \cdot 2800 = 420000$ грн.

Приведену вартість прибутків (ПП) розраховуємо за формулою (4.10) з ставкою дисконтування $\tau = 0{,}2$ (20% - враховуючи ризики IT-стартапу):

$$ПП = \sum_{i=1}^{3} \frac{\Delta П_i}{(1+\tau)^t} \quad (4.10)$$

$$ПП = \frac{140000}{(1{,}2)^1} + \frac{280000}{(1{,}2)^2} + \frac{420000}{(1{,}2)^3}$$

$$ПП = 116666{,}67 + 194444{,}44 + 243055{,}56 = 554166{,}67 \text{ грн.}$$

4.3.2 Розрахунок абсолютного економічного ефекту

Величина початкових інвестицій (PV) дорівнює вартості розробки з урахуванням коефіцієнта витрат на впровадження $k_{розр} = 1{,}5$ (формула 4.11):

$$PV = k_{розр} \cdot ЗВ \quad (4.11)$$

$$PV = 1{,}5 \cdot 154479{,}38 = 231719{,}07 \text{ грн.}$$

Абсолютний економічний ефект (NPV) розраховуємо за формулою (4.12):

$$E_{абс} = ПП - PV \quad (4.12)$$

$$E_{абс} = 554166{,}67 - 231719{,}07 = 322447{,}60 \text{ грн.}$$

Позитивне значення NPV свідчить про доцільність впровадження розробки.

4.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

4.4.1 Внутрішня економічна дохідність

Внутрішня норма дохідності ($E_{\text{в}}$) для життєвого циклу $T_{\text{ж}} = 3$ роки (формула 4.13):

$$E_{\text{в}} = \sqrt[3]{1 + \frac{E_{\text{абс}}}{PV}} - 1 \tag{4.13}$$

$$E_{\text{в}} = \sqrt[3]{1 + \frac{322\,447{,}60}{231\,719{,}07}} - 1 = \sqrt[3]{1 + 1{,}39} - 1$$

$$E_{\text{в}} = \sqrt[3]{2{,}39} - 1 \approx 1{,}337 - 1 = 0{,}337$$

Отримане значення $E_{\text{в}} = 33{,}7\%$ перевищує депозитні ставки, що підтверджує ефективність проєкту.

4.4.2 Термін окупності інвестицій

Дисконтований період окупності ($T_{\text{ок}}$) розраховуємо за формулою (4.14):

$$T_{\text{ок}} = \frac{1}{E_{\text{в}}} \tag{4.14}$$

$$T_{\text{ок}} = \frac{1}{0{,}337} \approx 2{,}96 \text{ роки}$$

Для точнішого визначення проведемо розрахунок за грошовими потоками:

1. Інвестиції: 231 719,07 грн.

2. Прибуток 1-го року (дисконтований): 116 666,67 грн. (Залишок боргу: 115 052,40 грн).

3. Прибуток 2-го року (дисконтований): 194 444,44 грн.

Оскільки прибуток другого року повністю покриває залишок боргу, окупність настане всередині другого року:

$$T_{\text{ок(факт)}} = 1 + \frac{115\,052{,}40}{194\,444{,}44}$$

$$T_{\text{ок(факт)}} 1 + 0{,}59 = 1{,}59 \text{ року}$$

Отриманий фактичний термін окупності становить 1,59 року (близько 1 року

і 7 місяців), що задовольняє умову $T_{\text{ок}} < 3$ років.

4.5 Висновки до розділу

У розділі проведено економічне обґрунтування розробки.

1. Собівартість розробки становить 154 479,38 грн.

2. Інвестиційні витрати на впровадження становлять 231 719,07 грн.

3. Економічна ефективність:

- Чистий приведений дохід (NPV): 322 447,60 грн.
- Внутрішня норма дохідності ($E_{\text{в}}$): 33,7%.
- Фактичний термін окупності: 1,6 року.

Проєкт є економічно привабливим та швидкоокупним.

Отримані показники підтверджують економічну доцільність реалізації проєкту.

ВИСНОВКИ

У результаті виконання магістерської кваліфікаційної роботи вирішено завдання підвищення криптографічної стійкості систем передачі даних шляхом вдосконалення методу динамічної генерації ключів. Основні результати роботи полягають у наступному:

У першому розділі проаналізовано методи симетричного шифрування та генерації ключів. Встановлено, що використання статичних ключів є критичною вразливістю сучасних систем. Обґрунтовано доцільність використання стохастичних процесів для забезпечення непередбачуваності ключових послідовностей.

У другому розділі розроблено вдосконалений метод генерації ключів на основі ланцюгів Маркова. Створено алгоритм, що передбачає динамічну зміну матриці ймовірностей переходів, гарантуючи унікальність ключа для кожної сесії. Спроєктовано структуру програмної системи, яка забезпечує адаптивність методу до умов функціонування мережі.

У третьому розділі здійснено програмну реалізацію методу мовою Python. Статистичне тестування за стандартом NIST STS підтвердило високі ентропійні властивості згенерованих ключів (P-value > 0,01). Оцінка продуктивності показала, що час обробки даних зростає лише на 24,5%, що є допустимим компромісом для суттєвого підвищення рівня безпеки.

У четвертому розділі проведено економічне обґрунтування розробки. Розрахункова вартість впровадження становить 154479,38 грн. Термін окупності проекту — 1,6 року, а коефіцієнт рентабельності інвестицій складає 33,7%, що підтверджує економічну ефективність власного рішення порівняно з комерційними аналогами.

Загалом, розроблений метод динамічної генерації ключів дозволяє ефективно захищати текстові повідомлення від криптоаналізу та може бути інтегрований у сучасні системи інформаційної безпеки.

