

Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи бакалавра
на тему: «Проект програмного застосунку для реалізації
алгоритму “Camellia” з підтримкою режимів блочного
шифрування та динамічним вибором параметрів»

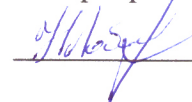
Виконала: студентка 4-го курсу групи КБ-21-1
спеціальності 125 «Кібербезпека»

 Карина ПОЛІЩУК

Керівник: старший викладач кафедри КІ та КБ

 Олександра ПОКОТИЛО

Рецензент: доцент кафедри ІПЗ, к.т.н.

 Надія ЛОБАНЧИКОВА

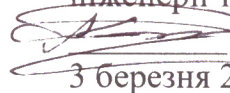
Житомир – 2025

Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки
Спеціальність 125 «Кібербезпека»
Освітня програма «Кібербезпека»

ЗАТВЕРДЖУЮ

Завідувач

кафедри комп'ютерної
інженерії та кібербезпеки

 Андрій Єфіменко
3 березня 2025 р.

ЗАВДАННЯ

на випускню кваліфікаційну роботу бакалавра

Студент: Поліщук Карина Русланівна

Тема роботи: «Проект програмного застосунку для реалізації алгоритму “Camellia” з підтримкою режимів блочного шифрування та динамічним вибором параметрів»

затверджена Наказом університету від 3 березня 2025 р. № 80/с

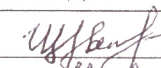
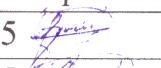
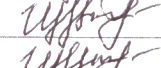
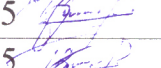
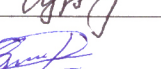
Термін здачі студентом закінченої роботи 10 червня 2025 р.

Вихідні дані роботи (зазначається предмет і об'єкт дослідження):
Об'єктом дослідження є процес розробки ПЗ для реалізації блочного шифрування Camellia. Предметом дослідження є методи та алгоритми блочного шифрування, режими його роботи.

Обладнання та програмне забезпечення:

1. Visual Studio Code
2. Python 3.12.1

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

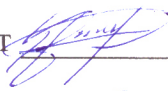
Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Щур Н.О.	14.03.2025 	14.03.2025 
2	Щур Н.О.	07.04.2025 	07.04.2025 
3	Щур Н.О.	27.05.2025 	27.05.2025 

Студент  Карина ПОЛІЩУК

Керівник  Олександра ПОКОТИЛО

Календарний план

№ з/п	Назви етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка завдання	17.03.2025	Виконано
2	Аналіз літературних джерел та існуючих рішень	20.03.2025	Виконано
3	Вивчення алгоритму Camellia та режимів блочного шифрування	24.03.2025	Виконано
4	Вибір програмного середовища та мови програмування	04.04.2025	Виконано
5	Вибір інструментів та розробка структури програмного застосу-нку	08.04.2025	Виконано
6	Реалізація основних модулів шифрування та дешифрування	28.04.2025	Виконано
7	Додавання підтримки різних режимів шифрування та динамічного вибору параметрів	10.05.2025	Виконано
8	Розробка графічного інтерфейсу користувача	14.05.2025	Виконано
9	Оптимізація продуктивності алгоритму Camellia	19.05.2025	Виконано
10	Тестування програмного застосу-нку, виправлення помилок	26.05.2025	Виконано
11	Оформлення пояснювальної записки	29.05.2025	Виконано

Студент  Карина ПОЛІЩУК

Керівник  Олександра ПОКОТИЛО

АНОТАЦІЯ

Кваліфікаційна робота на тему «Проект програмного застосунку для реалізації алгоритму “Camellia” з підтримкою режимів блочного шифрування та динамічним вибором параметрів» складається з переліку умовних скорочень, вступу, трьох розділів, висновків, списку використаної літератури та додатків.

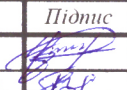
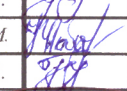
Пояснювальна записка містить 55 рисунків та займає 80 сторінок. Список використаних джерел налічує 20 найменувань і розташований на 3 сторінках.

Метою даної роботи є розробка програмного застосунку, що реалізує симетричний блоковий алгоритм шифрування “Camellia” з можливістю вибору одного з кількох режимів шифрування та динамічним налаштуванням криптографічних параметрів.

В роботі було описано теоретичні основи алгоритму “Camellia”, особливості генерації ключів різної довжини, принцип роботи режимів блочного шифрування (ECB, CBC, CFB, OFB), а також проведено аналіз криптостійкості та порівняння з іншими сучасними алгоритмами шифрування, такими як “AES” та ДСТУ 7624:2014 “Калина”. Представлено архітектуру програмного застосунку, реалізацію його основних функціональних модулів, результати тестування, аналіз продуктивності та перспективи вдосконалення.

У додатках подано лістинг програмної реалізації алгоритму “Camellia”, схеми процесів шифрування та розшифрування, а також графіки оптимізації.

Ключові слова: АЛГОРИТМ ШИФРУВАННЯ, CAMELLIA, СИМЕТРИЧНИЙ ШИФР, БЛОКОВИЙ ШИФРУВАННЯ.

					ККІК.КР.Б – 125 – 16 – ПЗ				
Змн.	Арк.	№ докум.	Підпис	Дата	Розробка програмного застосунку для реалізації алгоритму “Camellia” з підтримкою блочного шифрування та динамічним вибором параметрів	Літ.	Арк.	Аркуші	
Розроб.		Поліщук К.Р.		12.06.25					
Керівник		Покотило О.А.		12.06.25			4	98	
Реценз.		Лобанчикова Н.М.		12.06.25		ФІКТ, Гр. КБ-21-1			
Н. контр.		Покотило О.А.		12.06.25					
Затверд.		Єфіменко А.А.		12.06.25					

ANNOTATION

The qualification work titled “Project of a software application for the implementation of the Camellia algorithm with support for block encryption modes and dynamic parameter selection” consists of a list of abbreviations, an introduction, three chapters, conclusions, a list of references and appendices.

The explanatory note includes 55 figures and spans 80 pages. The list of references comprises 20 sources and extends over 3 pages.

This work aims to develop a software application that implements the Camellia symmetric block cipher algorithm, providing support for multiple encryption modes and enabling the dynamic configuration of cryptographic parameters.

The document explores the theoretical foundations of the Camellia encryption algorithm, detailing the process of key generation for various key lengths and the operational principles of common block cipher modes, including ECB, CBC, CFB and OFB. It further examines the algorithm’s cryptographic strength and provides a comparative analysis with other modern encryption standards such as AES and DSTU 7624:2014 ‘Kalyna’. Additionally, the document presents an architecture of the software application, the implementation of its core functional modules, the results of testing, performance evaluations and potential directions for future enhancement.

The attachment includes the source code of the software implementation of the Camellia algorithm, diagrams illustrating the encryption and decryption processes, as well as graphs demonstrating optimisation results.

Keywords: ENCRYPTION ALGORITHM, CAMELLIA, SYMMETRIC CIPHER, BLOCK CIPHER.

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				5
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

ВСТУП.....	8
РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ АЛГОРИТМУ SAMELLIA ТА РЕЖИМИ БЛОЧНОГО ШИФРУВАННЯ.....	10
1.1 Огляд сучасних алгоритмів шифрування та місце Samellia серед них	10
1.2 Опис алгоритму Samellia	12
1.3 Аналіз криптостійкості та можливі атаки	24
1.4 Порівняння алгоритму Samellia з AES, Kalyna та іншими	30
РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕН- НЯ	34
2.1 Функціональні можливості програми та вимоги до бібліотеки.....	34
2.2 Архітектура та структура бібліотеки.....	36
2.3 Обґрунтування вибору середовища програмування.....	37
2.4 Реалізація основних функцій	38
2.5 Аналіз продуктивності та адаптивне налаштування параметрів шифрування	48
РОЗДІЛ 3. ТЕСТУВАННЯ ТА АНАЛІЗ РОБОТИ ПРОГРАМНОГО ЗАСТОСУН- КУ.....	50
3.1 Тестування програми.....	50
3.2 Аналіз продуктивності шифрування в різних режимах	68
3.3 Порівняння продуктивності Samellia з AES та Kalyna.....	71
3.4 Перспективи покращення алгоритму та його застосування.....	74
ВИСНОВКИ	77
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	78
ДОДАТКИ	81

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ECB – Electronic Codebook (Режим електронної кодової книги)

CBC – Cipher Block Chaining (Режим зчеплення блоків шифру)

CFB – Cipher Feedback Mode (Режим зворотного зв'язку шифру)

OFB – Output Feedback Mode (Режим вихідного зворотного зв'язку)

S-Box – Substitution Box (Таблиця підстановок)

XOR – Exclusive OR (Операція виключного "АБО")

PKCS7 – Public-Key Cryptography Standards №7 (Стандарт заповнення блоків)

AES – Advanced Encryption Standard (Стандарт розширеного шифрування)

Feistel Network – Мережа Фейстеля (Структура блочного шифру)

IV – Initialization Vector (Ініціалізаційний вектор)

FL / FL⁻¹ – Функції логічної трансформації

KL, KR, KA, KB – Частини ключів і проміжні ключові змінні

kwt, ku, klv – Підключі, що застосовуються на етапах шифрування і розшифрування

GUI – Graphical User Interface (Графічний інтерфейс користувача)

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				7
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми. Забезпечення захисту інформації в умовах активного використання цифрових технологій та глобальної інформатизації є пріоритетним напрямом у сфері кібербезпеки. З огляду на високі ризики несанкціонованого доступу, спотворення чи втрати даних, криптографічні методи – зокрема алгоритми симетричного блочного шифрування – виступають ключовим інструментом підтримання конфіденційності, цілісності та автентичності інформації.

З-поміж сучасних алгоритмів симетричного шифрування, окрім загально-визнаного AES, все більшої уваги набуває алгоритм Camellia, розроблений консорціумом Mitsubishi Electric та NTT. Цей алгоритм вирізняється високою крипто-стійкістю, підтримкою ключів різної довжини (128, 192, 256 біт), ефективною реалізацією як у програмному, так і в апаратному середовищі, а також відповідністю міжнародним стандартам (ISO/IEC, IETF), що дає йому конкурентну перевагу серед альтернатив.

Порівняно з іншими алгоритмами, такими як AES чи національним українським стандартом ДСТУ 7624:2014 «Калина», Camellia демонструє конкурентоспроможну продуктивність, гнучкість та високу стійкість до відомих типів криптоатак. Водночас вітчизняна практика його застосування є обмеженою, що створює науково-практичну зацікавленість у дослідженні його структури, принципів роботи, режимів шифрування та реалізації в межах програмного забезпечення.

З огляду на вищезазначене, актуальним є завдання розробки програмного застосунку, що реалізує алгоритм Camellia із підтримкою стандартних режимів блочного шифрування та можливістю динамічного налаштування параметрів, зокрема типу вхідних даних, довжини ключа та обраного режиму роботи. Такий застосунок може бути використаний як навчальний інструмент для ознайомлення з принципами симетричної криптографії, а також як базовий засіб захисту конфіденційної інформації в локальних системах.

Метою кваліфікаційної роботи є розробка програмного застосунку, що реалізує алгоритм симетричного блочного шифрування Camellia, з підтримкою режимів ECB,

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				8
Змн.	Арк.	№ докум.	Підпис	Дата		

CBC, CFB та OFB, а також з можливістю обробки різних типів даних і динамічного вибору криптографічних параметрів, зокрема довжини ключа.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- провести аналіз сучасних алгоритмів симетричного шифрування та визначити переваги й недоліки Camellia порівняно з аналогами;
- дослідити структуру алгоритму Camellia, механізм генерації ключів та специфіку перетворень на кожному етапі шифрування;
- реалізувати підтримку основних режимів блочного шифрування в контексті алгоритму Camellia;
- здійснити програмну реалізацію функціональних модулів із підтримкою вибору режиму, введення ключа, обробки файлів різного типу;
- проаналізувати ефективність реалізованого застосунку з аналогічними алгоритмами такими як AES і Kalyna.

Об’єктом дослідження є процеси симетричного блочного шифрування інформації, які використовуються для забезпечення криптографічного захисту даних у програмних системах.

Предметом дослідження є алгоритм Camellia та програмні засоби його реалізації, а також методи організації режимів блочного шифрування, генерації ключів та обробки даних у межах прикладного програмного забезпечення.

У процесі виконання кваліфікаційної роботи було розроблено проект програмного застосунку, що забезпечує шифрування та розшифрування даних з використанням алгоритму Camellia. Застосунок підтримує обробку текстових та графічних файлів, дозволяє користувачу обирати режим шифрування та довжину ключа, а також виконує криптографічні перетворення відповідно до заданих параметрів. До складу системи також включено модулі тестування та аналізу продуктивності, що дозволяє оцінити ефективність використання алгоритму Camellia порівняно з іншими сучасними симетричними алгоритмами шифрування.

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				9
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ АЛГОРИТМУ CAMELLIA ТА РЕЖИМИ БЛОЧНОГО ШИФРУВАННЯ

1.1 Огляд сучасних алгоритмів шифрування та місце Camellia серед них

Алгоритм шифрування є основою будь-якого SSL-сертифіката, захищаючи передачу даних між серверами та клієнтами шляхом шифрування інформації, зберігаючи при цьому авторизований доступ. Важливість надійного шифрування стала надзвичайно важливою під час пандемії COVID-19.

Глобальні витрати на кібербезпеку мали зрости до 215 мільярдів доларів у 2024 році, тоді як кіберзлочини зросли на 300% під час пандемії, оскільки віддалені робочі системи виявили вразливість. Нездатність працівників захистити дані організації робить системи вразливими до хакерських атак, особливо ті, що використовують застарілі стандарти безпеки, які піддаються складним загрозам [1].

Шифрування даних є важливою складовою для організацій, що працюють на онлайн-платформах. Безпека даних значною мірою залежить від шифрування, оскільки воно захищає конфіденційну інформацію від несанкціонованого доступу.

Симетричне та асиметричне шифрування складають два основні типи алгоритмів шифрування. Симетричне шифрування використовує один ключ як для шифрування, так і для дешифрування даних. Цей метод є швидшим та ефективнішим, підходить для шифрування великих обсягів даних. Асиметричне шифрування, з іншого боку, використовує пару ключів – відкритий та закритий, і гарантує, що навіть якщо ключ шифрування є відкритим, лише відповідний закритий ключ може розшифрувати дані. Інтеграція цих алгоритмів у цифрові системи допомагає підтримувати конфіденційність та цілісність інформації під час її переміщення через потенційно незахищений простір Інтернету або зберігання в цифровому вигляді [2].

Впровадження надійних алгоритмів шифрування має вирішальне значення для захисту цифрового зв'язку та зберігання даних. З розвитком кіберзагроз розвиваються й більш просунуті алгоритми, такі як Advanced Encryption Standard (AES) та Rivest-Shamir-Adleman (RSA). Обидва широко визнані за свою силу та надійність

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		10

у захисті конфіденційних даних. На практиці шифрування забезпечує безпечні транзакції, конфіденційність комунікацій та захист конфіденційності користувачів на широкому спектрі пристроїв і платформ, що підкреслює його значення в сучасну цифрову епоху [3].

Сучасні комп'ютери використовують RSA для кодування та декодування інформації. Це асиметричний алгоритм, який працює з відкритим та закритим ключами. Його також називають криптографією з відкритим ключем. У безпеці RSA відкритий ключ відомий усім, тоді як закритий ключ зберігається в таємниці для декодування інформації. Коли будь-яке повідомлення шифрується відкритим ключем, закритий ключ розшифровує лише це повідомлення. Якщо закритий ключ скомпрометовано, весь принцип шифрування порушується [4].

Хакерам знадобиться висока обчислювальна потужність і значна кількість часу, щоб зламати RSA-шифрування. Стійкість шифрування залежить від розміру ключа. Коли розмір ключа подвоюється, стійкість шифрування збільшується. Ключ RSA може мати розмір 2048 або 3072 біти [3]. Раніше використовувався 1024-бітний ключ, але якимось чином він виявився вразливим до атак.

Не менш важливим у практиці інформаційного захисту є AES (Advanced Encryption Standard) – симетричний блоковий шифр, який став стандартом урядів і організацій у всьому світі. Прийнятий як заміна застарілого DES, AES вирізняється високою швидкістю, надійністю та ефективністю реалізації як у програмному, так і в апаратному середовищі [5]. Він підтримує ключі довжиною 128, 192 та 256 біт, що дозволяє обирати оптимальний баланс між продуктивністю та рівнем захисту. AES активно використовується в захисті Wi-Fi (WPA2/WPA3), VPN, мобільних пристроях, хмарних платформах, а також у шифруванні жорстких дисків і баз даних. Його криптостійкість базується на сильній математичній структурі та ефективному механізмі підстановок і перестановок, що робить його практично несприйнятливим до сучасних типів атак [5].

Серед інших сучасних алгоритмів, які відіграють важливу роль у забезпеченні безпеки, варто згадати ChaCha20, що є потоковим симетричним шифром із підвищеною стійкістю до атак типу timing attack, і тому став популярним

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				11
Змн.	Арк.	№ докум.	Підпис	Дата		

у мобільних додатках та VPN-сервісах [6]. ECC (еліптична криптографія) забезпечує високий рівень захисту при значно менших розмірах ключів, що робить її ефективною для пристроїв з обмеженими ресурсами. Поряд із цим, вчені й фахівці з безпеки активно працюють над створенням постквантових алгоритмів шифрування, які мають протистояти загрозам з боку майбутніх квантових комп'ютерів [7].

У цьому контексті Camellia займає помітне місце серед сучасних алгоритмів симетричного шифрування. Розроблений японськими компаніями Mitsubishi Electric і NTT, цей блоковий шифр має 128-бітну довжину блоку та підтримує ключі довжиною 128, 192 і 256 біт – так само як AES. Camellia був стандартизований міжнародними організаціями, зокрема ISO/IEC, і рекомендований до використання в таких протоколах, як TLS, IPsec, OpenPGP. Він був розроблений з урахуванням високої продуктивності як у програмних реалізаціях, так і в апаратних середовищах, що робить його придатним для широкого спектра застосувань – від вбудованих систем до серверних рішень [8].

Camellia має структуру, схожу на структуру AES, проте відрізняється внутрішньою організацією раундів шифрування та ключових перетворень, що забезпечує високу криптостійкість проти основних типів атак, таких як лінійний та диференційний криптоаналіз [9]. У той же час, Camellia демонструє швидкість, порівнянну з AES, і часто перевершує його в умовах обмежених обчислювальних ресурсів, що сприяє його поширенню в країнах Азії, а також у міжнародних стандартах [10].

З огляду на широкі можливості використання та високу стійкість до криптоаналітичних атак, алгоритм Camellia заслуговує на детальніший розгляд. У подальшому буде проаналізовано його структуру, принципи функціонування та особливості реалізації.

1.2 Опис алгоритму Camellia

Camellia використовує 18-раундову структуру Фейстеля для 128-бітових ключів і 24-раундову структуру Фейстеля для 192- і 256-бітових ключів, з

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		12

додатковими рівнями перетворення і логічними функціями, які називаються FL-функцією і FL⁻¹-функцією, що вставляються кожні 6 раундів. На рисунку 1.3 показано огляд шифрування з використанням 128-бітових ключів. Елемент з суфіксом_(n) показує, що елемент має довжину n-біт.