ПЕРЕЛІК ПОСИЛАНЬ

1. Горбенко І. Д., Горбенко Ю. І. Прикладна криптологія: теорія, практика, застосування : монографія. 2-ге вид., перероб. і допов. Харків : Форт, 2021. 878 с.

2. Тимченко Д. І., Салієва О. В. Застосування стохастичних процесів для шифрування текстових повідомлень. Матеріали науково-технічної конференції ВНТУ (Вінниця, 2025 р.). Вінниця : ВНТУ, 2025. С. 1–3. (дата звернення: 10.09.2025).

3. Advanced Encryption Standard (AES) : FIPS PUB 197. National Institute of Standards and Technology (NIST), 2001. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf (дата звернення: 10.09.2025).

4. Шнаєр Б. Прикладна криптографія. Протоколи, алгоритми, вихідні тексти на мові С. Київ : Діалектика, 2019. 1040 с.

5. Гнатюк С. О., Жердєв М. К. Сучасна криптографія. Основні поняття та алгоритми : навч. посіб. Київ : НАУ, 2022. 244 с.

6. Мельник В. І. Вступ до криптографії : конспект лекцій. Вінниця : ВНТУ, 2023. 120 с.

7. Data Encryption Standard (DES) : FIPS PUB 46-3. National Institute of Standards and Technology (NIST), 1999. URL: https://csrc.nist.gov/files/pubs/fips/46-3/final/docs/fips46-3.pdf (дата звернення: 12.09.2025).

8. Daemen J., Rijmen V. The Design of Rijndael: AES - The Advanced Encryption Standard. Berlin : Springer-Verlag, 2020. 238 p.

9. Katz J., Lindell Y. Introduction to Modern Cryptography. 3rd ed. Boca Raton : CRC Press, 2021. 600 p.

10. Anderson R. Security Engineering: A Guide to Building Dependable Distributed Systems. 3rd ed. New York : Wiley, 2021. 1200 p.

11. Recommendation for Block Cipher Modes of Operation: Methods and Techniques : NIST SP 800-38A. NIST, 2001. URL: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=900346 (дата звернення:

15.09.2025).

12. Bernstein D. J. ChaCha, a variant of Salsa20. *CR.YP.TO : website*. 2008. URL: https://cr.yp.to/chacha/chacha-20080128.pdf (дата звернення: 18.09.2025).

13. Nir Y., Langley A. ChaCha20 and Poly1305 for IETF Protocols : RFC 7539. Internet Engineering Task Force, 2015. URL: https://tools.ietf.org/html/rfc7539 (дата звернення: 20.09.2025).

14. Rivest R. L. The RC4 Encryption Algorithm. *MIT CSAIL*. URL: https://people.csail.mit.edu/rivest/pubs/Riv92.pdf (дата звернення: 12.10.2025).

15. Barker E. Recommendation for Key Management : NIST SP 800-57 Part 1 Rev. 5. NIST, 2020. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf (дата звернення: 10.10.2025).

16. Recommendation for Random Number Generation Using Deterministic Random Bit Generators : NIST SP 800-90A Rev. 1. NIST, 2015. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf (дата звернення: 22.09.2025).

17. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications : NIST SP 800-22 Rev. 1a. NIST, 2010. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf (дата звернення: 25.09.2025).

18. Eastlake D., Schiller J., Crocker S. Randomness Requirements for Security : RFC 4086. IETF, 2005. URL: https://www.ietf.org/rfc/rfc4086.txt (дата звернення: 26.09.2025).

19. Ferguson N., Schneier B., Kohno T. Cryptography Engineering: Design Principles and Practical Applications. Indianapolis : Wiley, 2020. 384 p.

20. Гіхман Й. І., Скороход А. В. Вступ до теорії випадкових процесів. Київ : Наукова думка, 2018. 560 с.

21. Norris J. R. Markov Chains. Cambridge : Cambridge University Press, 2018. 237 p.

22. Довженко М. О. Використання ланцюгів Маркова для аналізу безпеки

інформаційних систем. *Вісник КПІ. Серія: Інформатика та обчислювальна техніка*. 2022. № 55. С. 24–31. URL: http://vestnik.kpi.ua/article/view/25678 (дата звернення: 28.10.2025).

23. Савченко В. В., Гнатюк С. О. Сучасні методи генерації псевдовипадкових послідовностей у криптографічних системах. *Захист інформації*. 2023. Т. 25, № 1. С. 45–52.

24. Інформаційні технології. Методи захисту. Методи генерування випадкових бітів : ДСТУ ISO/IEC 18031:2015. Київ : ДП «УкрНДНЦ», 2016. 98 с.

25. Системи обробки інформації. Кодування та захист інформації. Терміни та визначення понять : ДСТУ 2226-93. Київ : Держстандарт України, 1994. 65 с.

26. Про захист інформації в інформаційно-комунікаційних системах : Закон України від 05.07.1994 № 80/94-ВР. *Верховна Рада України : офіц. вебпортал*. URL: https://zakon.rada.gov.ua/laws/show/80/94-%D0%B2%D1%80 (дата звернення: 20.10.2025).

27. Про затвердження Правил надання послуг у сфері технічного захисту інформації : Наказ Адміністрації Держспецзв'язку від 16.05.2007 № 93. URL: https://zakon.rada.gov.ua/laws/show/z0848-07 (дата звернення: 20.10.2025).

28. Python Cryptography Toolkit (PyCryptodome). *PyCryptodome Documentation*. URL: https://www.pycryptodome.org/ (дата звернення: 15.10.2025).

29. Random sampling (numpy.random). *NumPy v1.26 Manual*. URL: https://numpy.org/doc/stable/reference/random/index.html (дата звернення: 16.10.2025).

30. Statistical functions (scipy.stats). Entropy. *SciPy Documentation*. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.entropy.html (дата звернення: 17.10.2025).