Розклад клавіш генерує 64-розрядні підклавіші kw_t ($t = 1, 2, 3, 4$) для вибілювання вводу/виводу, k_u ($u = 1, 2, \dots, r$) для круглих функцій та kl_v ($v = 1, 2, \dots, r/3-2$)

Camellia: 128-бітний блоковий шифр, придатний для різних платформ для FL- та FL⁻¹-функцій від секретного ключа K, де r – кількість раундів.

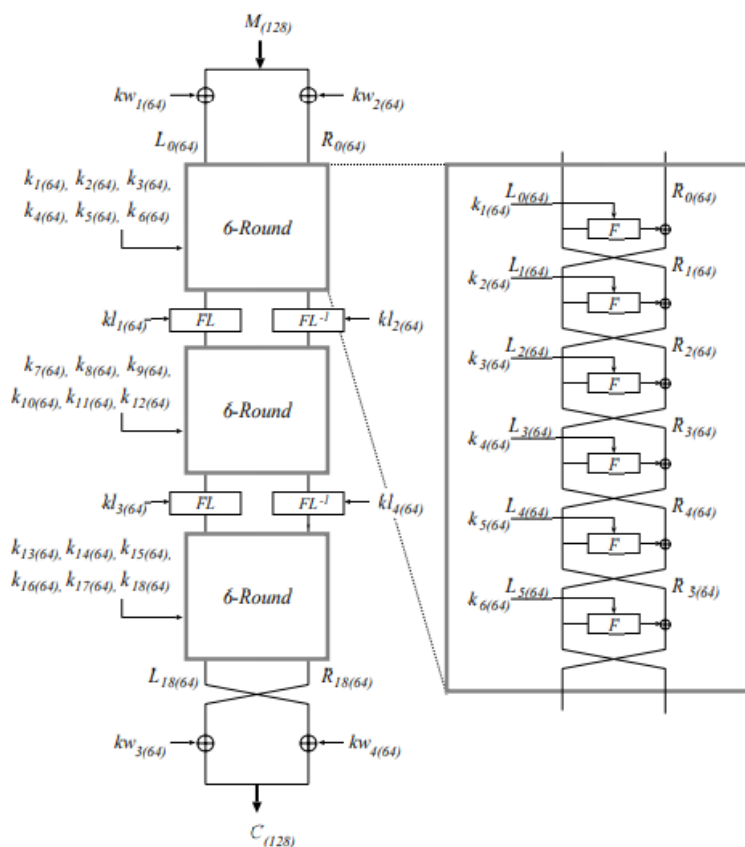


Рис.1.1 Процедура шифрування Camellia для 128-бітових ключів

Шифрування для 128-бітних ключів

Спочатку 128-бітовий відкритий текст M об'єднується за допомогою XOR з $kw_1 || kw_2$ і розділяється на два 64-бітних даних L_0 і R_0 , тобто $M \oplus (kw_1 || kw_2) = L_0 || R_0$. Далі виконуються наступні операції виконуються від $i = 1$ до 18, крім $i = 6$ і 12;

$$L_i = R_{i-1} \oplus F(L_{i-1}, k_i),$$

$$R_i = L_{i-1}$$

Для $i = 6$ і 12 виконано наступне:

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				13
Змн.	Арк.	№ докум.	Підпис	Дата		

$$L'_i = R_{i-1} \oplus F(L_{i-1}, k_i),$$

$$R'_i = L_{i-1},$$

$$L_i = FL\left(L'_i, kl_{i-1}^{\frac{1}{3}-1}\right),$$

$$R_i = FL^{-1}\left(R'_i, kl_{i-1}^{\frac{1}{3}}\right)$$

Нарешті, R_{18} і L_{18} конкатенуються і об'єднуються за схемою XOR з $kw_3 || kw_4$.
Отримане значення є 128-бітовим зашифрованим текстом, тобто $C = (R_{18} || L_{18}) \oplus (kw_3 || kw_4)$.

Шифрування для 192- та 256-бітних ключів

Подібно до шифрування 128-бітовими ключами, спочатку 128-бітний відкритий текст M об'єднується XOR з $kw_1 || kw_2$ і розділяється на два 64-бітних даних L_0 і R_0 , тобто, $M \oplus (kw_1 || kw_2) = L_0 || R_0$. Далі виконуються наступні операції від $r = 1$ до 24, за винятком для $r = 6, 12$ та 18 ;

$$L_r = R_{r-1} \oplus F(L_{r-1}, k_r),$$

$$R_r = L_{r-1}$$

Для $r = 6, 12$ та 18 виконується наступне:

$$L'_r = R_{r-1} \oplus F(L_{r-1}, k_r),$$

$$R'_r = L_{r-1},$$

$$L_r = FL\left(L'_r, kl_r^{\frac{1}{3}-1}\right),$$

$$R_r = FL^{-1}\left(R'_r, kl_r^{\frac{1}{3}}\right)$$

Нарешті, R_{24} і L_{24} конкатенуються і об'єднуються за схемою XOR з $kw_3 || kw_4$.
В результаті є 128-бітовим зашифрованим текстом, тобто $C = (R_{24} || L_{24}) \oplus (kw_3 || kw_4)$.

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				14
Змн.	Арк.	№ докум.	Підпис	Дата		

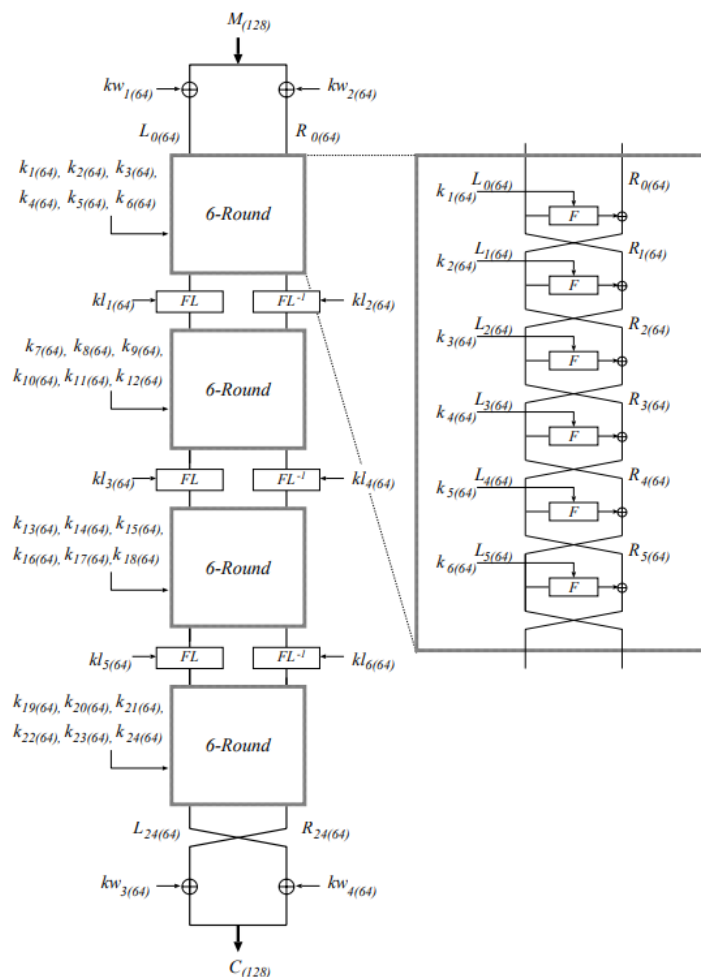


Рис.1.2 Процедура шифрування Camellia для 192 та 256-бітових ключів
Розшифрування

Процедура дешифрування Camellia може бути виконана так само, як і процедура шифрування, шляхом зміни порядку підключів, що є однією з переваг мереж Фейстеля. У Camellia шари з функціями FL/FL⁻¹ вставляються кожні 6 раундів, але ця властивість все одно зберігається.

На рисунку 1.3 показано процедуру дешифрування для 128-бітного ключа. Частина рандомізації даних має 18-раундову структуру Фейстеля з двома шарами функцій FL/FL⁻¹ після 6-го та 12-го раундів, а також 128-бітові операції XOR до першого раунду та після останнього. Частина генерації ключів створює підключі $kw_t^{(64)}$ (де $t = 1, 2, 3, 4$), $k_u^{(64)}$ (де $u = 1, 2, \dots, 18$) та $kl_v^{(64)}$ (де $v = 1, 2, 3, 4$) і секретного ключа К;

У частині рандомізації даних спочатку шифротекст $C_{(128)}$ XOR-иться з $kw_3^{(64)}$ || $kw_4^{(64)}$ і розділяється на $R_{18}^{(64)}$ та $L_{18}^{(64)}$ однакової довжини, тобто:

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				15
Змн.	Арк.	№ докум.	Підпис	Дата		

$$C_{(128)} \oplus (k\omega_3^{(64)} \parallel k\omega_4^{(64)}) = R_{18}^{(64)} \parallel L_{18}^{(64)}$$

Далі виконуються такі операції для $r = 18$ до 1, за винятком $r = 13$ і $r = 7$:

$$R_{r-1} = L_r \oplus F(R_r, k_r),$$

$$L_{r-1} = R_r$$

Для $r = 13$ та $r = 7$ виконується наступне:

$$R'_{r-1} = L_r \oplus F(R_r, k_r),$$

$$L'_{r-1} = R_r,$$

$$R_{r-1} = FL\left(R'_{r-1}, kl_{\frac{2(r-1)}{6}}\right),$$

$$L_{r-1} = FL^{-1}\left(L'_{r-1}, kl_{\frac{2(r-1)}{6}-1}\right)$$

Нарешті, $L_0^{(64)}$ та $R_0^{(64)}$ об'єднуються та XOR-яться з $k\omega_1^{(64)} \parallel k\omega_2^{(64)}$.

Результат є відкритим текстом, тобто:

$$M_{(128)} = (L_0^{(64)} \parallel R_0^{(64)}) \oplus (k\omega_1^{(64)} \parallel k\omega_2^{(64)})$$

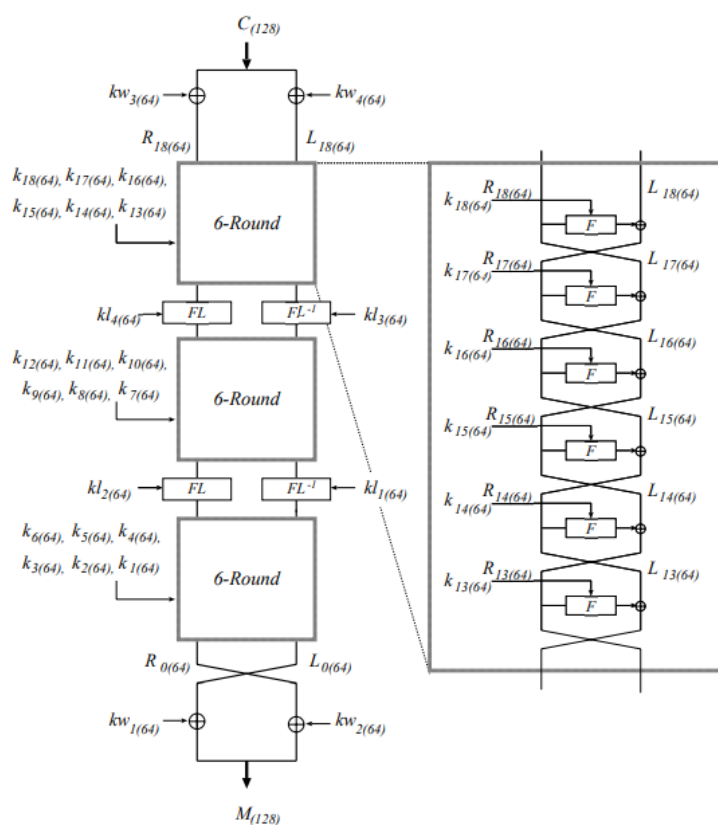


Рис.1.3. Процедура дешифрування Samellia для 128-бітових ключів

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				16
Змн.	Арк.	№ докум.	Підпис	Дата		

На рисунку 1.4 показано процедуру дешифрування для 192 та 256-бітного ключа. Частина рандомізації даних має 24-раундову структуру Фейстеля з трьома шарами функцій FL/FL^{-1} після 6-го, 12-го та 18-го раундів, а також 128-бітові операції XOR до першого раунду та після останнього. Частина генерації ключів створює підключі $k\omega_t^{(64)}$ (де $t = 1, 2, 3, 4$), $k_u^{(64)}$ (де $u = 1, 2, \dots, 24$) та $kl_v^{(64)}$ (де $v = 1, 2, 3, 4, 5, 6$) і секретного ключа K ;

У частині рандомізації даних спочатку шифротекст $C_{(128)}$ XOR-иться з $k\omega_3^{(64)}$ $\parallel k\omega_4^{(64)}$ і розділяється на $R_{24}^{(64)}$ та $L_{24}^{(64)}$ однакової довжини, тобто:

$$C_{(128)} \oplus (k\omega_3^{(64)} \parallel k\omega_4^{(64)}) = R_{24}^{(64)} \parallel L_{24}^{(64)}$$

Далі виконуються такі операції для $r = 24$ до 1, за винятком $r = 19, 13$ і 7:

$$R_{r-1} = L_r \oplus F(R_r, k_r),$$

$$L_{r-1} = R_r$$

Для $r = 19, 13$ та 7 виконується наступне:

$$R'_{r-1} = L_r \oplus F(R_r, k_r),$$

$$L'_{r-1} = R_r,$$

$$R_{r-1} = FL\left(R'_{r-1}, kl_{\frac{2(r-1)}{6}}\right),$$

$$L_{r-1} = FL^{-1}\left(L'_{r-1}, kl_{\frac{2(r-1)}{6}-1}\right)$$

Нарешті, $L_0^{(64)}$ та $R_0^{(64)}$ об'єднуються та XOR-ються з $k\omega_1^{(64)} \parallel k\omega_2^{(64)}$.

Результат є відкритим текстом, тобто:

$$M_{(128)} = (L_0^{(64)} \parallel R_0^{(64)}) \oplus (k\omega_1^{(64)} \parallel k\omega_2^{(64)})$$

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		17

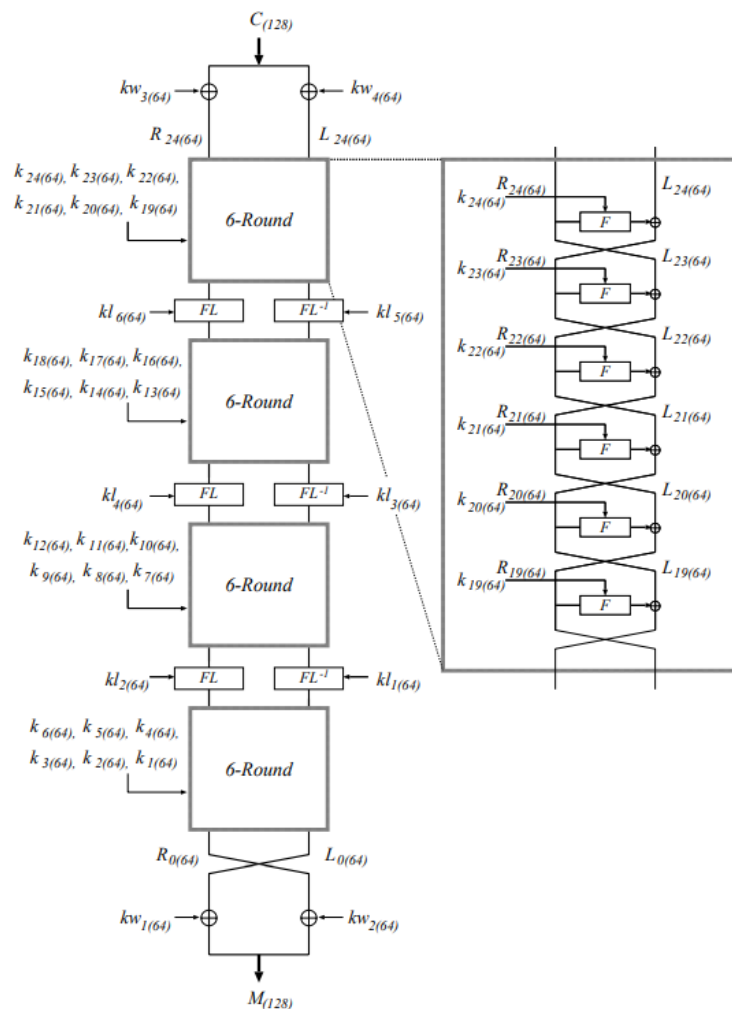


Рис.1.4. Процедура дешифрування Camellia для 128-бітових ключів

Опис генерації ключів у Camellia:

На рисунку 1.5 показано розклад ключів Camellia. Дві 128-бітові змінні K_L та K_R визначаються наступним чином. Для 128-бітових ключів як K_L використовується 128-бітовий ключ K , а K_R дорівнює 0. Для 192-бітових ключів як K_L використовується лівий 128-біт ключа K , а як K_R - конкатенація правого 64-бітового ключа K і доповнення правого 64-бітового ключа K . Для 256-бітових ключів лівий 128-біт ключа K використовується як K_L , а правий 128-біт K - як K_R .

Дві 128-бітові змінні K_A і K_B генеруються з K_L і K_R , як показано на рис.1.5. Зауважте, що K_B використовується лише у випадку, якщо довжина секретного ключа становить 192 або 256 біт. 64-розрядні константи Σ_i ($i = 1, 2, \dots, 6$) використовуються як «ключі» в мережі Фейстеля.

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				18
Змн.	Арк.	№ докум.	Підпис	Дата		

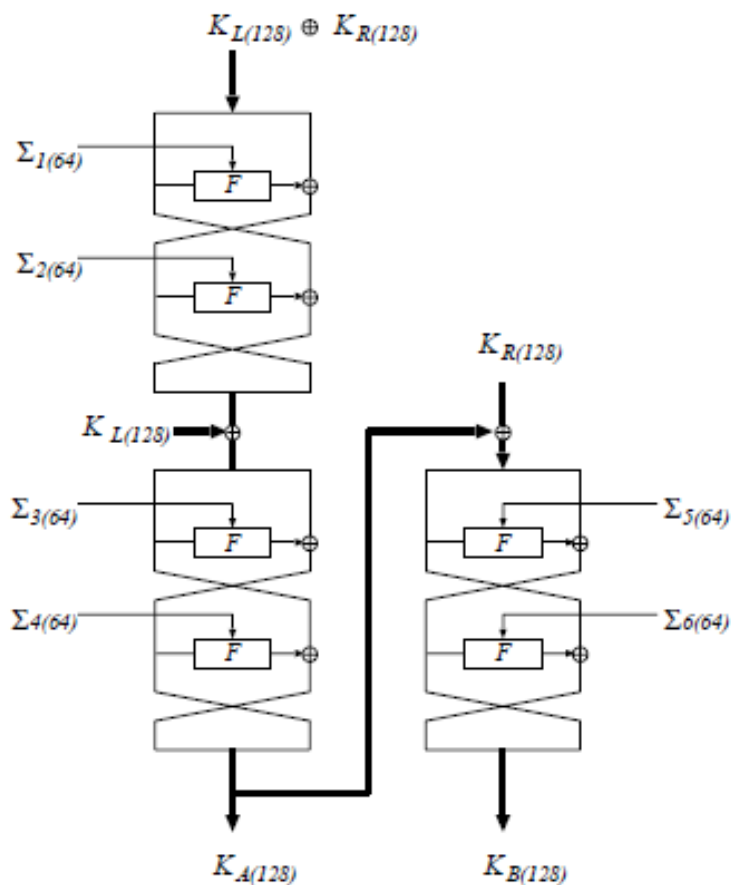


Рис.1.5 Ключовий розклад

Вони визначаються як неперервні значення від другого до сімнадцятого розряду шістнадцяткового представлення квадратного кореня з i -го простого числа. Ці константні значення наведено у таблиці 1.1.

Таблиця 1.1

Ключові константи розкладу

$\Sigma_i^{(64)}$	Константа
$\Sigma_1^{(64)}$	0xA09E667F3BCC908B
$\Sigma_2^{(64)}$	0xB67AE8584CAA73B2
$\Sigma_3^{(64)}$	0xC6EF372FE94F82BE
$\Sigma_4^{(64)}$	0x54FF53A5F1D36F1C
$\Sigma_5^{(64)}$	0x10E527FADE682D1D
$\Sigma_6^{(64)}$	0xB05688C2B3E6C1FD

64-розрядні підключі kw_t , k_u та kl_v генеруються з K_L , K_R , K_A та K_B . Підключі генеруються обертанням K_L , K_R , K_A та K_B і взяттям їх лівої або правої половини. Деталі наведено у рисунку 1.6. [11]

128-bit keys	subkey	value	192/256-bit keys	subkey	value
Prewhitening	kw_1	$(K_L \ll 0)_L$	Prewhitening	kw_1	$(K_L \ll 0)_L$
	kw_2	$(K_L \ll 0)_R$		kw_2	$(K_L \ll 0)_R$
F (Round1)	k_1	$(K_A \ll 0)_L$	F (Round1)	k_1	$(K_B \ll 0)_L$
F (Round2)	k_2	$(K_A \ll 0)_R$	F (Round2)	k_2	$(K_B \ll 0)_R$
F (Round3)	k_3	$(K_L \ll 15)_L$	F (Round3)	k_3	$(K_R \ll 15)_L$
F (Round4)	k_4	$(K_L \ll 15)_R$	F (Round4)	k_4	$(K_R \ll 15)_R$
F (Round5)	k_5	$(K_A \ll 15)_L$	F (Round5)	k_5	$(K_A \ll 15)_L$
F (Round6)	k_6	$(K_A \ll 15)_R$	F (Round6)	k_6	$(K_A \ll 15)_R$
FL	kl_1	$(K_A \ll 30)_L$	FL	kl_1	$(K_R \ll 30)_L$
FL^{-1}	kl_2	$(K_A \ll 30)_R$	FL^{-1}	kl_2	$(K_R \ll 30)_R$
F (Round7)	k_7	$(K_L \ll 45)_L$	F (Round7)	k_7	$(K_B \ll 30)_L$
F (Round8)	k_8	$(K_L \ll 45)_R$	F (Round8)	k_8	$(K_B \ll 30)_R$
F (Round9)	k_9	$(K_A \ll 45)_L$	F (Round9)	k_9	$(K_L \ll 45)_L$
F (Round10)	k_{10}	$(K_L \ll 60)_R$	F (Round10)	k_{10}	$(K_L \ll 45)_R$
F (Round11)	k_{11}	$(K_A \ll 60)_L$	F (Round11)	k_{11}	$(K_A \ll 45)_L$
F (Round12)	k_{12}	$(K_A \ll 60)_R$	F (Round12)	k_{12}	$(K_A \ll 45)_R$
FL	kl_3	$(K_L \ll 77)_L$	FL	kl_3	$(K_L \ll 60)_L$
FL^{-1}	kl_4	$(K_L \ll 77)_R$	FL^{-1}	kl_4	$(K_L \ll 60)_R$
F (Round13)	k_{13}	$(K_L \ll 94)_L$	F (Round13)	k_{13}	$(K_R \ll 60)_L$
F (Round14)	k_{14}	$(K_L \ll 94)_R$	F (Round14)	k_{14}	$(K_R \ll 60)_R$
F (Round15)	k_{15}	$(K_A \ll 94)_L$	F (Round15)	k_{15}	$(K_B \ll 60)_L$
F (Round16)	k_{16}	$(K_A \ll 94)_R$	F (Round16)	k_{16}	$(K_B \ll 60)_R$
F (Round17)	k_{17}	$(K_L \ll 111)_L$	F (Round17)	k_{17}	$(K_L \ll 77)_L$
F (Round18)	k_{18}	$(K_L \ll 111)_R$	F (Round18)	k_{18}	$(K_L \ll 77)_R$
Postwhitening	kw_3	$(K_A \ll 111)_L$	FL	kl_5	$(K_A \ll 77)_L$
	kw_4	$(K_A \ll 111)_R$	FL^{-1}	kl_6	$(K_A \ll 77)_R$
			F (Round19)	k_{19}	$(K_R \ll 94)_L$
			F (Round20)	k_{20}	$(K_R \ll 94)_R$
			F (Round21)	k_{21}	$(K_A \ll 94)_L$
			F (Round22)	k_{22}	$(K_A \ll 94)_R$
			F (Round23)	k_{23}	$(K_L \ll 111)_L$
			F (Round24)	k_{24}	$(K_L \ll 111)_R$
			Postwhitening	kw_3	$(K_B \ll 111)_L$
				kw_4	$(K_B \ll 111)_R$

Рис.1.6 Підключі для 128-бітових ключів та 192/256-бітових ключів

Режими роботи блочного шифру :

Режими роботи блочного шифру визначають, як надійно шифрувати та дешифрувати великі обсяги даних за допомогою блочного шифру. Блоковий шифр – це алгоритм шифрування, який обробляє дані блоками фіксованого розміру (наприклад, 128 біт), а не по одному біту за раз. Однак для шифрування даних, розмір яких перевищує один блок, використовуються різні режими роботи, щоб забезпечити безпеку та ефективність. Ось кілька поширених режимів. Ось кілька поширених режимів:

Електронна кодова книга (ECB)

Електронна кодова книга є найпростішим режимом функціонування блокового шифру. Це простіше через пряме шифрування кожного блоку вхідного

відкритого тексту, а вихід у формі блоків зашифрованого шифротексту. Як правило, якщо розмір повідомлення перевищує b біт, його можна розбити на групу блоків і процедуру повторити.

Процедура ECB проілюстрована нижче:

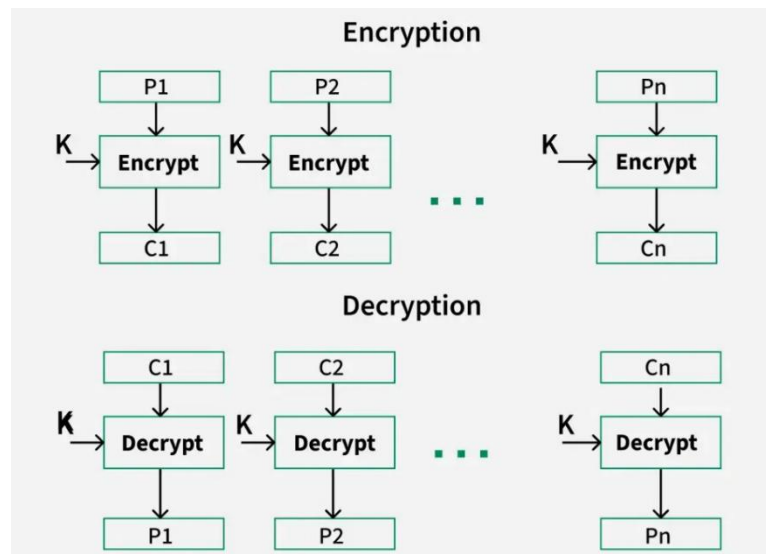


Рис.1.7 Схема блочного шифрування в режимі ECB

Переваги використання ECB

- Можливе паралельне шифрування блоків бітів, тому це швидший спосіб шифрування.

- Простий спосіб блокового шифру.

Недоліки використання ECB

- Схильний до криптоаналізу, оскільки між відкритим і зашифрованим текстом існує прямий зв'язок.

- Ідентичні блоки відкритого тексту створюють ідентичні блоки зашифрованого тексту, які можуть виявити шаблони.

З'єднання блоків шифру (CBC)

Зв'язування блоків шифру або CBC є вдосконаленням ECB, оскільки ECB компрометує деякі вимоги безпеки. У CBC попередній блок шифрування надається як вхідні дані для наступного алгоритму шифрування після XOR з вихідним блоком відкритого тексту. У двох словах, блок шифру створюється шляхом шифрування результату XOR попереднього блоку шифру та поточного блоку відкритого тексту.

Переваги CBC

- CBC добре працює для вхідних даних більше b біт.
- CBC є хорошим механізмом автентифікації.
- Кращий опір криптоаналізу, ніж ECB.
- Більш безпечний, ніж ECB, оскільки він приховує шаблони.

Недоліки CBC

- Для шифрування та дешифрування потрібен попередній блок зашифрованого тексту, що ускладнює паралельну обробку.

Процес проілюстровано тут:

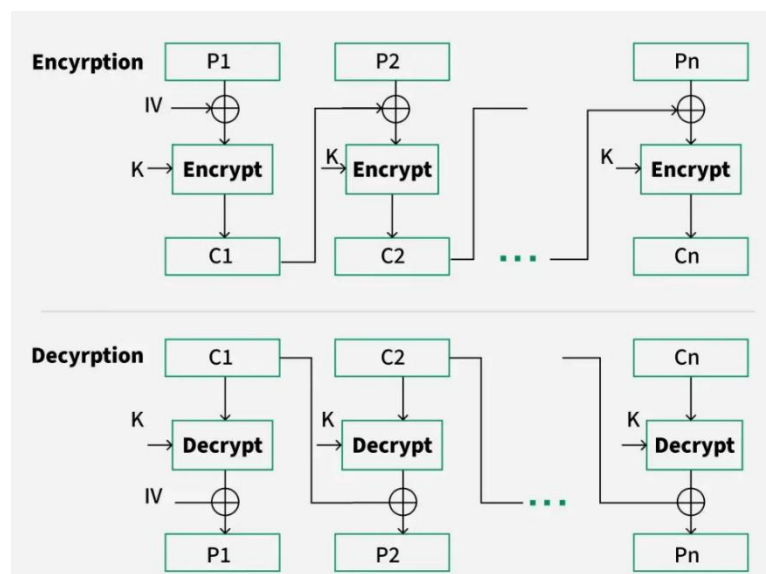


Рис.1.8 Схема блочного шифрування в режимі CBC

Режим зворотного зв'язку шифру (CFB)

У цьому режимі шифр передається як зворотний зв'язок до наступного блоку шифрування з деякими новими специфікаціями: спочатку початковий вектор IV використовується для першого шифрування, а вихідні біти поділяються як набір бітів s та bs . Біти лівої сторони s вибираються разом із бітами відкритого тексту, до яких застосовується операція XOR. Результат передається як вхідний сигнал у зсувний регістр, який має bs бітів у lhs , s бітів у rhs , і процес продовжується.

Переваги CFB

- Оскільки через використання зсувного регістра відбувається деяка втрата даних, тому застосовувати криптоаналіз важко.
- Може обробляти потоки даних будь-якого розміру.

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		22

Недоліки використання CFB

- Недоліки CFB такі ж, як і режиму CBC. Як втрати блоків, так і одночасне шифрування кількох блоків не підтримуються шифруванням. Однак розшифровка є розпаралелюваною та стійкою до втрат.

- Трохи складніший і може поширювати помилки.

Нижче наведено процес шифрування та дешифрування для одного і того ж, обидва використовують алгоритми шифрування.

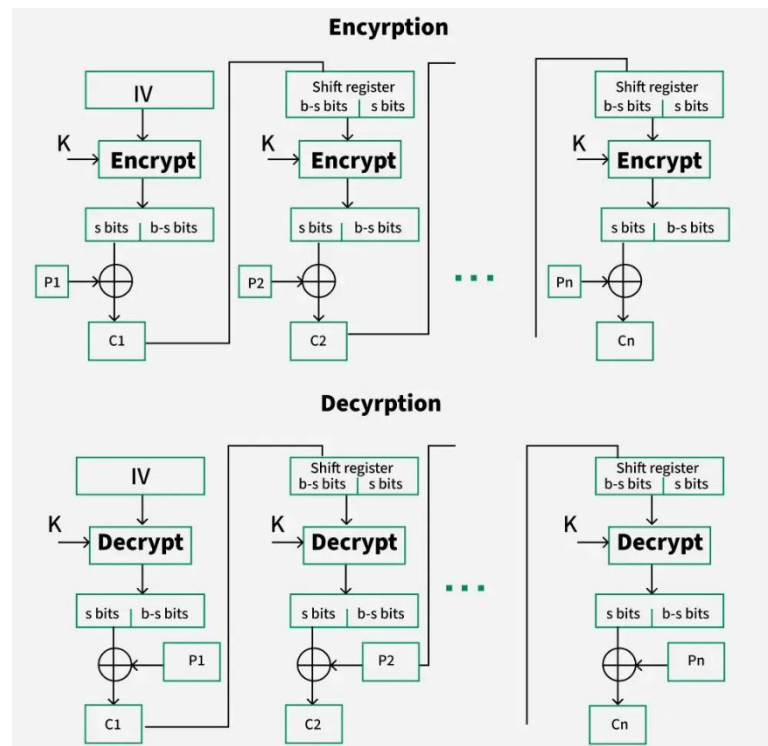


Рис.1.9 Схема блочного шифрування в режимі CFB

Режим вихідного зворотного зв'язку (OFB)

Режим вихідного зворотного зв'язку відбувається майже так само, як і режим зворотного зв'язку шифру, за винятком того, що він надсилає зашифрований вихід як зворотний зв'язок замість фактичного шифру, який є виводом XOR. У цьому режимі вихідного зворотного зв'язку всі біти блоку надсилаються замість надси- лання вибраних s бітів. Режим вихідного зворотного зв'язку блокового шифру за- безпечує велику стійкість до помилок передачі бітів. Це також зменшує залежність або зв'язок шифру з відкритим текстом.

Переваги OFB

- У випадку CFB помилка одного біта в блоці поширюється на всі наступні блоки. Цю проблему вирішує OFB, оскільки він не містить бітових помилок у блоці відкритого тексту. Таким чином, помилки в передачі не поширюються.

Недоліки OFB

- Недоліком OFB є те, що через режими роботи він більш сприйнятливий до атаки модифікації потоку повідомлень, ніж CFB.

- Якщо потік ключів використовується повторно, безпека буде порушена. [12]

Процедура OFB проілюстрована нижче:

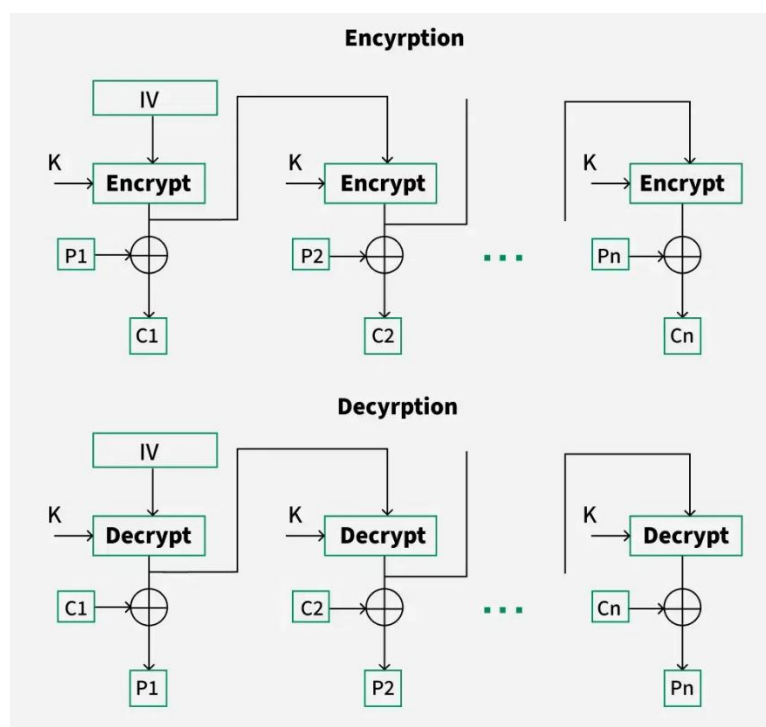


Рис.1.10 Схема блочного шифрування в режимі OFB

1.3 Аналіз криптостійкості та можливі атаки

Диференціальні атаки є найпоширенішою формою криптоаналізу блокових шифрів. Загальноприйнятим підходом до проєктування байт-орієнтованого шифру, стійкого до диференціальних атак, є забезпечення того, щоб кожна диференціальна характеристика мала певну кількість активних S-боксів. Окрім диференціальних властивостей самих S-боквів, ця кількість також залежить від розміру стану в сценарії з одним ключем та від розміру ключа в сценарії зі спорідненим ключем.

Далі наведено ймовірності найкращих диференціальних траєкторій залежно від кількості активних S-блоків.

Одноклавішні диференціали

У сценарії з одним ключем ми припускаємо, що ключ не відрізняється, а у відкритому тексті є деяка початкова різниця. Для знаходження ймовірностей та кількості активних S-блоків у найкращих характеристиках з одним ключем ми використовуємо вдосконалений підхід грубого перебору, заснований на методі Мацуї, який використовується для знаходження найкращих характеристик в DES. Ми використовуємо термін «найкращий» для характеристик з найвищою ймовірністю. Як згадувалося раніше, Camellia використовує чотири різних типи S-блоків: максимальна диференціальна ймовірність кожного з них становить 2^{-6} .

Ми використовуємо ці ймовірності як оцінки для активних S-блоків. Найкращі характеристики одного ключа з точки зору мінімальної кількості активних S-блоків наведено в таблиці 1.2.

Таблиця 1.2

Кількість активних S-блоків у диференціальних характеристиках Camellia

Раунди	Активні S-блоки
1	0
2	1
3	2
4	7
5	9
6	11
7	13
8	15
9	18
10	21
11	22
12	25
13	26
14	30
15	32
16	34

Оскільки ймовірність кожного активного S-боксу дорівнює 2^{-6} , а розмір блоку 128 біт, то для того, щоб бути достовірною, характеристика може мати не більше $[128/6] = 21$ активний S-бокс. Враховуючи таблицю, ми можемо легко визначити верхню межу для найкращих диференціальних атак, заснованих на нові диференціальних характеристик:

Спостереження 2. Для всіх розмірів ключа Camellia жодна диференціальна характеристика з одним ключем, що охоплює понад 10 раундів, не має ймовірність вищу за 2^{-128} .

Теорія обчислення диференціальних характеристик значно випереджає можливість обчислення ймовірностей самих диференціалів. Тому не можна надати точну межу кількості раундів, достатньої для стійкості Camellia до диференціальних атак. Однак за стандартних припущень, що кількість характеристик у межах одного диференціалу є малою, а також враховуючи відносно великий запас стійкості для найкращої характеристики на понад 10 раундах – очікується, що Camellia є стійким до диференціальних атак з одним ключем.

Диференціали пов'язаних ключів

Застосування непобайтових обертових зсувів у процесі генерації підключів унеможлиблює проведення ефективного аналізу стійкості шифру Camellia до атак із пов'язаними ключами. Попередній аналіз був виконаний для модифікованої версії Camellia, у якій усі зсуви було приведено до кратних 8. Результати показали відсутність ефективних атак цього типу для зазначеної версії шифру. Хоча адаптація алгоритму пошуку для застосування до оригінальної версії Camellia є теоретично можливою, на практиці вона залишається нездійсненною через надмірну обчислювальну складність.