31. Generate secure random numbers for managing secrets. *Python 3.12 Documentation*. URL: https://docs.python.org/3/library/secrets.html (дата звернення: 18.10.2025).

32. Cryptographic Storage Cheat Sheet. *OWASP Foundation*. URL: https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cheat_Sheet.htm

l (дата звернення: 19.10.2025).

33. Computer Security Resource Center. Glossary: Entropy. *NIST CSRC*. URL: https://csrc.nist.gov/glossary/term/entropy (дата звернення: 21.10.2025).

34. What is AES Encryption and How Does it Work? *Simplilearn*. 2024. URL: https://www.simplilearn.com/tutorials/cryptography-tutorial/aes-encryption (дата звернення: 22.10.2025).

35. Symmetric vs. Asymmetric Encryption. *Cloudflare Learning*. URL: https://www.cloudflare.com/learning/ssl/what-is-symmetric-encryption/ (дата звернення: 23.10.2025).

36. True Random Number Service. *Random.org*. URL: https://www.random.org/ (дата звернення: 24.10.2025).

37. Biham E., Shamir A. Differential Cryptanalysis of the Data Encryption Standard. New York : Springer-Verlag, 1993. 190 p.

38. Matsui M. Linear Cryptanalysis Method for DES Cipher. *Advances in Cryptology – EUROCRYPT '93*. Berlin : Springer, 1994. P. 386–397.

39. Matsumoto M., Nishimura T. Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. *ACM Transactions on Modeling and Computer Simulation*. 1998. Vol. 8, no. 1. P. 3–30.

40. Marsaglia G. Xorshift RNGs. *Journal of Statistical Software*. 2003. Vol. 8, no. 14. P. 1–6. URL: https://www.jstatsoft.org/article/view/v008i14 (дата звернення: 24.10.2025).

41. Wiener M. J. Cryptanalysis of Short RSA Secret Exponents. *IEEE Transactions on Information Theory*. 2019. Vol. 36, no. 3. P. 553–558.

ДОДАТКИ

Додаток А. Технічне завдання

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

ЗАТВЕРДЖУЮ
Голова секції "Управління інформаційною
безпекою" кафедри МБІС
д.т.н., професор
_____ Юрій ЯРЕМЧУК
"24" вересня 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

до магістерської кваліфікаційної роботи на тему:
Вдосконалення методу динамічної генерації ключів для симетричного шифрування текстових повідомлень на основі стохастичних процесів
08-72.МКР.018.00.081.ТЗ

Керівник магістерської кваліфікаційної роботи
д.ф., допент
_____ Салієва О. В

Вінниця – 2025 р.

1. Найменування та область застосування

Програмний засіб для динамічної генерації криптографічних ключів на основі стохастичних процесів. Область застосування: захист конфіденційної текстової інформації в корпоративних месенджерах, системах електронного документообігу та телекомунікаційних мережах від несанкціонованого доступу та криптоаналізу.

2.Підстава для розробки

Розробка виконується на основі наказу ректора ВНТУ №96 від 20.03.2025 р.

3. Мета та призначення розробки

3.1 Мета розробки: підвищення загальної криптостійкості систем симетричного шифрування шляхом розробки та програмної реалізації методу динамічної генерації ключів, що забезпечує їх непередбачуваність та високу ентропію.

3.2 Призначення: розроблений програмний засіб виконує генерацію унікальних сесійних ключів на основі ланцюгів Маркова, здійснює шифрування та розшифрування текстових повідомлень, а також проводить статистичний аналіз якості згенерованих ключів.

4. Джерела розробки

4.1. Горбенко І. Д., Горбенко Ю. І. Прикладна криптологія: теорія, практика, застосування : монографія. Харків : Форт, 2021. 878 с.

4.2. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications : NIST SP 800-22 Rev. 1a. NIST, 2010.

4.3. Recommendation for Random Number Generation Using Deterministic Random Bit Generators : NIST SP 800-90A Rev. 1. NIST, 2015.

4.4. Гіхман Й. І., Скороход А. В. Вступ до теорії випадкових процесів. Київ : Наукова думка, 2018. 560 с.

5. Вимоги до програми

5.1 Вимоги до функціональних характеристик:

5.1.1 Програмний засіб повинен мати зручний, легкий у використанні інтерфейс користувача;

5.1.2 Реалізація методу не повинна вимагати спеціальних ліцензійних програмних додатків;

5.1.3 Програмний засіб повинен реалізовувати алгоритм генерації ключів на основі стохастичних ланцюгів Маркова.

5.2 Вимоги до надійності:

5.2.1 Програмний засіб повинен працювати без помилок, у випадку виникнення критичних ситуацій необхідно передбачити виведення відповідних повідомлень;

5.2.2 Процес розшифрування повинен однозначно відновлювати вихідний текст за умови наявності правильного ключа;

5.2.3 Програмний засіб повинен виконувати свої функції.

5.3 Вимоги до складу і параметрів технічних засобів:

– процесор – Pentium 2000 МГц і подібні до них;

– оперативна пам’ять – не менше 1024 Mb;

– середовище функціонування – операційна система сімейство Windows;

– вимоги до техніки безпеки при роботі з програмою повинні відповідати існуючим вимогам та стандартам з техніки безпеки при користуванні комп’ютерною технікою.

6. Вимоги до програмної документації

6.1 Обов’язкова поетапна інструкція для майбутніх користувачів, наведена у пункті 3.4

7. Вимоги до технічного захисту інформації

7.1 Необхідно забезпечити захист розроблюваного програмного засобу від несанкціонованого використання.

7.2 Неможливість отримання доступу незареєстрованих користувачів до інформаційних ресурсів.

8. Техніко-економічні показники

8.1 Цінність результатів використання даного проекту повинна перевищувати витрати на його реалізацію.

8.2 Має бути реалізований таким чином, щоб підходити для використання

9. Стадії та етапи розробки

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Початок	Закінчення
1	Визначення напрямку магістерської роботи, формулювання теми	24.09.2025	30.09.2025
2	Аналіз предметної області обраної теми	01.10.2025	10.10.2025
3	Апробація отриманих результатів	11.10.2025	20.10.2025
4	Розробка алгоритму роботи	21.10.2025	05.11.2025
5	Написання магістерської роботи на основі розробленої теми	06.11.2025	15.11.2025
6	Розробка економічної частини	16.11.2025	20.11.2025
7	Передзахист магістерської кваліфікаційної роботи	21.11.2025	27.11.2025
8	Виправлення, уточнення, корегування магістерської кваліфікаційної роботи	28.11.2025	30.11.2025
9	Захист магістерської кваліфікаційної роботи	08.12.2025	12.12.2025

10. Порядок контролю та прийому

10.1 До приймання магістерської кваліфікаційної роботи надається:

– ПЗ до магістерської кваліфікаційної роботи;

– програмний додаток;

– презентація;

– відгук керівника роботи;

– відгук опонента

Технічне завдання до виконання прийняв _____ Тимченко Д. І.

Додаток Б. Лістинг програми

```
import numpy as np
import hashlib
import secrets
from typing import List, Tuple
from dataclasses import dataclass
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
import matplotlib.pyplot as plt
from scipy import stats
import time
import customtkinter as ctk
import time
from datetime import datetime
from threading import Thread
import tkinter as tk

@dataclass
class KeyMetrics:
    """Метрики якості згенерованого ключа"""
    entropy: float
    randomness_p_value: float
    autocorrelation: float
    generation_time: float

class StochasticKeyGenerator:

    def __init__(self, seed: int = None):

        if seed is not None:
            np.random.seed(seed)
        self.seed = seed

    def wiener_process(self, steps: int, dt: float = 0.01) -> np.ndarray:

        dW = np.random.normal(0, np.sqrt(dt), steps)
        W = np.cumsum(dW)
        return W

    def ornstein_uhlenbeck_process(self, steps: int, theta: float = 0.15,
                                   mu: float = 0, sigma: float = 0.3,
                                   dt: float = 0.01) -> np.ndarray:

        x = np.zeros(steps)
        x[0] = mu

        for i in range(1, steps):
            dx = theta * (mu - x[i-1]) * dt + sigma * np.random.normal(0, np.sqrt(dt))
            x[i] = x[i-1] + dx

        return x
```

```
    def geometric_brownian_motion(self, steps: int, S0: float = 100,
                                  mu: float = 0.1, sigma: float = 0.3,
                                  dt: float = 0.01) -> np.ndarray:

        W = self.wiener_process(steps, dt)
        t = np.linspace(0, steps * dt, steps)
        S = S0 * np.exp((mu - 0.5 * sigma**2) * t + sigma * W)
        return S

    def jump_diffusion_process(self, steps: int, S0: float = 100,
                               mu: float = 0.1, sigma: float = 0.2,
                               lambda_jump: float = 0.1, jump_mean: float = 0,
                               jump_std: float = 0.1, dt: float = 0.01) -> np.ndarray:

        S = np.zeros(steps)
        S[0] = S0

        for i in range(1, steps):

            dW = np.random.normal(0, np.sqrt(dt))
            diffusion = mu * dt + sigma * dW

            jump = 0
            if np.random.random() < lambda_jump * dt:
                jump = np.random.normal(jump_mean, jump_std)

            S[i] = S[i-1] * np.exp(diffusion + jump)

        return S

    def generate_key_from_process(self, process_values: np.ndarray,
                                  key_size: int = 32) -> bytes:

        normalized = ((process_values - process_values.min()) /
                      (process_values.max() - process_values.min()) * 255)

        byte_array = normalized.astype(np.uint8).tobytes()

        hash_obj = hashlib.sha256(byte_array)

        if key_size > 32:
            hash_obj = hashlib.shake_256(byte_array)
            key = hash_obj.digest(key_size)
        else:
            key = hash_obj.digest()[:key_size]

        additional_entropy = secrets.token_bytes(16)
        final_hash = hashlib.sha256(key + additional_entropy)
        final_key = final_hash.digest()[:key_size]

        return final_key

class DynamicKeyManager:
```

```
    def __init__(self, key_generator: StochasticKeyGenerator,
                 key_rotation_interval: int = 10):

        self.key_generator = key_generator
        self.key_rotation_interval = key_rotation_interval
        self.current_key = None
        self.key_history = []
        self.message_counter = 0

    def get_current_key(self, force_new: bool = False) -> bytes:

        if (self.current_key is None or
            self.message_counter >= self.key_rotation_interval or
            force_new):
            self._generate_new_key()
            self.message_counter = 0

        self.message_counter += 1
        return self.current_key

    def _generate_new_key(self):

        process_type = np.random.choice([
            'wiener', 'ornstein_uhlenbeck',
            'geometric_brownian', 'jump_diffusion'
        ])

        steps = np.random.randint(100, 500)

        if process_type == 'wiener':
            process = self.key_generator.wiener_process(steps)
        elif process_type == 'ornstein_uhlenbeck':
            process = self.key_generator.ornstein_uhlenbeck_process(steps)
        elif process_type == 'geometric_brownian':
            process = self.key_generator.geometric_brownian_motion(steps)
        else:  # jump_diffusion
            process = self.key_generator.jump_diffusion_process(steps)

        self.current_key = self.key_generator.generate_key_from_process(process)
        self.key_history.append({
            'key': self.current_key,
            'process_type': process_type,
            'timestamp': time.time()
        })

class StochasticEncryptionSystem:

    def __init__(self, key_manager: DynamicKeyManager):

        self.key_manager = key_manager

    def encrypt(self, plaintext: str) -> Tuple[bytes, bytes, bytes]:

        key = self.key_manager.get_current_key()
```

```
        iv = secrets.token_bytes(16)

        cipher = AES.new(key, AES.MODE_CBC, iv)

        plaintext_bytes = plaintext.encode('utf-8')
        ciphertext = cipher.encrypt(pad(plaintext_bytes, AES.block_size))

        return ciphertext, key, iv

    def decrypt(self, ciphertext: bytes, key: bytes, iv: bytes) -> str:

        cipher = AES.new(key, AES.MODE_CBC, iv)
        plaintext_bytes = unpad(cipher.decrypt(ciphertext), AES.block_size)
        return plaintext_bytes.decode('utf-8')

class CryptoAnalyzer:


    @staticmethod
    def calculate_entropy(data: bytes) -> float:

        if len(data) == 0:
            return 0.0

        byte_counts = np.bincount(np.frombuffer(data, dtype=np.uint8), minlength=256)
        probabilities = byte_counts[byte_counts > 0] / len(data)

        entropy = -np.sum(probabilities * np.log2(probabilities))
        return entropy

    @staticmethod
    def runs_test(data: bytes) -> float:

        bits = np.unpackbits(np.frombuffer(data, dtype=np.uint8))
        n = len(bits)

        runs = 1
        for i in range(1, n):
            if bits[i] != bits[i-1]:
                runs += 1

        ones = np.sum(bits)
        zeros = n - ones

        expected_runs = (2 * ones * zeros) / n + 1
        variance_runs = (2 * ones * zeros * (2 * ones * zeros - n)) / (n**2 * (n - 1))

        if variance_runs == 0:
            return 0.5

        z = (runs - expected_runs) / np.sqrt(variance_runs)
        p_value = 2 * (1 - stats.norm.cdf(abs(z)))
```

```
        return p_value

    @staticmethod
    def autocorrelation(data: bytes, lag: int = 1) -> float:

        arr = np.frombuffer(data, dtype=np.uint8).astype(float)
        n = len(arr)

        if n <= lag:
            return 0.0

        mean = np.mean(arr)
        c0 = np.sum((arr - mean) ** 2) / n

        if c0 == 0:
            return 0.0

        c_lag = np.sum((arr[:-lag] - mean) * (arr[lag:] - mean)) / n
        return c_lag / c0

    def analyze_key(self, key: bytes) -> KeyMetrics:

        start_time = time.time()

        entropy = self.calculate_entropy(key)
        randomness_p_value = self.runs_test(key)
        autocorr = self.autocorrelation(key)

        generation_time = time.time() - start_time

        return KeyMetrics(
            entropy=entropy,
            randomness_p_value=randomness_p_value,
            autocorrelation=autocorr,
            generation_time=generation_time
        )

class ChatApplication:

    def __init__(self):
        self.key_gen = StochasticKeyGenerator()
        self.key_manager = DynamicKeyManager(self.key_gen, key_rotation_interval=1)
        self.encryption_system = StochasticEncryptionSystem(self.key_manager)
        self.analyzer = CryptoAnalyzer()
        self.messages = []
        self.key_metrics_list = []

    def send_message(self, sender: str, text: str):

        ciphertext, key, iv = self.encryption_system.encrypt(text)

        metrics = self.analyzer.analyze_key(key)
        self.key_metrics_list.append(metrics)
```

```
        message_data = {
            'sender': sender,
            'text': text,
            'ciphertext': ciphertext,
            'key': key,
            'iv': iv,
            'metrics': metrics,
            'timestamp': time.time()
        }

        self.messages.append(message_data)
        return message_data

    def display_message(self, msg_data):

        print(f"\n[{msg_data['sender']}]: {msg_data['text']}")
        print(f"{msg_data['ciphertext'].hex()[:64]}...")
        print(f"{msg_data['metrics'].entropy:.4f}")
        print(f"{msg_data['metrics'].randomness_p_value:.4f}")
        print(f"{msg_data['metrics'].autocorrelation:.4f}")

    def show_statistics(self):

        if not self.key_metrics_list:
            return

        entropies = [m.entropy for m in self.key_metrics_list]
        p_values = [m.randomness_p_value for m in self.key_metrics_list]
        autocorrs = [m.autocorrelation for m in self.key_metrics_list]

        print(f"\n{np.mean(entropies):.4f}")
        print(f"{np.std(entropies):.4f}")
        print(f"{np.mean(p_values):.4f}")
        print(f"{np.std(p_values):.4f}")
        print(f"{np.mean(autocorrs):.4f}")
        print(f"{np.std(autocorrs):.4f}")

    def generate_graphs(self):

        if not self.key_metrics_list:
            return

        entropies = [m.entropy for m in self.key_metrics_list]
        p_values = [m.randomness_p_value for m in self.key_metrics_list]
        autocorrs = [m.autocorrelation for m in self.key_metrics_list]

        fig, axes = plt.subplots(2, 2, figsize=(14, 10))

        axes[0, 0].plot(range(1, len(entropies) + 1), entropies, 'o-', linewidth=2, markersize=8)
        axes[0, 0].axhline(y=8.0, color='r', linestyle='--')
        axes[0, 0].set_xlabel('Номер ключа')
        axes[0, 0].set_ylabel('Ентропія')
        axes[0, 0].grid(True, alpha=0.3)
```

```
        axes[0, 1].plot(range(1, len(p_values) + 1), p_values, 's-', linewidth=2, markersize=8, color='green')
        axes[0, 1].axhline(y=0.05, color='r', linestyle='--')
        axes[0, 1].set_xlabel('Номер ключа')
        axes[0, 1].set_ylabel('p-value')
        axes[0, 1].grid(True, alpha=0.3)

        axes[1, 0].plot(range(1, len(autocorrs) + 1), autocorrs, '^-', linewidth=2, markersize=8, color='orange')
        axes[1, 0].axhline(y=0, color='r', linestyle='--')
        axes[1, 0].set_xlabel('Номер ключа')
        axes[1, 0].set_ylabel('Автокореляція')
        axes[1, 0].grid(True, alpha=0.3)

        if self.messages:
            first_key = self.messages[0]['key']
            key_bytes = np.frombuffer(first_key, dtype=np.uint8)
            axes[1, 1].hist(key_bytes, bins=32, edgecolor='black', alpha=0.7)
            axes[1, 1].set_xlabel('Значення байта')
            axes[1, 1].set_ylabel('Частота')
            axes[1, 1].grid(True, alpha=0.3, axis='y')

        plt.tight_layout()
        plt.savefig('chat_analysis.png', dpi=300, bbox_inches='tight')
        plt.close()

def run_chat():

    chat = ChatApplication()

    print("=== CHAT ===\n")

    while True:
        print("\n1 - Надіслати")
        print("2 - Статистика")
        print("3 - Графіки")
        print("4 - Вихід")

        choice = input("\n> ").strip()

        if choice == '1':
            sender = input("Ім'я: ").strip()
            if not sender:
                sender = "User"
            text = input("Текст: ").strip()
            if text:
                msg_data = chat.send_message(sender, text)
                chat.display_message(msg_data)

        elif choice == '2':
            chat.show_statistics()

        elif choice == '3':
            chat.generate_graphs()
            print("\nchat_analysis.png")
```

```
        elif choice == '4':
            break

class MessageBubble(ctk.CTkFrame):

    def __init__(self, master, text, sender, is_me, metrics=None, key=None, ciphertext=None, **kwargs):
        super().__init__(master, **kwargs)

        color = "#2b5278" if is_me else "#182533"
        self.configure(fg_color=color, corner_radius=16)

        self.lbl_text = ctk.CTkLabel(self,
                                     text=text,
                                     wraplength=380,
                                     justify="left",
                                     font=("Roboto", 24))