Основною перешкодою є велика кількість можливих конфігурацій активних байтів у підключах навіть при фіксації активних байтів у ключах K_L , K_R , K_A , K_B . Наприклад, якщо в ключі K_L активним є лише найменш значущий байт, тоді при зсуві K_L на 15 бітів ($k_3 = K_L \ll 15$) можливе активування лише другого байта (за відсутності відмінностей у молодшому біті активного байта), лише першого байта

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		26

(якщо відмінність наявна тільки в молодшому біті) або одночасно першого та другого байтів (у разі повної відмінності у всіх бітах активного байта). Таким чином, для одного активного байта виникає три варіанти, тобто розгалуження дорівнює трьом. Відповідно, загальне розгалуження зростає експоненційно зі збільшенням кількості активних байтів у підключі. З урахуванням того, що в Camellia-128 використовується 18 підключів, загальна кількість можливих варіантів відмінностей у підключах при одній зміні в K_L , K_R , K_A або K_B робить аналіз практично нездійсненним.

Оцінити, чи сприяють обертання підключів підвищенню стійкості шифру до атак типу диференціалів пов'язаних ключів, неможливо, оскільки, на відміну від байторієнтованої версії, повний пошук в оригінальній реалізації є непридатним для реалізації на практиці. Водночас, значне розгалуження активних байтів у підключах теоретично збільшує кількість можливих диференціальних трас пов'язаних ключів. Наприклад, у випадку з одним активним байтом у K_L , у байторієнтованій версії відмінність у підключі k_3 фіксована та міститься у другому байті. Натомість в оригінальній версії відмінність може бути локалізована у першому, другому або в обох байтах одночасно. Відповідно, при спробі скасування активних байтів зростає кількість можливих комбінацій, що дозволяє нейтралізувати різницю в різних положеннях підключу. Водночас, точне визначення відмінностей активних байтів у ключах і підключах для заданого шляху атаки не завжди є можливим.

Зазначена раніше властивість доповнення також може трактуватися як атака типу диференціалів пов'язаних ключів. Проте, з огляду на низьку ймовірність виникнення відповідного диференціала в процесі генерації підключів, вона не становить загрози криптографічній стійкості шифру Camellia в контексті атак із пов'язаними ключами.

Бумеранговий криптоаналіз

У бумерангових атаках обрані диференціальні характеристики не обов'язково повинні охоплювати повну кількість раундів шифру. Кількість раундів для побудови атаки обирається таким чином, щоб забезпечити максимальну ймовірність успішного проведення атаки бумерангом. Оскільки ймовірності

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		27

найкращих характеристик для скороченої кількості раундів вже відомі, можливо обчислити загальну ймовірність (а відповідно і складність) найефективніших бумерангових атак.

Для визначення найкращої атаки типу "бумеранг" з одним ключем слід враховувати результати, наведені у таблиці 1.2. У випадку, коли верхня частина характеристики охоплює 3 раунди, а нижня – 4 раунди, тобто загалом 7 раундів, ймовірність бумерангової атаки може перевищувати 2^{-128} (за умови, що всі активні S-блоки забезпечують максимальну ймовірність 2^{-128}). Для будь-якої іншої комбінації раундів, сумарна кількість яких перевищує 7, ймовірність атаки буде меншою за 2^{-128} . Таким чином, можна зробити висновок, що для 8 раундів атака бумерангом є неефективною. Хоча використання розширених методів може дозволити зловмиснику пропускати раунди на початку, у середині або в кінці структури атаки, загальний запас криптографічної міцності залишається значним. Це дозволяє стверджувати, що приблизно 11 раундів шифру Camellia є стійкими до атак бумерангом.

Розгляд випадку так званих підсилених бумерангів не дає зловмиснику переваги порівняно з класичним підходом бумерангового криптоаналізу.

Усічений диференціальний криптоаналіз

У межах усічених диференціальних атак дослідник не простежує конкретні значення різниці, що поширюється через раунди шифру, а аналізує лише положення байтів, у яких виникає різниця (тобто активні байти). У результаті, саме лінійні перетворення в шифрі мають вирішальний вплив на ймовірність реалізації характеристики.

Для знаходження найефективніших усічених диференціалів зі скороченою кількістю раундів було застосовано підхід Мацуї у поєднанні зі стандартними припущеннями:

1. S-блоки не впливають на ймовірність – активний байт не може стати неактивним і навпаки;
2. Операція XOR може скасувати два активні байти з ймовірністю 2^{-8} ;

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		28

3. Матричне множення здатне сформувати вихідний стовпець з t активними байтами з ймовірністю $2^{-8(8-t)}$, за винятком випадку, коли на вході є лише один активний байт – тоді ймовірність становить 1.

Був реалізований метод повного перебору в просторі всіх можливих усічених диференціалів, результати якого наведені в таблиці 1.3. На основі отриманих даних сформульовано наступне спостереження:

Спостереження 3. Для шифру Camellia не існує усічених диференціалів, що охоплюють більше ніж 7 раундів.

Варто зазначити, що усічені диференціали, представлені в таблиці, мають точно визначені позиції активних байтів. У разі пом'якшення цих обмежень потенційно можливо побудувати диференціали для більшої кількості раундів. Однак очікується, що навіть за таких умов кількість охоплених раундів навряд чи перевищуватиме 7 значною мірою.

Беручи до уваги значний запас криптостійкості шифру Camellia, можна зробити висновок, що повнораундна версія шифру є стійкою до усічених диференціальних атак.

Таблиця 1.3

Ймовірності усічених диференціалів Camellia

Раунди	Ймовірності $-\log_2$
1	0
2	24
3	48
4	64
5	80
6	104
7	120
8	136

Атаки t -type slide застосовуються до шифрів із подібною структурою раундів. У випадку з Camellia це неактуально через особливості генерації раундових ключів – кожен раунд у структурі Фейстеля використовує різну 64-бітну константу.

Таким чином, Camellia є стійка до атаки типу slide як зі сторони одного ключа, так і зі спорідненими ключами.

Ротаційні атаки на поточному етапі розвитку криптоаналізу були застосовані виключно до криптографічних примітивів, що базуються на операціях addition-rotation-XOR. Для можливості реалізації таких атак щодо шифрів із байторієнтованою структурою підстановок та перестановок необхідно, щоб ротаційні вхідні пари відрізнялися на величину, кратну 8. Незважаючи на відповідність Camellia цій умові, в його схемі розкладання ключа застосовується значна кількість раундових констант, які не є ротаційно-симетричними. У зв'язку з цим ротаційні атаки не можуть бути ефективно адаптовані до шифру Camellia. [13]

1.4 Порівняння алгоритму Camellia з AES, Kalyna та іншими

Розширений стандарт шифрування AES та Camellia – це два широко використовувані алгоритми із симетричним ключем блоку с у галузі криптографії. Хоча обидва алгоритми забезпечують безпечні послуги шифрування та дешифрування, вони відрізняються своїми характеристиками продуктивності.

AES – це широко прийнятий стандарт шифрування, розроблений Національним інститутом стандартів і технологій (NIST) у 2001 році. Він підтримує розміри ключів 128, 192 та 256 бітів і працює зі 128 блоками. AES відомий своєю високою продуктивністю та широко використовується в різних сферах застосування, включаючи шифрування даних, протоколи безпечного зв'язку та хмарне сховище.

З іншого боку, Camellia – це блоковий шифр, розроблений японською компанією NTT у 2000 році. Він працює зі 128-бітними блоками та підтримує розміри ключів 128, 192 та 256 бітів. Camellia розроблена для швидкої та безпечної роботи, що робить її популярним вибором для різних застосувань, включаючи протоколи безпечного зв'язку та шифрування даних [14].

Kalyna – це український національний стандарт блочного шифрування, розроблений для заміни застарілого ГОСТ 28147-89. Kalyna підтримує більший діапазон розмірів блоків і ключів - 128, 256 та 512 біт, а кількість раундів

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		30

змінюється залежно від цих параметрів (від 10 до 18 раундів). Алгоритм базується на SP-мережі, але має низку унікальних особливостей, таких як використання чотирьох різних S-блоків замість одного, а також складніший механізм генерації підключів. Це підвищує стійкість Kalyna до сучасних видів криптоаналізу, зокрема алгебраїчного, лінійного та диференційного [15].

Щодо продуктивності, Kalyna демонструє високу швидкодію, яка в деяких випадках перевищує показники AES. Наприклад, при використанні 128-бітного ключа Kalyna працює приблизно на 3% швидше за AES, а при 256-бітному ключі - дещо повільніше або порівняно з AES залежно від розміру блоку [16]. Camellia, у свою чергу, зазвичай показує продуктивність, трохи нижчу за AES і Kalyna, але все ще достатньо високу для більшості практичних застосувань [17].

Щодо стійкості, Kalyna має перевагу завдяки більшій кількості раундів і складнішій структурі, що забезпечує додатковий захист від різних типів атак [18]. Camellia також вважається надійним і пройшла численні експертизи, однак її структура є більш класичною [19]. AES, незважаючи на свою поширеність, залишається дуже стійким алгоритмом, особливо завдяки широкій підтримці апаратних прискорень, що дозволяє ефективно застосовувати його в різних середовищах.

Додатково варто згадати алгоритм SEED, розроблений Корейським агентством інформаційної безпеки (KISA) у 1998 році. Це симетричний блочний шифр на основі мережі Фейстеля з 16 раундами, 128-бітними блоками і 128-бітним ключем. Він використовує дві 8×8 таблиці підстановки, а структура раундової функції містить рекурсивні елементи, що нагадують алгоритм MISTY1. SEED широко застосовується у Південній Кореї, зокрема у фінансовому секторі та урядових протоколах TLS і S/MIME, оскільки забезпечує достатній рівень безпеки, який перевищує застарілі 40-бітні шифри. Проте SEED менш поширений за межами Південної Кореї і не підтримується більшістю сучасних браузерів та інтернет-додатків [20].

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		31

Порівняльна характеристика блочних шифрів

Критерії	AES	Camellia	Kalyna	SEED
Розробник	NIST	NTT	ДССЗІ України	KISA
Рік розробки	2001	2000	2014	1998
Розмір блоку, біт	128	128	128, 256, 512	128
Розмір ключа, біт	128, 192, 256	128, 192, 256	128, 256, 512	128
Кількість раундів	10, 12, 14	18, 24, 32 (залежить від ключа)	10-18 (залежить від блоку та ключа)	16
Структура	SP-мережа	SP-мережа	SP-мережа з 4 S-блоками	Мережа Фейстеля з MISTY1 подібною функцією
Продуктив- ність	Висока, апаратне прискорення	Трохи нижча за AES	Вища за AES (128 біт), нижча або рівна (256 біт)	Достатня, але нижча за AES і неапаратно прискорена
Стійкість до атак	Дуже висока, ретельно перевірена	Висока, пройшла експертизу	Дуже висока, з урахуванням сучасних методів атак	Висока, але менш складна структура
Сфера застосування	Універсальна, міжнародні стандарти	Протоколи безпеки, сховище	Національні стандарти України, захист крити- чних систем	В основному в Південній Кореї, TLS, S/MIME
Глобальне поширення	Дуже широке	Помірне	Обмежене (переважно в Україні)	Локальне (Південна Корея)

Висновки до першого розділу

У цьому розділі було проаналізовано теоретичні основи алгоритму Camellia, його структуру, принципи роботи та режими блочного шифрування. Показано, що Camellia, як і AES, є потужним симетричним алгоритмом, здатним забезпечити високий рівень захисту даних. Порівнюючи з AES і Kalyna, дозволило виявити сильні та слабкі сторони кожного з алгоритмів. Зокрема, AES вирізняється підтримкою апаратного прискорення, Kalyna – складнішою структурою та високою стійкістю, тоді як Camellia демонструє хорошу продуктивність і гнучкість реалізації.

Також підтверджено високу стійкість Camellia до основних типів криптоаналітичних атак, що робить його надійним кандидатом для використання у практичних криптографічних системах, зокрема в рамках реалізації програмного застосунку, який є об'єктом цього дослідження.

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		33

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Функціональні можливості програми та вимоги до бібліотеки

У бакалаврській дипломній роботі потрібно розробити програму, яка виконує шифрування та розшифрування даних на основі перестановок байтів. Програмний застосунок повинен мати вигляд Windows-додатку, реалізованого з використанням стандартних елементів керування, а також забезпечувати зручний графічний інтерфейс користувача.

Основною метою створення цього програмного продукту є надання користувачеві інструменту, який дозволяє швидко та інтуїтивно зрозуміло здійснювати операції шифрування і дешифрування інформації, не заглиблюючись у технічні деталі реалізованого алгоритму.

Програма має виконувати низку функціональних завдань, серед яких:

- введення та виведення даних за допомогою текстових полів;
- генерація або введення ключа перестановки;
- реалізація процесів шифрування та розшифрування;
- збереження результатів роботи у файл;
- відкриття файлів для обробки;
- повідомлення про помилки у разі некоректних дій користувача.

Інтерфейс програми має бути розроблений відповідно до вимог зручності використання та візуальної привабливості. Передбачено використання таких елементів:

- Текстові поля – для взаємодії з вхідними та вихідними даними;
- Кнопки – для запуску основних функцій: шифрування, розшифрування, відкриття, збереження;
- Діалогові вікна – для вибору файлів та збереження результатів;
- Вікна повідомлень – для інформування користувача про успішне виконання операцій або виникнення помилок.

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		34

Опис основних модулів програмного забезпечення

Назва модуля	Основні функції	Призначення
camellia.py (модуль)	- Клас CamelliaCipherApp (графічний інтерфейс) - Вибір режимів шифрування - Введення/виведення тексту - Шифрування/дешифрування файлів - Взаємодія з буфером, збереження результатів	Основний інтерфейсний модуль. Забезпечує повноцінну роботу додатку з користувачем
Camellia CipherApp (клас)	Реалізує усі взаємодії GUI, вибір файлів, режимів, шифрування, дешифрування	Ядро програми, зв'язок між інтерфейсом і криптомодулями
encrypt_by_mode()	Універсальне шифрування залежно від вибраного режиму (ECB, CBC, CFB, OFB)	Спрощення логіки виклику потрібного алгоритму
decrypt_by_mode_for_file()	Універсальне дешифрування файлів з урахуванням режиму та типу даних	Забезпечує правильне оброблення тексту, аудіо, зображень
encrypt_file(), encrypt_image(), encrypt_audio()	Функції шифрування конкретних типів даних	Дає можливість шифрувати різні типи файлів у відповідному режимі
decrypt_file(), decrypt_image(), decrypt_audio()	Дешифрування із розпізнаванням формату вихідного файлу	Відновлення файлів після шифрування
library_camellia.py (модуль)	- Алгоритм Camellia: camellia_encrypt_block, camellia_decrypt_block - Підтримка режимів: camellia_ecb_encrypt, cbc_encrypt, тощо - Паддінг, S-блоки, FL/FL ⁻¹ - key_schedule()	Криптографічне ядро програми. Реалізація блочного шифрування згідно стандарту
get_random_bytes()	Генерує криптографічно стійкі випадкові байти для ключів і IV	Безпечна генерація ключових параметрів

Опис основних модулів програмного забезпечення

Назва модуля	Основні функції	Призначення
pad_data() / unpad_data()	Реалізація доповнення даних згідно з PKCS7	Забезпечення відповідності розміру блоку для шифрування
application_schedules.py (модуль)	Клас ApplicationSchedules: вимірювання часу шифрування, побудова графіків	Модуль для аналізу продуктивності додатку та візуалізації результатів
_execute_benchmark()	Запускає шифрування файлів у різних режимах або порівняльні тести	Збір статистики для різних режимів шифрування та типів файлів
_plot_modes(), _plot_by_type(), _plot_comparison()	Побудова діаграм продуктивності	Візуальне представлення результатів бенчмаркінгу

2.2 Архітектура та структура бібліотеки

Архітектура бібліотеки має реалізувати цілісну та функціонально завершену модель програмного модуля для виконання блочного симетричного шифрування згідно з алгоритмом Camellia. Бібліотека повинна забезпечувати не лише коректність криптографічних перетворень, а й гнучкість у виборі параметрів, зокрема довжини ключа та режиму роботи, а також можливість обробки вхідних даних різного типу.

Передбачається, що центральним елементом архітектури є криптографічне ядро, яке має відповідати специфікаціям алгоритму Camellia, з підтримкою ключів довжиною 128, 192 та 256 біт. Це ядро повинне реалізовувати основну раундову структуру на основі мережі Фейстеля, включаючи використання S-блоків, FL- та FL^{-1} -функцій, а також операцій XOR, обертань і перестановок. У складі цього

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		36

модуля має бути передбачений механізм генерації раундових підключів відповідно до стандартного ключового розкладу.

Окремий функціональний блок повинен забезпечувати підтримку різних режимів блочного шифрування, зокрема ECB, CBC, CFB та OFB. Для режимів, що потребують ініціалізаційного вектора, архітектура має включати генератор IV та відповідну логіку його обробки. Також необхідною є реалізація механізму доповнення блоків згідно з PKCS7, що дозволяє коректно працювати з даними довільної довжини.

Бібліотека повинна містити засоби попередньої обробки даних, що надходять на шифрування. Зокрема, передбачено можливість зчитування тексту з файлів формату TXT, DOCX, PDF, а також обробки зображень та аудіофайлів у форматах PNG, JPG, WAV. Усі дані мають перетворюватися у відповідну байтову форму для подальшої обробки в криптографічному ядрі.

Загальна структура бібліотеки має бути модульною, що дає змогу підтримувати її розширення, супровід та повторне використання. Передбачається, що кожен модуль відповідає за окремий етап обробки – від підготовки вхідних даних до формування зашифрованого результату або його дешифрування. Такий підхід дозволяє зберігати логічну цілісність системи, забезпечує її стабільність, адаптивність та відповідність сучасним вимогам до інформаційної безпеки.

Структурна схема реалізації бібліотеки представлена в [додатку Д](#). Вона демонструє взаємозв'язки між основними класами та модулями (CamelliaCipherApp, ApplicationSchedules, library_camellia.py) та ілюструє логіку взаємодії між графічним інтерфейсом, криптографічним ядром і функціями шифрування файлів. Також у додатку наведено блок-схеми основних алгоритмів шифрування й дешифрування, що реалізовані у відповідних модулях.

2.3 Обґрунтування вибору середовища програмування

Розробка програмного засобу шифрування та дешифрування даних буде проводитися за допомогою мови програмування Python у середовищі Visual Studio Code.

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		37

Мова Python обирається з огляду на її сучасну роль у сфері прикладної криптографії та високий рівень підтримки з боку відкритого програмного співтовариства. Вона відзначається простим і логічним синтаксисом, що сприятиме швидкому написанню та тестуванню окремих частин програми. Особливістю Python є динамічна типізація та інтерактивне середовище розробки, які дозволяють ефективно проводити відлагодження програмного коду вже на етапі розробки.

Python підтримується широким набором бібліотек для роботи з графікою, файлами, текстовими та мультимедійними форматами, а також для побудови графічного інтерфейсу. Це дозволяє уникнути надмірної реалізації базового функціоналу вручну та зосередитися на розробці прикладної логіки застосунку. Вбудовані механізми обробки винятків та модульна структура сприятимуть створенню надійного та стабільного коду, який легко масштабувати та підтримувати.

Visual Studio Code як середовище розробки буде використовуватися завдяки своїй зручності, кросплатформеності та розширюваності. Підтримка численних розширень для Python, таких як інтегрований аналізатор помилок, підсвічування синтаксису, автодоповнення, а також вбудований термінал – дозволяє ефективно писати, запускати та тестувати код без необхідності перемикання між додатковими інструментами. Окрім того, можливість інтеграції з системами контролю версій Git забезпечує організовану роботу над проектом на всіх етапах його створення.

Оскільки розробка буде охоплювати не лише алгоритмічну частину, а й побудову графічного інтерфейсу, взаємодію з файлами, візуалізацію та збереження результатів, Python у поєднанні з Visual Studio Code є оптимальним вибором. Такий підхід дозволяє реалізувати поставлені завдання в єдиному середовищі без потреби у сторонньому інструментарії, що відповідає вимогам до ефективності, доступності та підтримуваності розробки навчального програмного забезпечення.

2.4 Реалізація основних функцій

Введення даних:

Етап введення даних є одним із ключових у реалізації процесу шифрування та дешифрування, оскільки саме на цьому рівні визначається джерело інформації,

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		38

що підлягатиме криптографічному перетворенню. Програмний застосунок повинен забезпечувати можливість зчитування вхідних даних з різних джерел, підтримувати кілька форматів файлів, а також надавати користувачеві зручний спосіб введення тексту безпосередньо через графічний інтерфейс.

Передбачається два основні способи введення даних: ручне введення через текстову область та імпорт даних із зовнішніх файлів. Для реалізації першого способу використовується компонент Text, який входить до стандартного інструментарію бібліотеки tkinter. Даний елемент дозволяє користувачеві вводити або редагувати текстові дані безпосередньо в програмному вікні. Для зручності реалізовано вертикальну прокрутку, яка забезпечує доступ до повного обсягу введеної інформації. Створення області для введення виглядає наступним чином:

```
self.input_text = tk.Text(self.input_text_frame, height=8, width=60, bg="#7E8A97", fg="#E3E9F0",  
insertbackground="black", font=("Arial", 10), yscrollcommand=self.input_text_scroll.set)|
```

Окрім ручного введення, користувач може швидко вставити текст із буфера обміну, що реалізовано функцією paste_from_clipboard(). Такий механізм є зручним у випадках, коли дані копіюються з інших джерел, наприклад, із документа чи вебсторінки:

```
def paste_from_clipboard(self):  
    try:  
        clipboard_text = self.root.clipboard_get()  
        self.input_text.delete("1.0", tk.END)  
        self.input_text.insert(tk.END, clipboard_text)  
    except tk.TclError:  
        messagebox.showwarning("Увага", "У буфері обміну немає тексту.")
```

Другим способом є імпорт файлів через діалогове вікно вибору, яке реалізовано за допомогою модуля filedialog. Користувач має змогу вибрати текстовий або мультимедійний файл, який буде оброблятися на байтовому рівні. У межах застосунку реалізовано підтримку таких форматів: .txt, .docx, .pdf, .png, .jpg, .mp3, .wav. Залежно від типу вибраного файлу застосовується відповідний метод обробки.

У випадку з файлами .txt дані зчитуються як бінарна інформація без додаткової обробки:

```
with open(filepath, "rb") as file:  
    data = file.read()
```

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		39

Для документів формату .docx використовується бібліотека python-docx, яка дозволяє зчитувати текст абзаців:

```
doc = Document(filepath)
data = "\n".join([para.text for para in doc.paragraphs]).encode()
```

У разі роботи з PDF-файлами застосовується бібліотека PyPDF2, що дає змогу витягнути текст з кожної сторінки документа:

```
reader = PyPDF2.PdfReader(file)
data = "\n".join([page.extract_text() for page in reader.pages if page.extract_text()]).encode()
```

Усі зчитані дані незалежно від джерела конвертуються у байтову форму, необхідну для подальших криптографічних операцій.

Окрему увагу варто приділити забезпеченню стійкості до помилок під час зчитування вхідних даних. У програмі передбачено повідомлення про помилки у випадку несумісного формату, відсутності доступу до файлу або його пошкодження. Це реалізовано через повідомлення messagebox, які інформують користувача про тип помилки та пропонують альтернативні дії.

Вибір параметрів шифрування:

Оскільки алгоритм Camellia підтримує декілька режимів шифрування, а також змінну довжину ключа, важливим етапом є формування параметрів криптографічного перетворення. Програмний застосунок повинен забезпечувати можливість гнучкого вибору конфігурації шифрування, що включає як довжину ключа, так і режим роботи алгоритму. Вибрані параметри визначають не лише рівень криптостійкості, але й спосіб обробки вхідних блоків даних.

У графічному інтерфейсі реалізовано випадаючі списки (Combobox), за допомогою яких користувач може обрати бажану довжину ключа – 128, 192 або 256 біт. Кожне з цих значень відповідає одному з трьох дозволених варіантів згідно з офіційною специфікацією алгоритму Camellia. Для зберігання обраного значення використовується змінна key_size_var, прив'язана до елемента інтерфейсу:

```
self.key_size_var = tk.StringVar(value="128")
self.key_size_menu = ttk.Combobox(self.frame, textvariable=self.key_size_var, values=["128", "192", "256"],
state="readonly", width=5)
```

Після вибору довжини ключа програма автоматично генерує криптографічний ключ відповідної довжини у байтах. Для цього використовується

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		40

функція `os.urandom()`, яка генерує криптографічно безпечну випадкову послідовність:

```
def generate_key(self):
    try:
        key_size = int(self.key_size_var.get())
        if key_size not in (128, 192, 256):
            raise ValueError("Неправильна довжина ключа")
        return os.urandom(key_size // 8)
    except Exception:
        messagebox.showerror("Помилка", "Неправильна довжина ключа! Виберіть 128, 192 або 256.")
    return None
```

У процесі дешифрування користувач має змогу самостійно ввести ключ у шістнадцятковому представленні. Для цього реалізовано вікно з полем введення, яке з'являється після вибору файлу:

```
key_hex = simpledialog.askstring("Введіть ключ", "Введіть шістнадцятковий ключ (32/48/64 символи):")
if not key_hex:
    return
try:
    key = binascii.unhexlify(key_hex)
except binascii.Error:
    messagebox.showerror("Помилка", "Неправильний формат ключа!")
return
```

Не менш важливою складовою є вибір режиму шифрування. У межах застосунку реалізовано підтримку чотирьох стандартних режимів: ECB (Electronic Codebook), CBC (Cipher Block Chaining), CFB (Cipher Feedback) та OFB (Output Feedback). Кожен із них має свою специфіку та використовується в різних криптографічних сценаріях.

- ECB – найпростіший режим, де кожен блок шифрується незалежно. Незважаючи на простоту, цей режим вважається найменш безпечним через збереження повторюваних шаблонів у шифротексті.
- CBC – використовує зворотний зв'язок між блоками. Перший блок поєднується з випадковим вектором ініціалізації (IV), кожен наступний з попереднім зашифрованим блоком.
- CFB і OFB – працюють як потокові шифри та забезпечують високу стійкість до атак у разі обробки даних байт за байтом.

Вибір режиму також реалізовано через Combobox:

```
self.mode_var = tk.StringVar(value="ECB")
self.mode_menu = ttk.Combobox(self.frame, textvariable=self.mode_var, values=["ECB", "CBC", "CFB", "OFB"],
state="readonly", width=5)
```

Для режимів, які потребують вектора ініціалізації (IV), програма автоматично генерує його довжиною 16 байт:

```
iv = get_random_bytes(16)
```

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		41

Цей вектор зберігається разом із ключем та шифротекстом і використовується при дешифруванні. Він не є секретним, але його правильне збереження є критично важливим для відновлення даних.

Важливою перевагою реалізованого підходу є поєднання гнучкості та безпеки. Користувач має можливість не лише обрати параметри шифрування відповідно до конкретного завдання, але й переглянути або скопіювати ключ для подальшого використання. Після завершення шифрування ключ виводиться в окремому вікні з кнопкою для копіювання в буфер обміну:

```
self.root.clipboard_clear()
self.root.clipboard_append(binascii.hexlify(key).decode())
self.root.update()
```

Процес шифрування:

Процес шифрування є центральним етапом функціонування програмного засобу, оскільки саме в цей момент виконується безпосереднє криптографічне перетворення вхідних даних. Реалізація цієї функції базується на використанні алгоритму Camellia, який підтримується сучасними криптографічними бібліотеками та відповідає міжнародним стандартам. У межах програмного забезпечення використовується бібліотека cryptography, яка забезпечує стабільну реалізацію Camellia та дозволяє гнучко працювати з різними режимами шифрування.

Перед початком шифрування вхідні дані проходять процедуру доповнення (падінгу), оскільки алгоритм Camellia є блочним і потребує фіксованої довжини вхідного блоку (128 біт або 16 байт). Якщо довжина вхідного повідомлення не кратна довжині блоку, додається службова інформація згідно зі стандартом PKCS7, яка дозволяє точно відновити початковий обсяг даних під час дешифрування. Процедура реалізована наступним чином:

```
def pad_data(data):
    pad_len = 16 - (len(data) % 16)
    return data + bytes([pad_len] * pad_len)
```

Після підготовки вхідних даних створюється об'єкт шифру (Cipher) на основі вибраного режиму, ключа та (за потреби) вектора ініціалізації. Конструкція об'єкта

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		42

залежить від того, який режим шифрування було обрано користувачем у графічному інтерфейсі:

```
if mode == "ECB":
    encrypted = camellia_ecb_encrypt(data, key)
elif mode == "CBC":
    encrypted = camellia_cbc_encrypt(data, key, iv)
elif mode == "CFB":
    encrypted = camellia_cfb_encrypt(data, key, iv)
elif mode == "OFB":
    encrypted = camellia_ofb_encrypt(data, key, iv)
```

Після завершення шифрування результат потрібно відобразити користувачеві. Для цього всі основні параметри виводяться у текстове поле: ключ, IV (якщо потрібен) та сам зашифрований текст у шістнадцятковому представленні. Приклад структури виводу:

```
self.output_text.delete("1.0", tk.END)
self.output_text.insert(tk.END, f"Ключ: {binascii.hexlify(key).decode()}\n")
if iv:
    self.output_text.insert(tk.END, f"IV: {binascii.hexlify(iv).decode()}\n")
self.output_text.insert(tk.END, f"Зашифрований текст: {binascii.hexlify(encrypted).decode()}")
```

Вивід реалізовано через компонент Text, який очищається перед кожною новою операцією та заповнюється результатом:

```
self.output_text.delete("1.0", tk.END)
self.output_text.insert(tk.END, output)
```

Після шифрування користувач може зберегти результати у вигляді окремих файлів: ключ, IV та зашифрований текст розділяються та записуються у відповідні .txt-файли. Це забезпечує не лише зручність, а й безпечне зберігання криптографічних параметрів для подальшого дешифрування.

```
final_data = iv + encrypted if iv else encrypted
with open("encrypted_output.enc", "wb") as file:
    file.write(final_data)
with open("key.txt", "w", encoding="utf-8") as file:
    file.write(binascii.hexlify(key).decode())
if iv:
    with open("iv.txt", "w", encoding="utf-8") as file:
        file.write(binascii.hexlify(iv).decode())
```

Усі операції супроводжуються захистом від помилок: якщо вхідні дані відсутні, система видає попередження, а у разі несправності шифрувального алгоритму виводиться відповідне повідомлення. Це підвищує надійність

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		43

функціоналу й робить його зручним навіть для користувачів без спеціальної технічної підготовки.

Процес дешифрування:

Процес дешифрування є логічним продовженням операції шифрування і виконує обернене перетворення даних з метою відновлення початкового змісту повідомлення або вмісту файлу. На відміну від шифрування, під час дешифрування необхідно враховувати специфіку попереднього перетворення, включаючи точне значення криптографічного ключа, режим шифрування, а також (за потреби) вектор ініціалізації. Неправильне задання хоча б одного з параметрів може призвести до неможливості відновлення оригінальних даних або до помилок при обробці.

У програмному застосунку дешифрування реалізовано з урахуванням очікуваного формату вхідних даних. Для текстової обробки використовується трирядковий або дворядковий формат: ключ, (IV) і зашифрований текст. Розпізнавання кількості рядків та їх структура реалізовані через попередній аналіз введеного тексту у вхідному полі:

```
lines = [line.strip() for line in input_data.split("\n") if line.strip()]
if len(lines) < 2:
    messagebox.showerror("Помилка", "Некоректний формат даних! Очікується два або три рядки (ключ, IV, шифротекст).")
    return
```

Після цього відбувається перетворення ключа і, за потреби, вектора ініціалізації з шістнадцяткового формату у байтовий:

```
key_line = next((line for line in lines if line.startswith("Ключ:")), None)
data_line = next((line for line in lines if line.startswith("Зашифрований текст:")), None)

if not key_line or not data_line:
    messagebox.showerror("Помилка", "Не вдалося знайти ключ або зашиф-рований текст.")
    return

key = binascii.unhexlify(key_line.split(":")[1].strip())
encrypted_data = binascii.unhexlify(data_line.split(":")[1].strip())

if len(encrypted_data) < 16:
    messagebox.showerror("Помилка", "Зашифровані дані занадто короткі (менше 16 байт)")
    return
```

Режим дешифрування визначається на основі значення, вибраного користувачем у випадваючому списку. Для режимів CBC, CFB та OFB вимагається IV, для ECB – ні. У разі відсутності IV для режимів, які його потребують, виводиться повідомлення про помилку.

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		44

Далі створюється об'єкт дешифрувальника `decryptor()` і виконується обробка зашифрованого повідомлення:

```
decrypted = self.decrypt_by_mode(encrypted_data, key)
if self.mode_var.get() in ("ECB", "CBC") and self.looks_like_text(decrypted.decode('utf-8', errors='ignore')):
    try:
        decrypted = unpad_data(decrypted)
    except Exception:
        pass
```

Після отримання дешифрованого байтового потоку програма намагається зняти паддінг, який був доданий під час шифрування. Процедура зняття паддінгу реалізована з додатковими перевірками правильності:

```
def unpad_data(data):
    pad_len = data[-1]
    if pad_len < 1 or pad_len > 16:
        raise ValueError("Invalid padding")
    if data[-pad_len:] != bytes([pad_len] * pad_len):
        raise ValueError("Invalid padding bytes")
    return data[:-pad_len]
```

Якщо розшифровані дані є текстом, вони виводяться у відповідне текстове поле. У протилежному випадку (наприклад, при розшифруванні зображень або аудіо) користувач отримає відповідне повідомлення з пропозицією зберегти файл:

```
text = self.safe_decode_text(decrypted)
if text is None:
    self.output_text.delete("1.0", tk.END)
    self.output_text.insert(tk.END, "⚠ Розшифровані дані – не текст. Збережіть їх у файл.")
    return
self.output_text.delete("1.0", tk.END)
self.output_text.insert(tk.END, f"Розшифрований текст:\n{text}")
```

Окрему увагу в реалізації приділено захисту від помилок. Усі потенційно критичні ділянки коду обгорнуті в блоки `try-except`, що дозволяє уникнути аварійного завершення програми та надати користувачу зрозуміле повідомлення про помилку. Це особливо актуально у випадках, коли ключ або IV задані неправильно, або шифротекст пошкоджений.

Робота з файлами:

У сучасних прикладних інформаційних системах захист інформації не обмежується лише текстовими даними. Практична необхідність часто вимагає захищати також документи, графічні зображення та мультимедійні файли. З огляду на це, у межах розроблюваного програмного застосунку реалізовано підтримку роботи з різними типами файлів. Програма здатна виконувати шифрування та

дешифрування текстових документів, зображень та аудіофайлів. Обробка кожного типу файлу має свою специфіку, яка враховується під час реалізації.

Обробка текстових файлів:

Для текстових даних передбачено підтримку файлів формату .txt, .docx та .pdf. При завантаженні таких файлів дані зчитуються у байтовому представленні для подальшої криптографічної обробки. Текстовий формат обробляється стандартними засобами мови Python:

```
if ext in (".txt", ".docx", ".pdf"):
    with open(filepath, "rb") as file:
        data = file.read()
```

Після шифрування текстового файлу результат записується у новий документ того ж типу, або в окремий .enc-файл. При збереженні зашифрованих текстових даних можна також окремо зберігати ключ та IV у службові файли.

```
encrypted_filepath = filepath + ".enc"
with open(encrypted_filepath, "wb") as file:
    file.write(iv + encrypted if iv else encrypted)
with open(key_file, "w", encoding="utf-8") as file:
    file.write(key)
```

Обробка зображень:

Зображення у форматах .png та .jpg також підтримуються системою. При шифруванні графічного файлу обробка здійснюється на рівні байтів: весь файл читається у вигляді бінарного потоку, після чого шифрується згідно з обраним режимом.

```
image = Image.open(filepath).convert("RGB")
width, height = image.size
img_bytes = image.tobytes()
size_header = struct.pack(">II", width, height)
full_data = size_header + img_bytes
```

Зберігання відбувається в новий графічний файл (наприклад, encrypted_image.png), який згодом може бути дешифрований, якщо користувач має правильний ключ та IV.

```
encrypted_filepath = filepath + ".enc"
with open(encrypted_filepath, "wb") as file:
    file.write(iv + encrypted if iv else encrypted)
```

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		46

Обробка аудіофайлів:

Для аудіофайлів підтримуються формати .mp3 та .wav. Шифрування реалізується за принципом обробки байтового потоку. Після зчитування вмісту файл шифрується стандартним способом, а результат зберігається з відповідним розширенням:

```
with open(filepath, "rb") as file:
    audio_data = file.read()
    encrypted, iv = self.encrypt_by_mode(audio_data, key)
```

Відтворення зашифрованого аудіофайлу в класичному вигляді є неможливим, але після дешифрування бінарні дані зберігаються у вихідному форматі з можливістю повторного прослуховування.

```
save_path = filedialog.asksaveasfilename(defaultextension=".wav",
                                          filetype=[("Аудіофайли", "*.mp3;*.wav")])
if save_path:
    with open(save_path, "wb") as file:
        file.write(decrypted)
```

Інтерфейс користувача для роботи з файлами:

У графічному інтерфейсі користувач має змогу обрати тип дії (шифрування/дешифрування), натиснути кнопку вибору файлу, після чого дані автоматично зчитуються та обробляються. Для кожного типу файлу передбачено відповідну реалізацію обробника, яка викликається у межах функцій encrypt_file() або decrypt_file():

```
ext = os.path.splitext(filepath)[-1].lower()

try:
    if ext in (".txt", ".docx", ".pdf"):
        with open(filepath, "rb") as file:
            data = file.read()
    else:
        messagebox.showerror("Помилка", f"Формат файлу {ext} не підтримується!")
        return
except Exception as e:
    messagebox.showerror("Помилка читання", str(e))
    return

encrypted, iv = self.encrypt_by_mode(data, key)
with open(filepath + ".enc", "wb") as file:
    file.write(iv + encrypted if iv else encrypted)
```

Аналогічно, при виконанні дешифрування, застосовується функція decrypt_file(), у якій реалізовано логіку зчитування ключа, дешифрування даних та

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		47

збереження результату у вибраному форматі. Фрагмент коду виглядає наступним чином

```
key_hex = simpledialog.askstring("Введіть ключ", "Введіть шістнадцятковий ключ:")
key = binascii.unhexlify(key_hex)

with open(filepath, "rb") as file:
    encrypted_data = file.read()

decrypted = self.decrypt_by_mode_for_file(encrypted_data, key, "binary")

save_path = filedialog.asksaveasfilename(...)
with open(save_path, "wb") as file:
    file.write(decrypted)
```

Це забезпечує автоматичну класифікацію файлів і використання правильного методу обробки без необхідності втручання користувача.