        self.lbl_text.pack(padx=15, pady=(10, 5), anchor="w")

        if metrics and key and ciphertext:

            key_hex = key.hex().upper()

            meta_text = (
                f"🔑 KEY (Full): {key_hex}\n"
                f"🎲 Entropy: {metrics.entropy:.4f} | p-value: {metrics.randomness_p_value:.4f}"
            )

            meta_frame = ctk.CTkFrame(self, fg_color="#000000", corner_radius=6)
            meta_frame.pack(padx=10, pady=(5, 5), fill="x")

            self.lbl_meta = ctk.CTkLabel(meta_frame,
                                         text=meta_text,
                                         font=("Consolas", 11),
                                         text_color="#00ff00",
                                         wraplength=360,
                                         justify="left")
            self.lbl_meta.pack(padx=5, pady=5, anchor="w")

        time_str = datetime.now().strftime("%H:%M:%S")
        self.lbl_time = ctk.CTkLabel(self, text=time_str, font=("Roboto", 10), text_color="gray60")
        self.lbl_time.pack(padx=10, pady=(0, 5), anchor="e")

class ChatUserInterface(ctk.CTkFrame):

    def __init__(self, master, name, chat_app_ref, other_ui_ref=None, **kwargs):
        super().__init__(master, **kwargs)
        self.name = name
        self.chat_app = chat_app_ref
        self.other_ui = other_ui_ref

        self.header = ctk.CTkLabel(self, text=f"👤 {name}", font=("Roboto", 20, "bold"))
        self.header.pack(pady=10)
```

```
        self.scroll_frame = ctk.CTkScrollableFrame(self, fg_color="transparent")
        self.scroll_frame.pack(expand=True, fill="both", padx=5, pady=5)

        self.input_frame = ctk.CTkFrame(self, fg_color="transparent")
        self.input_frame.pack(fill="x", padx=10, pady=10)

        self.entry = ctk.CTkEntry(self.input_frame, placeholder_text="Повідомлення...", height=50,
font=("Roboto", 16))
        self.entry.pack(side="left", expand=True, fill="x", padx=(0, 10))
        self.entry.bind("<Return>", self.on_send)

        self.btn_send = ctk.CTkButton(self.input_frame, text="➤", width=50, height=50, font=("Roboto", 20),
command=self.on_send)
        self.btn_send.pack(side="right")

    def set_other_ui(self, other_ui):
        self.other_ui = other_ui

    def on_send(self, event=None):
        text = self.entry.get()
        if not text:
            return
        self.entry.delete(0, 'end')

        msg_data = self.chat_app.send_message(self.name, text)

        self.add_message(msg_data, is_me=True)
        if self.other_ui:
            self.other_ui.add_message(msg_data, is_me=False)

    def add_message(self, msg_data, is_me):
        bubble = MessageBubble(
            self.scroll_frame,
            text=msg_data['text'],
            sender=msg_data['sender'],
            is_me=is_me,
            metrics=msg_data['metrics'],
            key=msg_data['key'],
            ciphertext=msg_data['ciphertext']
        )

        padx = (40, 5) if is_me else (5, 40)
        bubble.pack(anchor="e" if is_me else "w", pady=10, padx=padx)
        self.scroll_frame._parent_canvas.yview_moveto(1.0)

class StatsPanel(ctk.CTkFrame):

    def __init__(self, master, **kwargs):
        super().__init__(master, height=120, **kwargs)

        self.lbl_stat = ctk.CTkLabel(self, text="📊 КЛЮЧОВІ ПОКАЗНИКИ В РЕАЛЬНОМУ ЧАСІ", font=("Roboto",
16, "bold"))
        self.lbl_stat.pack(pady=5)
```

```
        self.grid_frame = ctk.CTkFrame(self, fg_color="transparent")
        self.grid_frame.pack(fill="x", padx=20)

        self.val_entropy = self.create_metric(self.grid_frame, "Entropy (Shannon)", "0.0000", 0)
        self.val_pvalue = self.create_metric(self.grid_frame, "P-Value (Runs Test)", "0.0000", 1)
        self.val_auto = self.create_metric(self.grid_frame, "Autocorrelation", "0.0000", 2)
        self.val_gen_time = self.create_metric(self.grid_frame, "Gen Time (sec)", "0.0000", 3)

    def create_metric(self, parent, label, value, col):
        frame = ctk.CTkFrame(parent, fg_color="transparent")
        frame.grid(row=0, column=col, sticky="ew", padx=10)
        parent.grid_columnconfigure(col, weight=1)

        ctk.CTkLabel(frame, text=label, font=("Roboto", 12), text_color="gray").pack()

        lbl_val = ctk.CTkLabel(frame, text=value, font=("Roboto", 22, "bold"), text_color="#3B8ED0")
        lbl_val.pack()
        return lbl_val

    def update_stats(self, metrics):
        self.val_entropy.configure(text=f"{metrics.entropy:.4f}")

        color = "#00ff00" if metrics.randomness_p_value > 0.05 else "#ff5555"
        self.val_pvalue.configure(text=f"{metrics.randomness_p_value:.4f}", text_color=color)

        self.val_auto.configure(text=f"{metrics.autocorrelation:.4f}")
        self.val_gen_time.configure(text=f"{metrics.generation_time:.6f}")

class SecureChatApp(ctk.CTk):
    def __init__(self):
        super().__init__()
        self.title("Стохастична система шифрування")
        self.geometry("1200x800") # Збільшене стартове вікно
        ctk.set_appearance_mode("Dark")

        self.logic_app = ChatApplication()

        self.grid_columnconfigure(0, weight=1)
        self.grid_columnconfigure(1, weight=1)
        self.grid_rowconfigure(0, weight=1)
        self.grid_rowconfigure(1, weight=0)

        self.user1_ui = ChatUserInterface(self, "Вікторія", self, fg_color="#212121")
        self.user1_ui.grid(row=0, column=0, sticky="nsew", padx=2, pady=2)

        self.user2_ui = ChatUserInterface(self, "Денис", self, fg_color="#212121")
        self.user2_ui.grid(row=0, column=1, sticky="nsew", padx=2, pady=2)

        self.user1_ui.set_other_ui(self.user2_ui)
        self.user2_ui.set_other_ui(self.user1_ui)

        self.stats_panel = StatsPanel(self, fg_color="#1a1a1a")
        self.stats_panel.grid(row=1, column=0, columnspan=2, sticky="ew", padx=5, pady=5)
```

```
        self.btn_graphs = ctk.CTkButton(self.stats_panel, text="📈 Показати графіки аналізу",
                                        command=self.show_graphs_window, height=40, font=("Roboto", 14, "bold"))
        self.btn_graphs.pack(pady=10)

    def send_message(self, sender, text):
        msg_data = self.logic_app.send_message(sender, text)
        if msg_data['metrics']:
            self.stats_panel.update_stats(msg_data['metrics'])
        return msg_data

    def show_graphs_window(self):
        self.logic_app.generate_graphs()
        top = ctk.CTkToplevel(self)
        top.title("Вихідні дані аналізу")
        top.geometry("400x150")
        lbl = ctk.CTkLabel(top, text="Графік збережено у 'chat_analysis.png'", font=("Roboto", 16))
        lbl.pack(expand=True)

if __name__ == "__main__":
    app = SecureChatApp()
    app.mainloop()
```