2.5 Аналіз продуктивності та адаптивне налаштування параметрів шифрування

У процесі розробки криптографічного програмного забезпечення важливо не лише реалізувати функціональну повноту, але й забезпечити ефективну роботу алгоритмів та зручність адаптації параметрів під різні завдання. У зв'язку з цим окрему увагу приділено питанням оптимізації продуктивності та реалізації механізмів, що дозволяють налаштовувати криптографічні параметри в динамічному режимі.

З цією метою в інтерфейс програмного застосунку буде інтегровано інструмент бенчмаркінгу, який дозволить користувачеві здійснювати аналіз продуктивності при виборі різних режимів шифрування. Після завантаження файлів система надасть можливість обрати один із трьох режимів аналізу:

- Порівняння режимів шифрування (ECB, CBC, CFB, OFB) — дозволить оцінити відносну швидкість кожного режиму на вибраних даних;
- Аналіз продуктивності за типами файлів (текст, зображення, аудіо) — забезпечить групування результатів з метою виявлення особливостей обробки;
- Порівняння базової та оптимізованої реалізації — буде реалізовано шляхом дворазового шифрування і дозволить виявити вплив повторної ініціалізації, кешування або оптимізації виконання.

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		48

Усі вимірювання часу виконуватимуться за допомогою високоточних таймерів з використанням методу `time.perf_counter()`. Результати будуть візуалізовані у вигляді графіків за допомогою засобів бібліотеки `matplotlib`.

Система відображатиме результати у вигляді стовпчастих діаграм з розбивкою за режимами або за типами вхідних файлів. Для цього буде реалізовано відповідні методи агрегування та виводу статистичних даних.

Ключові фрагменти реалізації бенчмаркінгу передбачатимуть використання такої логіки:

```
for mode_name in ["ECB", "CBC", "CFB", "OFB"]:  
    app.mode_var.set(mode_name)  
    key = get_random_bytes(16)  
    encrypted, _ = app.encrypt_by_mode(data, key)
```

Для підтримки адаптивного налаштування буде запроваджено можливість автоматичного визначення типу файлу та рекомендації найбільш доцільного режиму шифрування. Наприклад, при виявленні аудіофайлу програма автоматично надаватиме пріоритет потоковим режимам (CFB, OFB), тоді як для документів доцільніше буде застосовувати блочний режим CBC.

Висновки до другого розділу

У другому розділі було розкрито бачення архітектури та функціонального наповнення програмного застосунку для шифрування та розшифрування даних із використанням алгоритму Camellia. Описано основні принципи побудови системи, логіку взаємодії з користувачем, а також технічні рішення, що забезпечують обробку текстових, графічних і аудіофайлів. Основна увага зосереджена на поєднанні зручності інтерфейсу з гнучкістю вибору параметрів шифрування.

Обрані технології – мова Python та середовище Visual Studio Code виправдані як з точки зору реалізації криптографічного функціоналу, так і для побудови графічного інтерфейсу. Здійснено аналіз особливостей кількох режимів шифрування, що будуть реалізовані в застосунку, з урахуванням їхніх переваг і потенційних обмежень. Це дозволяє орієнтувати розробку не лише на академічну мету, а й на практичне використання програми.

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				49
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ТЕСТУВАННЯ ТА АНАЛІЗ РОБОТИ ПРОГРАМНОГО ЗАСТОСУНКУ

3.1 Тестування програми

Після реалізації програмного забезпечення було проведено його тестування з метою перевірки коректності роботи всіх основних функцій, стабільності інтерфейсу та обробки різних типів вхідних даних.

Для перевірки працездатності графічного інтерфейсу Camellia Encryption Tool було проведено тестування шифрування та розшифрування тексту у режимі ЕСВ з ключем довжиною 128 біт. На Рис. 3.1 показано головне вікно програми з введеним англomовним текстом, вибраними параметрами шифрування та згенерованим результатом у полі «Результат».

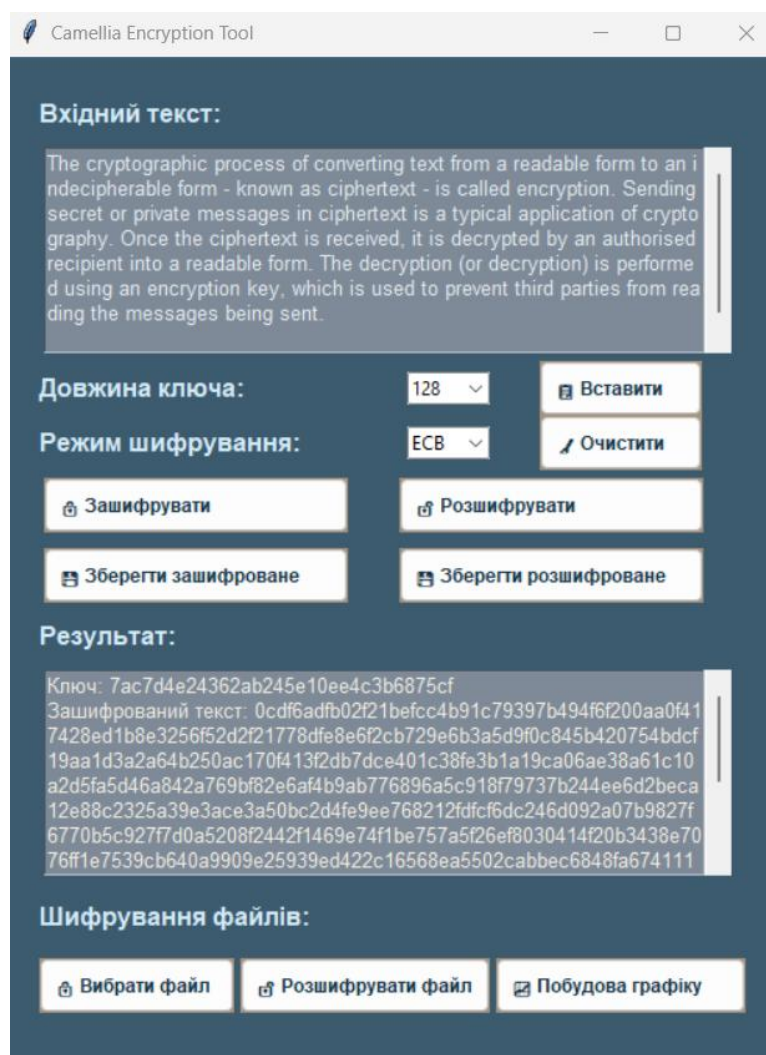


Рис. 3.1. Головне вікно програми після шифрування тексту

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				50
Змн.	Арк.	№ докум.	Підпис	Дата		

Після отримання зашифрованого результату користувач має змогу зберегти ключ та зашифрований текст у два окремі файли. На Рис. 3.2 представлено вікно збереження, у якому задається назва майбутнього файлу.

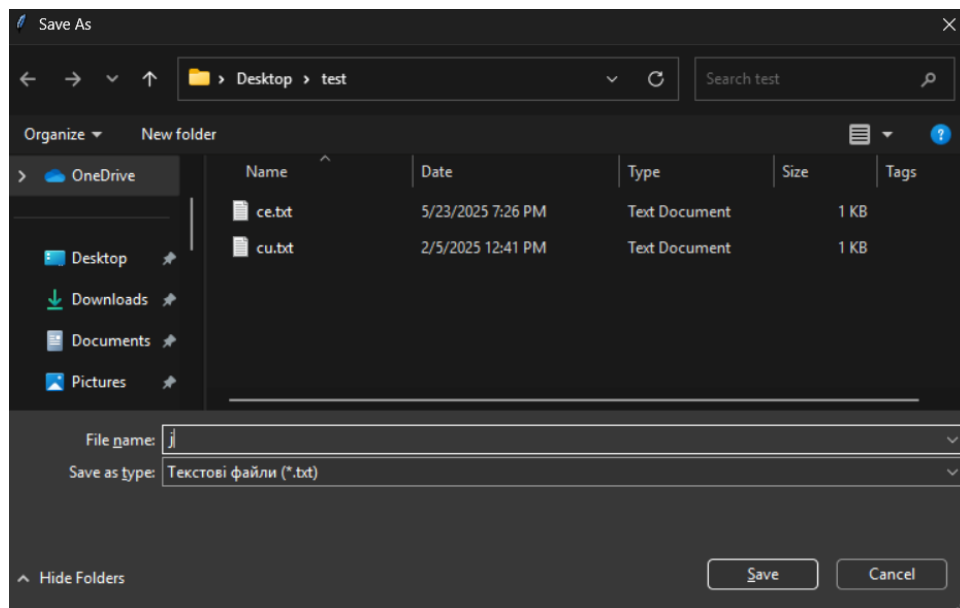


Рис. 3.2. Діалогове вікно для збереження результатів шифрування

Після завершення операції збереження виводиться повідомлення про успішне створення файлів з ключем та зашифрованим текстом (Рис. 3.3).

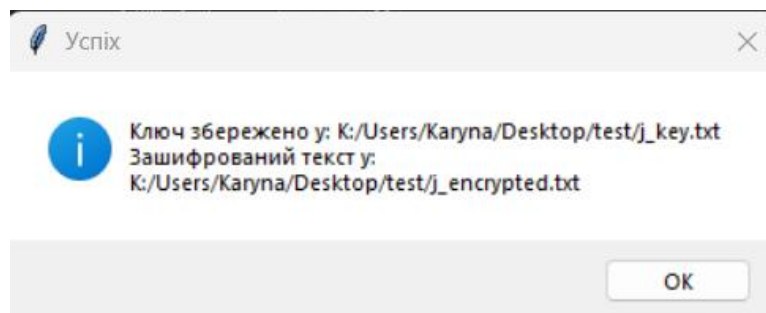


Рис. 3.3. Повідомлення про успішне збереження ключа та зашифрованого тексту

Вміст відповідних файлів можна переглянути у текстовому редакторі. На Рис. 3.4 представлено файл j_key.txt, що містить шістнадцяткове подання ключа, та файл j_encrypted.txt, який містить результат шифрування.

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				51
Змн.	Арк.	№ докум.	Підпис	Дата		

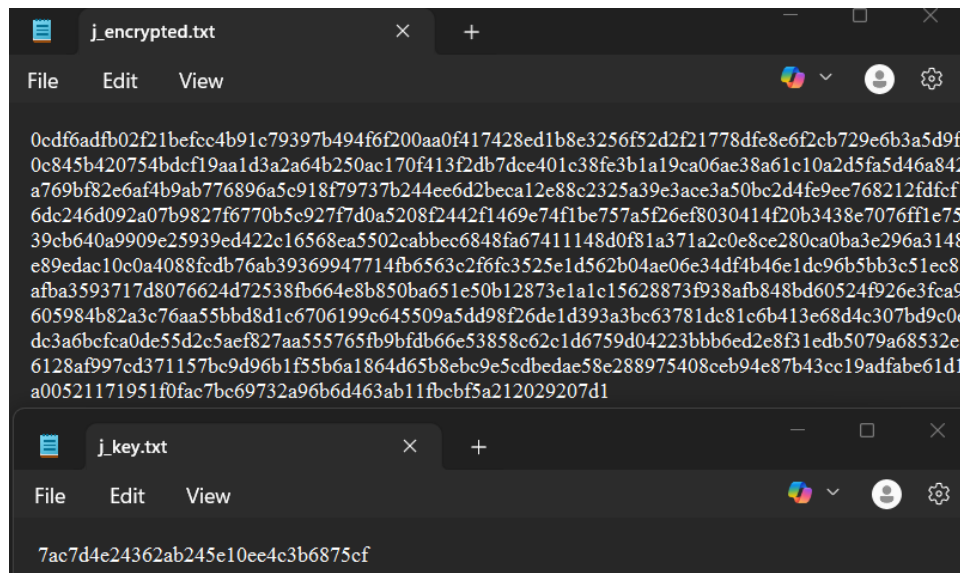


Рис. 3.4. Вміст зашифрованого тексту та ключа у відповідних файлах

Далі було виконано розшифрування зашифрованого тексту. На Рис. 3.5 видно, що результат розшифрування повністю відповідає вхідному тексту, що підтверджує коректність реалізації зворотної процедури.

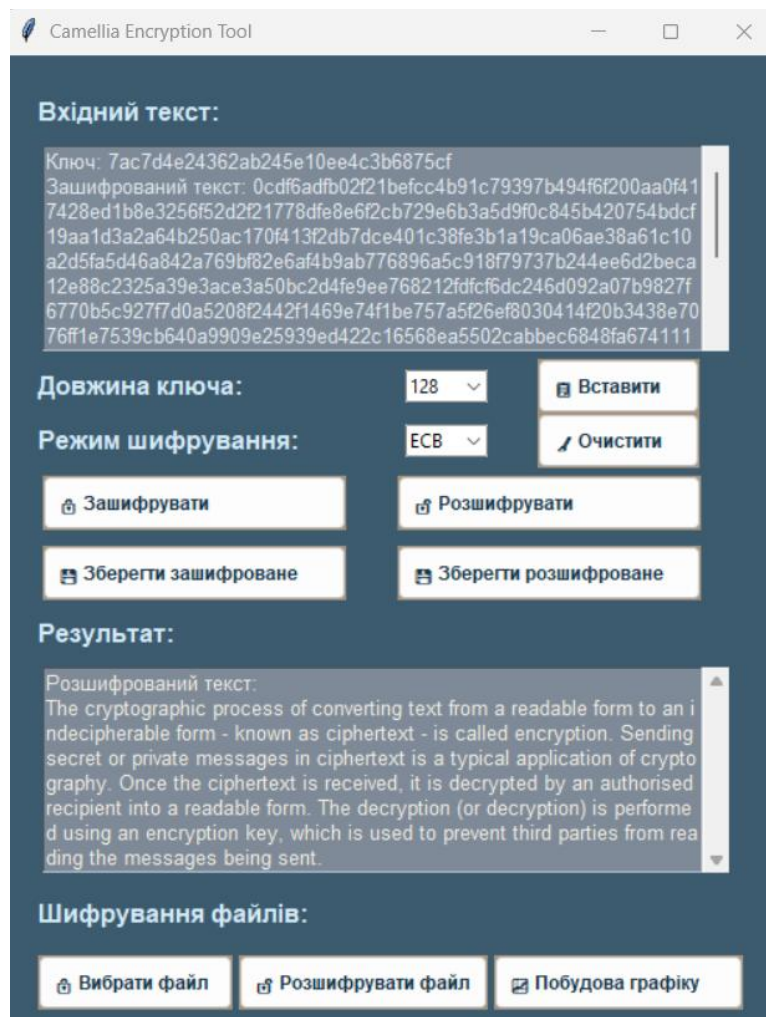


Рис. 3.5. Розшифрування тексту з виведенням результату у вікні програми

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				52
Змн.	Арк.	№ докум.	Підпис	Дата		

Розшифрований текст також можна зберегти у текстовий файл. Процес вибору шляху та імені файлу показано на Рис. 3.6.

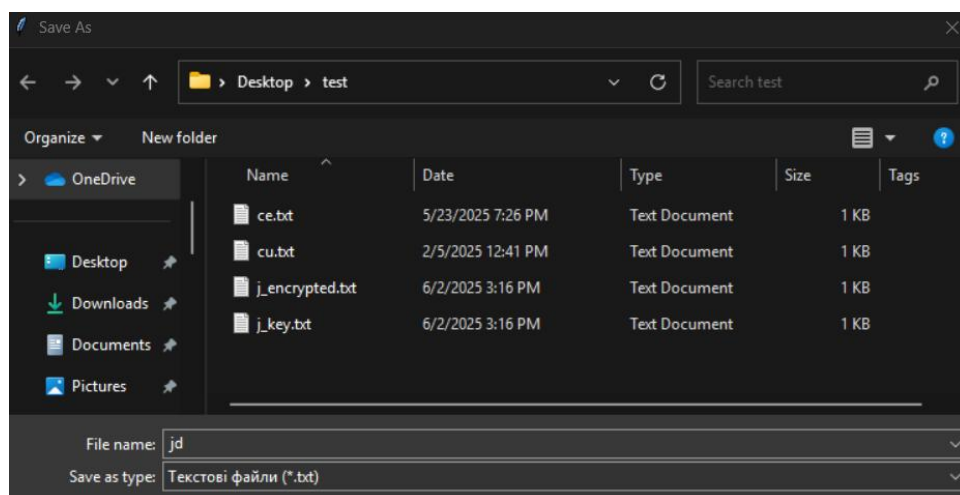


Рис. 3.6. Збереження результату розшифрування

Після збереження з'являється відповідне повідомлення (Рис. 3.7), яке підтверджує успішне завершення операції.

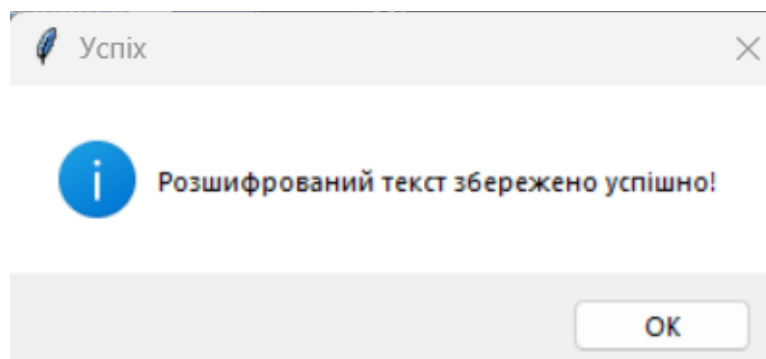


Рис. 3.7. Повідомлення про збереження розшифрованого тексту

Насамкінець, на Рис. 3.8 показано вміст збереженого файлу jd.txt, який містить коректно відновлений текстовий вміст.

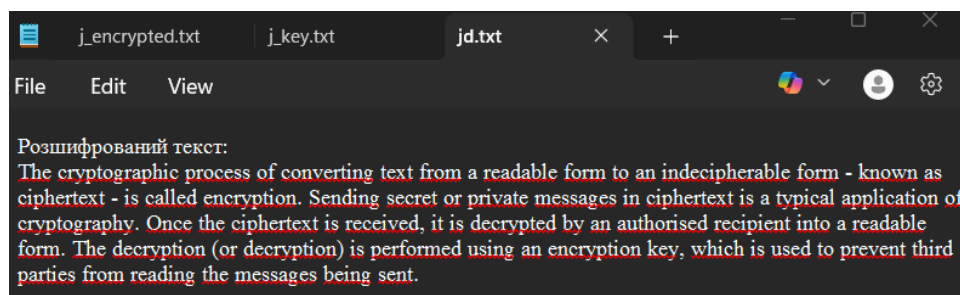


Рис. 3.8. Вміст файлу з розшифрованим текстом

Окрім англomовного тексту, було протестовано шифрування та розшифрування україномовного тексту з використанням ключа довжиною 192 біти

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		53

та режиму CFB. На Рис. 3.9 продемонстровано інтерфейс програми після шифрування українського тексту. Видно, що результат містить ключ, вектор ініціалізації (IV) та зашифрований текст.