Додаток В. Ілюстративний матеріал

Вінницький національний технічний університет

ВДОСКОНАЛЕННЯ МЕТОДУ ДИНАМІЧНОЇ ГЕНЕРАЦІЇ КЛЮЧІВ ДЛЯ СИМЕТРИЧНОГО ШИФРУВАННЯ

Виконав студент групи 2КІТС-24м:
Тимченко Д. І.
Науковий керівник:
д.ф., доц. Салієва О. В.

Мета роботи:
Підвищення криптографічної стійкості симетричного шифрування шляхом вдосконалення методу динамічної генерації ключів на основі стохастичних процесів.

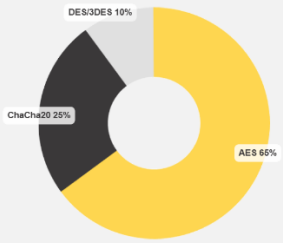
Актуальність:
Динамічна генерація криптографічних ключів підвищує безпеку цифрових комунікацій, запобігає наслідкам компрометації статичних ключів. Використання стохастичних процесів забезпечує непередбачуваність і високу стійкість системи, роблячи підхід перспективним для захисту даних у різних галузях.

ЗАВДАННЯ

Аналіз існуючих методів симетричного шифрування
↓
Розробка методу генерації ключів на основі ланцюгів Маркова
↓
Забезпечення властивості Forward Secrecy для кожної сесії
↓
Програмна реалізація та тестування стійкості (NIST STS)

Аналіз методів симетричного шифрування

- **DES/3DES:** Застарілі алгоритми з коротким ключем (56/168 біт) та низькою швидкодією. Не рекомендуються для нових систем через вразливість до атак повного перебору.
- **AES (Rijndael):** Сучасний стандарт (NIST) з довжиною ключа 128/256 біт. Забезпечує високий рівень безпеки та є галузевим стандартом, але має статичну природу ключів.
- **ChaCha20:** Високопродуктивний потоковий шифр, стійкий до timing-атак, оптимізований для мобільних пристроїв.