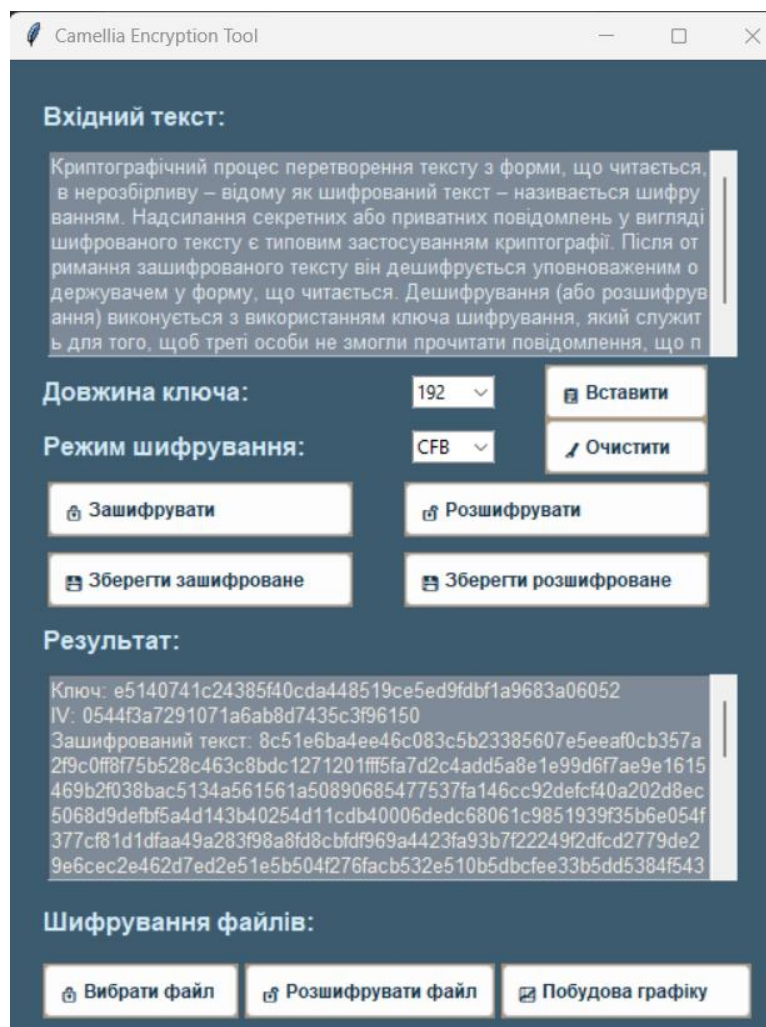


Рис. 3.9. Шифрування україномовного тексту у режимі CFB з довжиною ключа 192 біти

Далі було виконано розшифрування отриманого шифртексту. На Рис. 3.10 показано результат – повністю відновлений україномовний вхідний текст, який точно збігається з оригіналом. Це свідчить про правильну реалізацію алгоритму Camellia у режимі CFB для 192-бітового ключа.

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		54

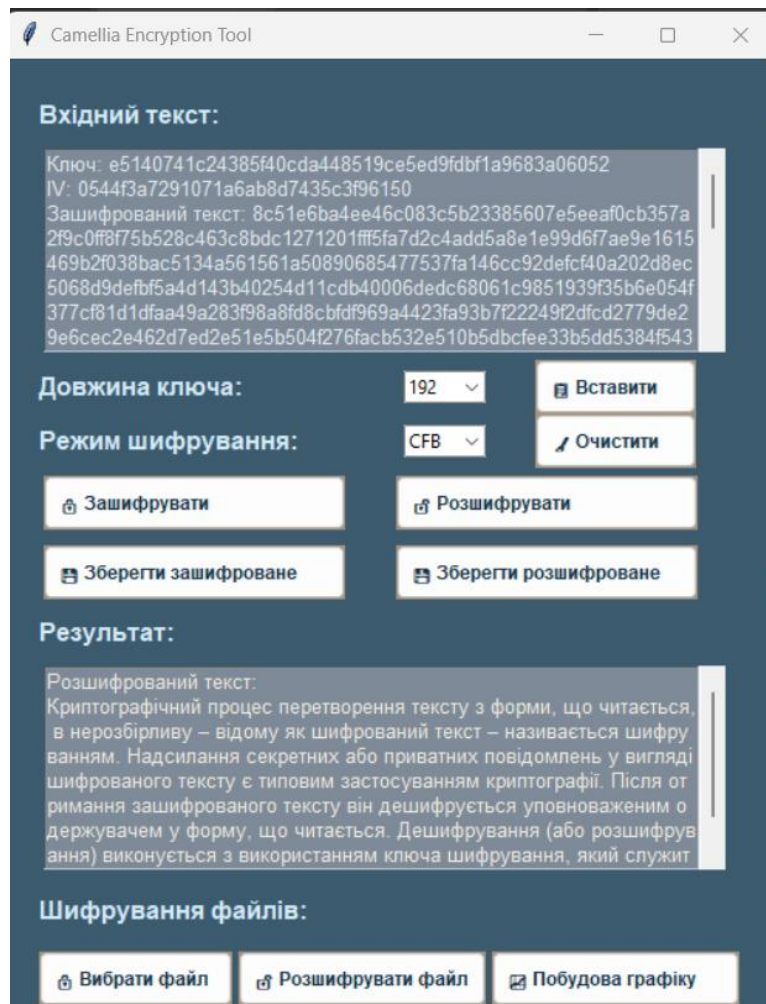


Рис. 3.10. Результат розшифрування україномовного тексту

Однією з важливих функцій Camellia Encryption Tool є можливість шифрування зображень у форматах PNG і JPEG. Для перевірки цієї можливості було обрано тестове зображення та проведено повний цикл: шифрування, копіювання ключа, розшифрування, збереження, візуальна перевірка.

На Рис. 3.11 показано вікно вибору файлу, де обирається зображення для шифрування.

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		55

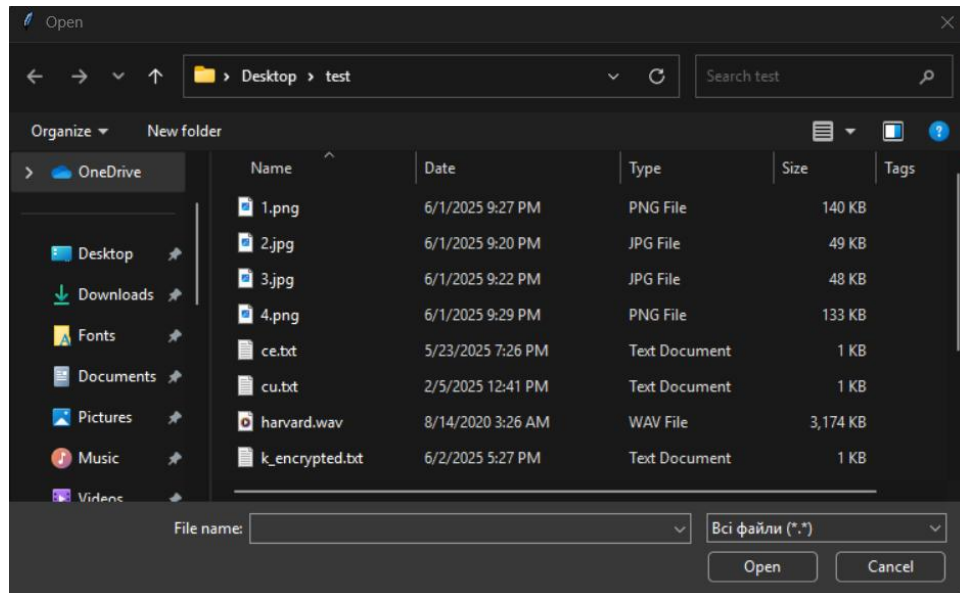


Рис. 3.11. Вибір зображення для шифрування

Після завершення процесу шифрування програма повідомляє користувача про успішне створення зашифрованого файлу з розширенням .enc (Рис. 3.12).

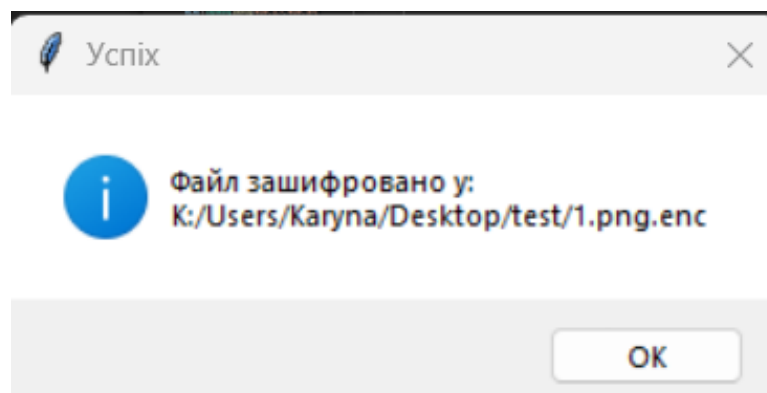


Рис. 3.12. Повідомлення про успішне шифрування зображення

Після завершення шифрування зображення відкривається вікно з відображеним згенерованим ключем шифрування. Користувач може натиснути кнопку для його копіювання та закриття діалогового вікна. Це зображено на Рис. 3.13.

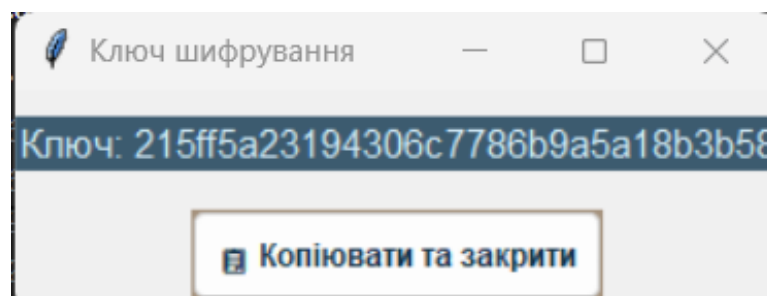


Рис. 3.13. Вікно з ключем шифрування та кнопкою копіювання

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		56

Одночасно користувачеві відображається вікно з ключем, який автоматично копіюється у буфер обміну (Рис. 3.14).

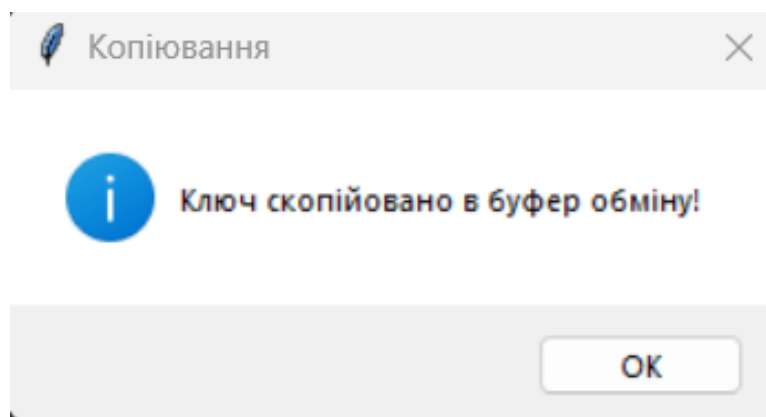


Рис. 3.14. Повідомлення про копіювання ключа шифрування

Далі для розшифрування обирається зашифрований файл з розширенням .enc, як показано на Рис. 3.15.

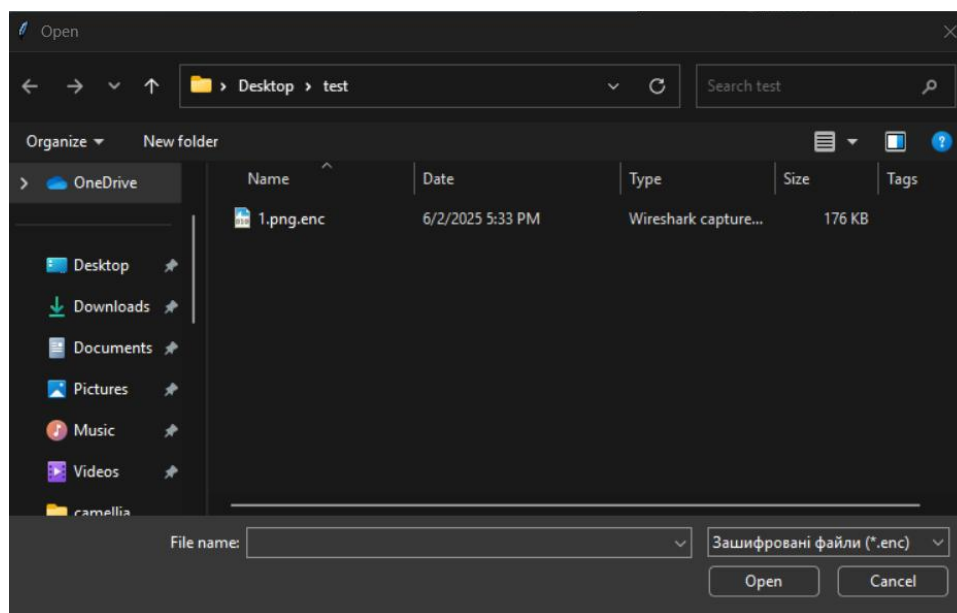


Рис. 3.15. Вибір зашифрованого зображення для розшифрування

Після цього відкривається вікно збереження ключа, в якому користувач вставляє отриманий ключ (Рис. 3.16).

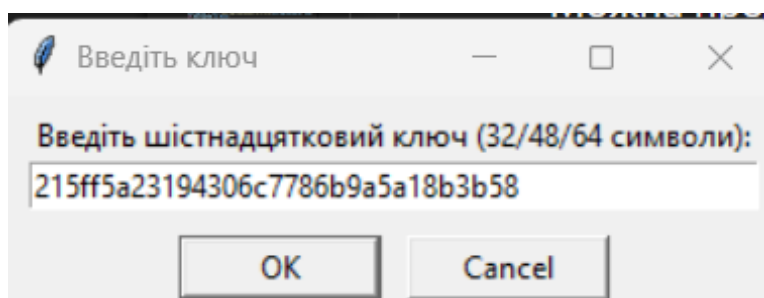


Рис. 3.16. Введення ключа для розшифрування зображення

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		57

Програма дозволяє обрати ім'я та місце для збереження розшифрованого зображення, що видно на Рис. 3.17.

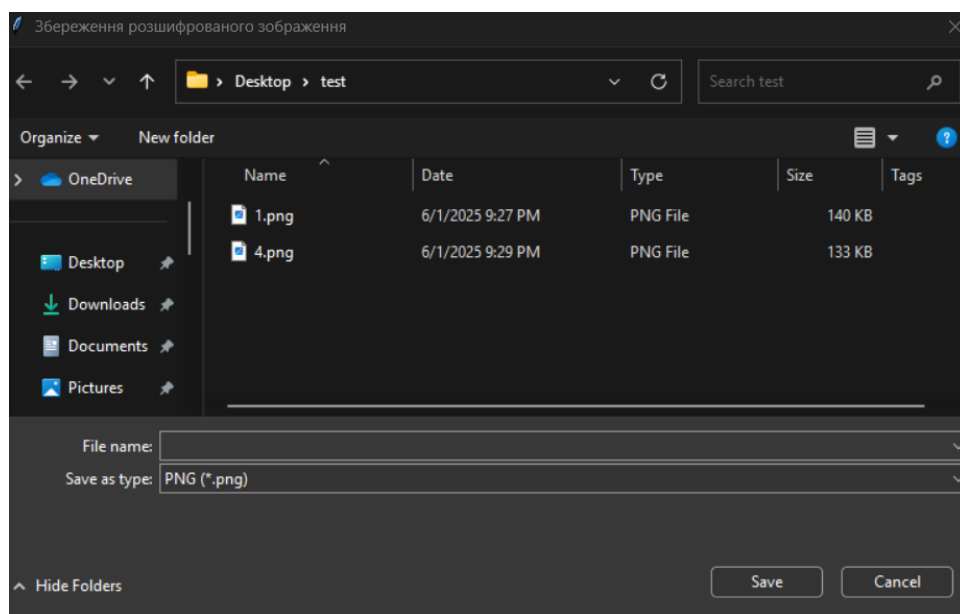


Рис. 3.17. Вибір шляху збереження відновленого зображення

Після успішного завершення розшифрування програма повідомляє про це користувача (Рис. 3.18).

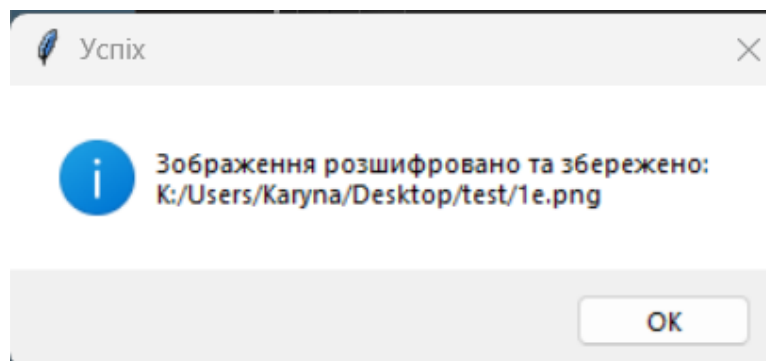


Рис. 3.18. Повідомлення про успішне збереження розшифрованого зображення

Для перевірки якості збереження зображення були відкриті як оригінальний файл 1.png, так і розшифрований файл 1e.png. Обидва файли зображені на Рис. 3.19 та Рис. 3.20 відповідно. Видно, що зображення ідентичні, що підтверджує коректність алгоритму.

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		58

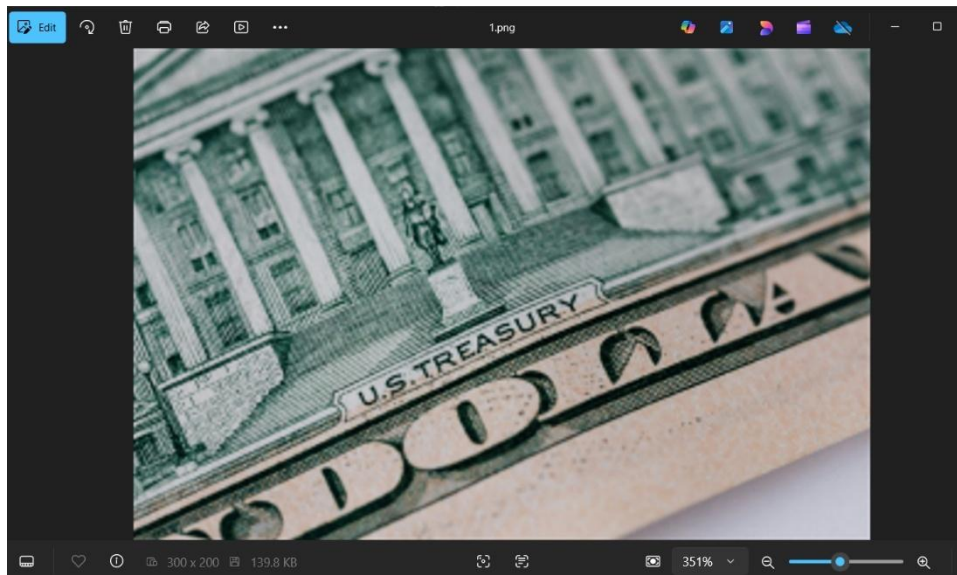


Рис. 3.19. Оригінальне зображення перед шифруванням

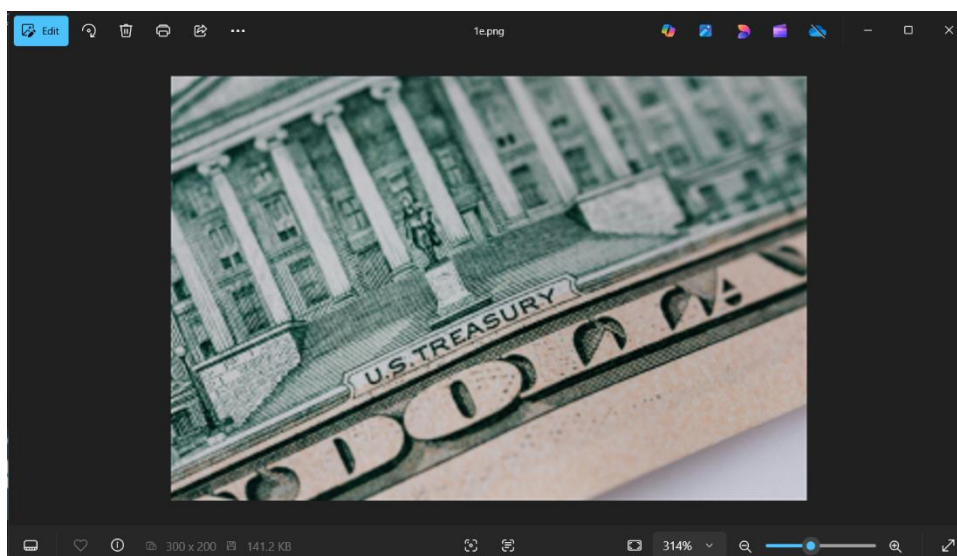


Рис. 3.20. Відновлене зображення після розшифрування

Шифрування документів Microsoft Word (*.docx) виконується за тією ж логікою, що й для зображень або текстових файлів, продемонстрованою раніше. Після вибору потрібного документа користувач виконує його шифрування. На Рис. 3.21 показано повідомлення про успішне збереження зашифрованого файлу разом із вікном, яке містить згенерований ключ шифрування.