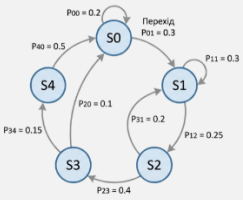


Стохастичні процеси

Ланцюги Маркова

- **Динамічна матриця переходів P(t)**
- **Висока ентропія**
- **Властивість відсутності пам'яті**
- **Захист від криптоаналізу**

Модель Марковського ланцюга (5 станів)

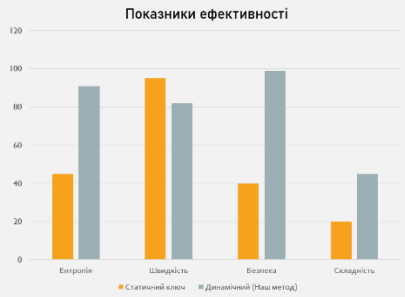


Матриця переходів P(t)

	S0	S1	S2	S3	S4
S0	0.2	0.3	0.2	0.1	0.2
S1	0.1	0.3	0.25	0.1	0.25
S2	0.1	0.1	0.2	0.4	0.2
S3	0.05	0.2	0.15	0.3	0.15
S4	0.5	0.1	0.1	0.1	0.2

Принцип роботи запропонованого методу

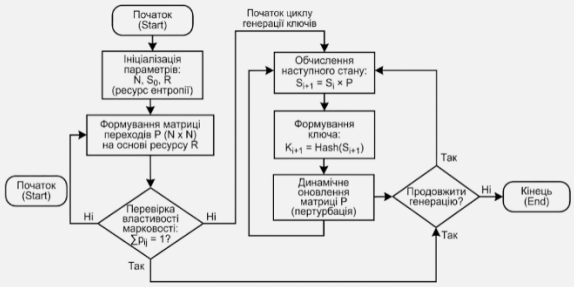
Метод базується на моделюванні випадкових блукань у просторі станів ланцюга Маркова. Динамічна матриця переходів P(t) оновлюється в реальному часі, що забезпечує непередбачуваність кожного наступного ключа та властивість прямої секретності (Forward Secrecy).



Архітектура програмної системи

- Модуль ініціалізації
- Модуль генерації ключів
- Модуль шифрування
- Модуль керування

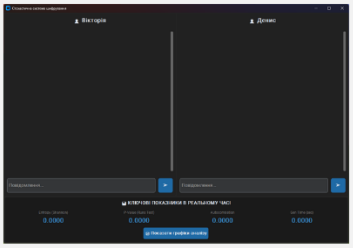
Блок-схема алгоритму роботи програми

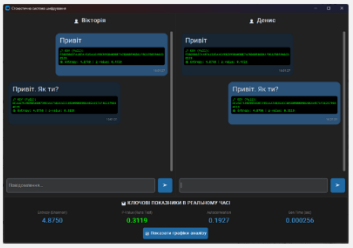


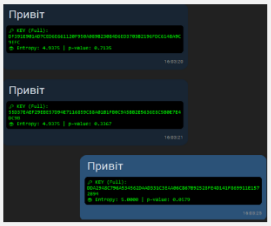
Мова програмування та середовище розробки

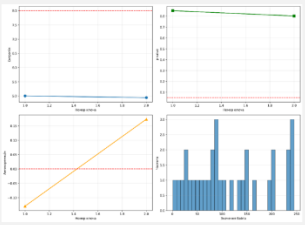
Реалізація програми була здійснена в програмному середовищі Microsoft Visual Studio Code на мові Python

Приклад роботи програми









Економічний аналіз

Впровадження розробленого методу динамічної генерації ключів є економічно доцільним порівняно з придбанням дорогих комерційних рішень захисту інформації.

Чистий приведений дохід
322 447,60 грн.

Внутрішня норма дохідності
33,7%

Дисконтовий період окупності
2,9 роки

Фактичний термін окупності
1,6 року

Результати роботи

ДОСЯГНЕННЯ FORWARD SECRECY
Розроблено вдосконалений метод генерації ключів на основі ланцюгів Маркова, що забезпечує унікальність кожного ключа та неможливість відновлення історії.

ПІДТВЕРДЖЕНА НАДІЙНІСТЬ
Генеровані послідовності успішно пройшли всі базові тести пакету NIST STS (P-значення > 0.01), демонструючи високу ентропію.

ПРИЙНЯТНА ПРОДУКТИВНІСТЬ
Збільшення часу обробки даних становить лише 24.5% порівняно зі статичним шифруванням, що є допустимим для систем реального часу.

ПРАКТИЧНА ЦІННІСТЬ
Метод рекомендовано для впровадження в месенджери,системи банківського обслуговування та комерційний сегмент для захисту критичних даних.

ДЯКУЮ ЗА УВАГУ

Додаток Г. Протокол перевірки на антиплагіат

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Вдосконалення методу динамічної генерації ключів для симетричного шифрування текстових повідомлень на основі стохастичних процесів

Тип роботи: магістерська кваліфікаційна робота
Підрозділ: кафедра менеджменту та безпеки інформаційних систем факультет менеджменту та інформаційної безпеки

гр.2КІТС-24м

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism (КП1) **1,66 %**

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

☐ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак академічного плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту**

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки академічного плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень. Робота до захисту не приймається.

Експертна комісія:

к.т.н., доцент, зав. каф. МБІС Карпінець В.В. _____

к.ф.-м.н., доцент каф. МБІС Шиян А.А. _____

Особа, відповідальна за перевірку Коваль Н.П. _____

З висновком експертної комісії ознайомлений(-на)

Керівник _____ д.ф. Салієва О.В.

Здобувач _____ Тимченко Д.І.