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		59

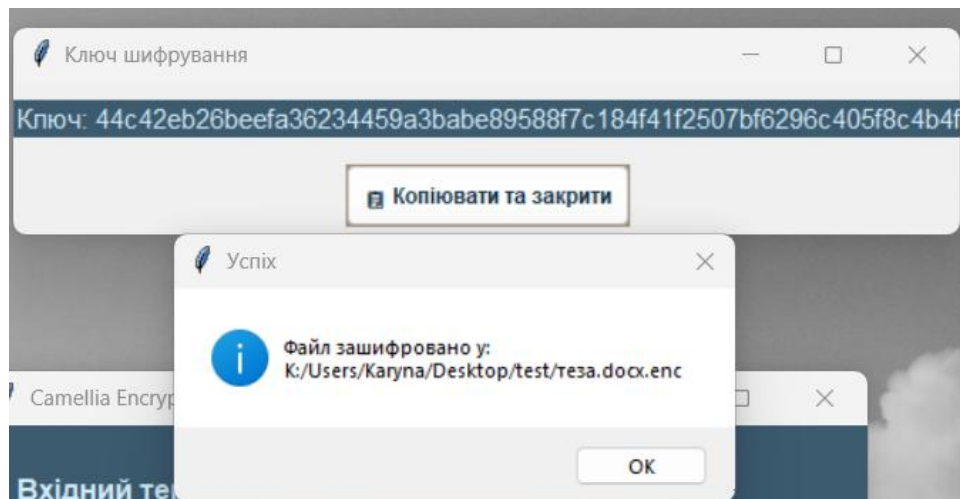


Рис. 3.21. Вікно з ключем шифрування та повідомлення про успішне збереження зашифрованого DOCX-файлу

На етапі розшифрування користувач вводить цей самий ключ у відповідне поле, як показано на Рис. 3.22.

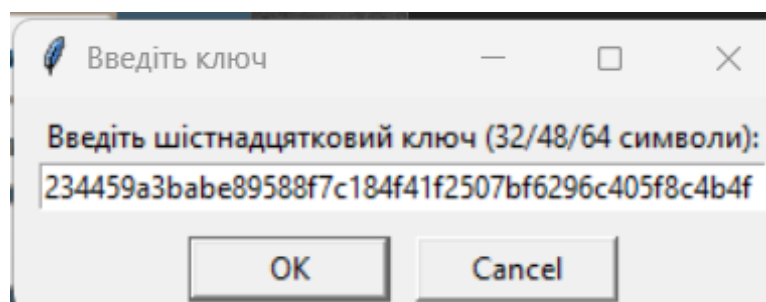


Рис. 3.22. Введення шістнадцяткового ключа для розшифрування

Після цього програма надає змогу вказати назву та місце для збереження відновленого документа (Рис. 3.23).

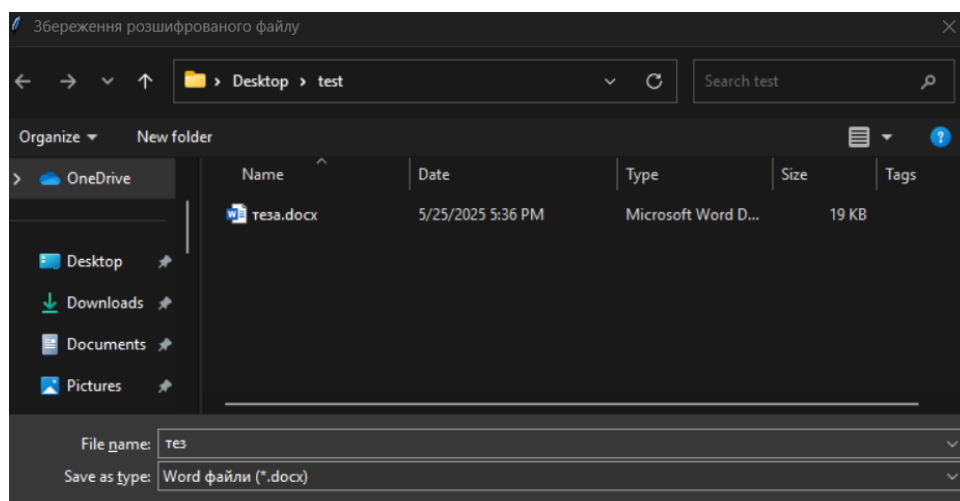


Рис. 3.23. Вікно збереження розшифрованого документа формату DOCX

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				60
Змн.	Арк.	№ докум.	Підпис	Дата		

Після розшифрування файл відкривається без спотворення змісту або структури. На Рис. 3.24 показано документ після розшифрування у Word – усі текстові стилі, форматування та вміст збережено.

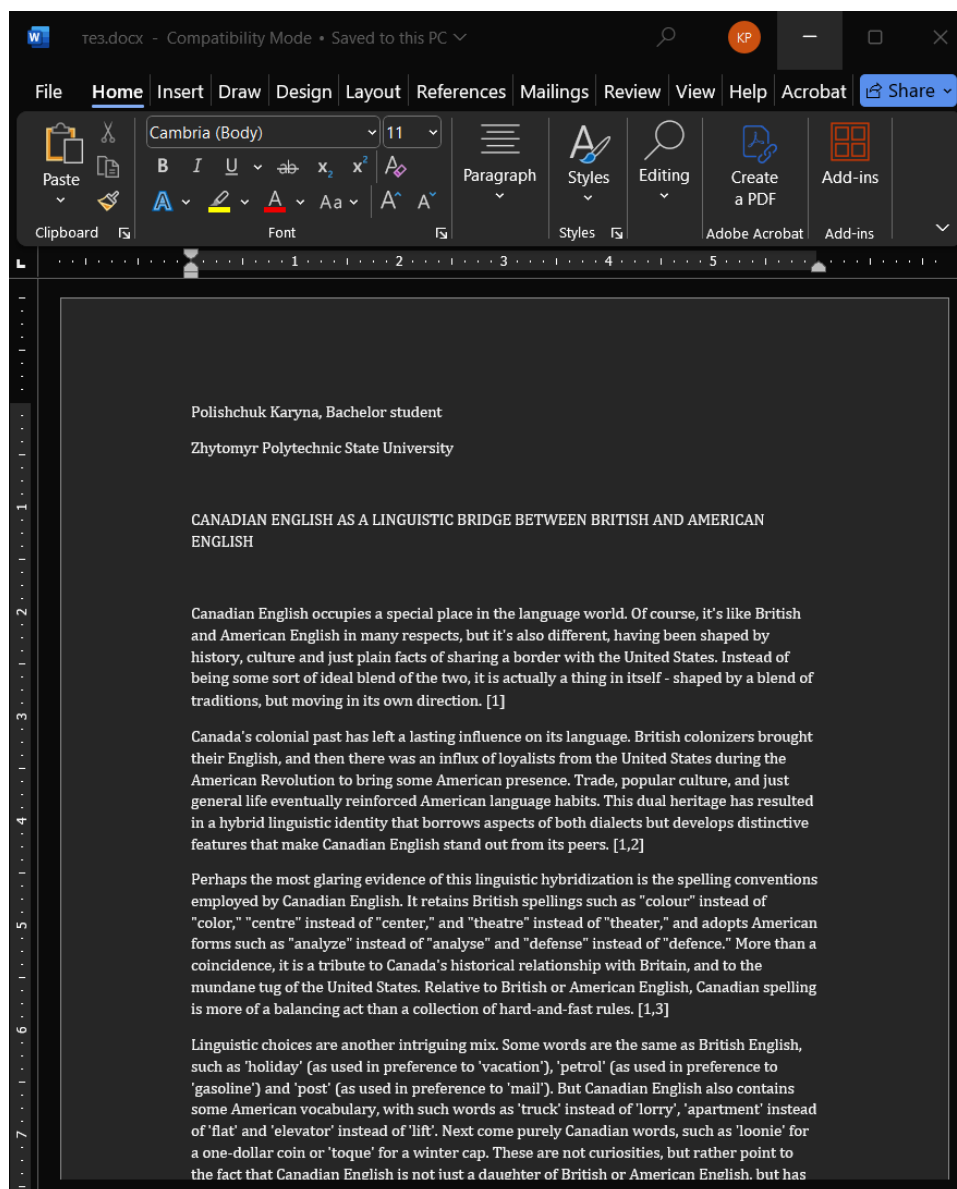


Рис. 3.24. Відновлений документ після розшифрування

Для порівняння, на Рис. 3.25 наведено той самий документ у його оригінальному вигляді до шифрування. Як видно, зміст повністю збігається, що підтверджує правильність реалізації алгоритму Camellia для DOCX-файлів.

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				61
Змн.	Арк.	№ докум.	Підпис	Дата		

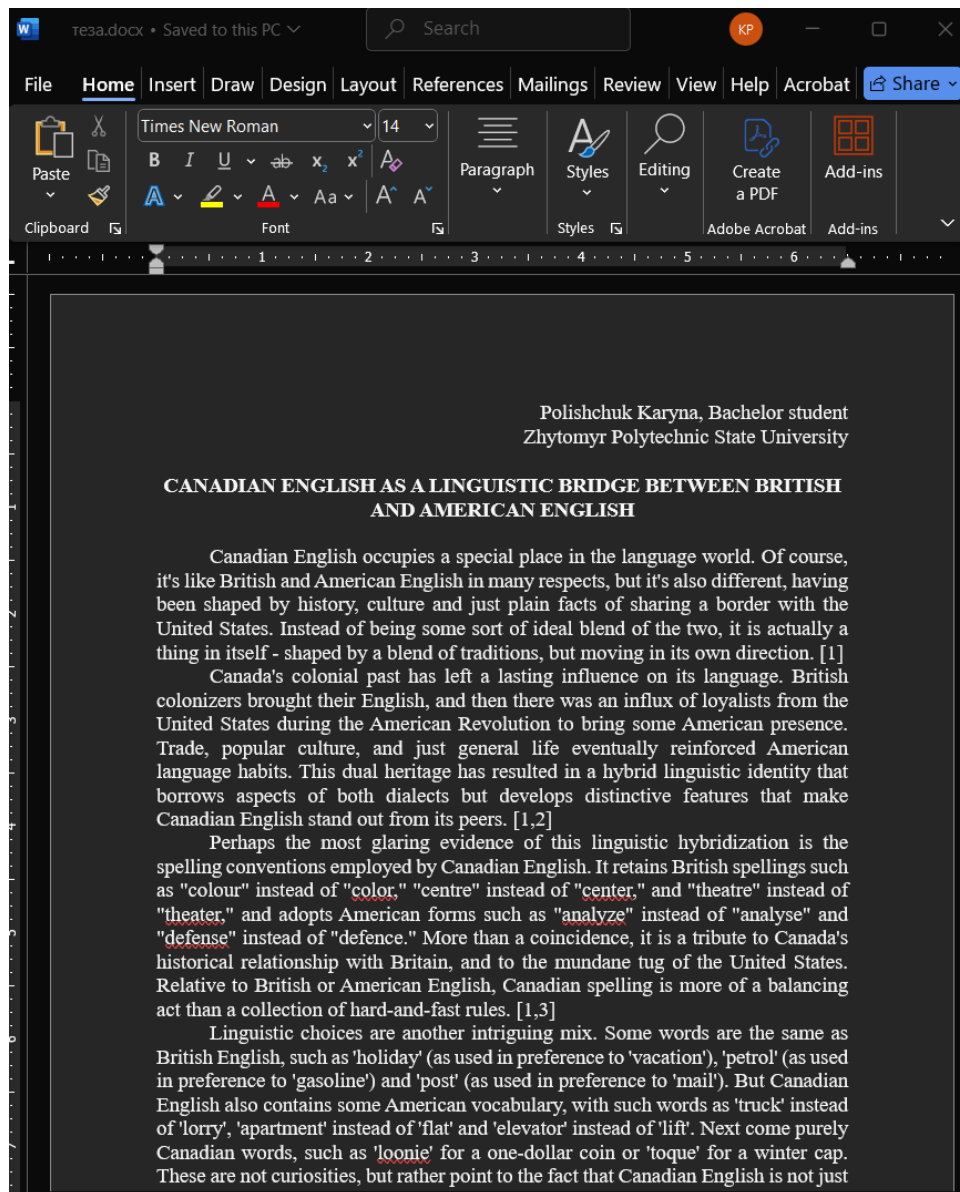


Рис. 3.25. Оригінальний документ Word до шифрування

Програма також підтримує роботу з аудіофайлами у форматах .mp3 та .wav. Процес їхнього шифрування та відновлення аналогічний до роботи з іншими типами файлів. На Рис. 3.26 показано вибір MP3-файлу для шифрування.

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		62

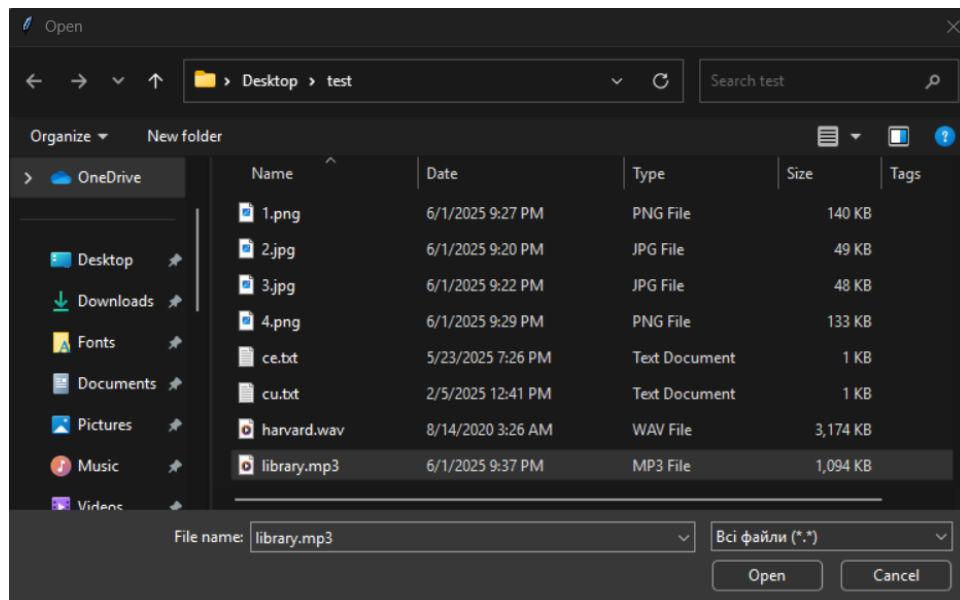


Рис. 3.26. Вибір аудіофайлу для шифрування

Після завершення шифрування у вікні зберігається файл з розширенням .enc, а також формується ключ, необхідний для розшифрування (аналогічно до попередніх типів файлів).

Далі користувач розпочинає процес розшифрування: програма відкриває вікно вибору зашифрованого аудіофайлу. Після цього виконується збереження результату, як показано на Рис. 3.27.

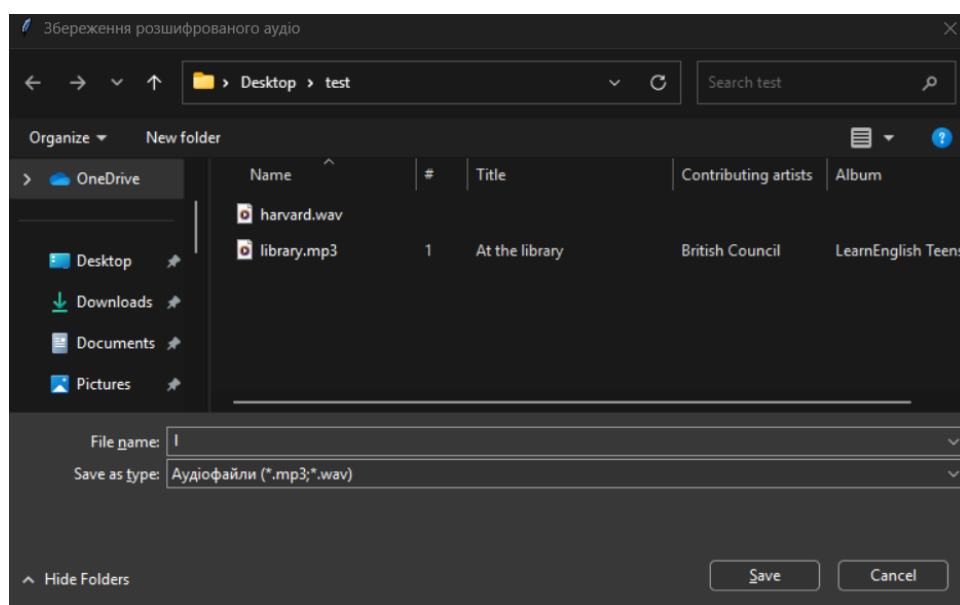


Рис. 3.27. Збереження розшифрованого аудіофайлу у форматі MP3 або WAV

У результаті в директорії з'являються вихідний файл, зашифрований варіант та розшифрований аудіофайл, як це видно на Рис. 3.28.

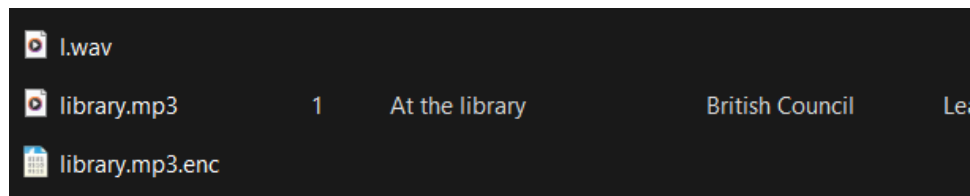


Рис. 3.28. Перевірка наявності зашифрованого та розшифрованого аудіофайлу

Для підтвердження коректності розшифрування файли було відкрито у стандартному медіаплеєрі. Рис. 3.29 демонструє оригінальний аудіофайл до шифрування зі збереженими метаданими (обкладинка, назва, автор).

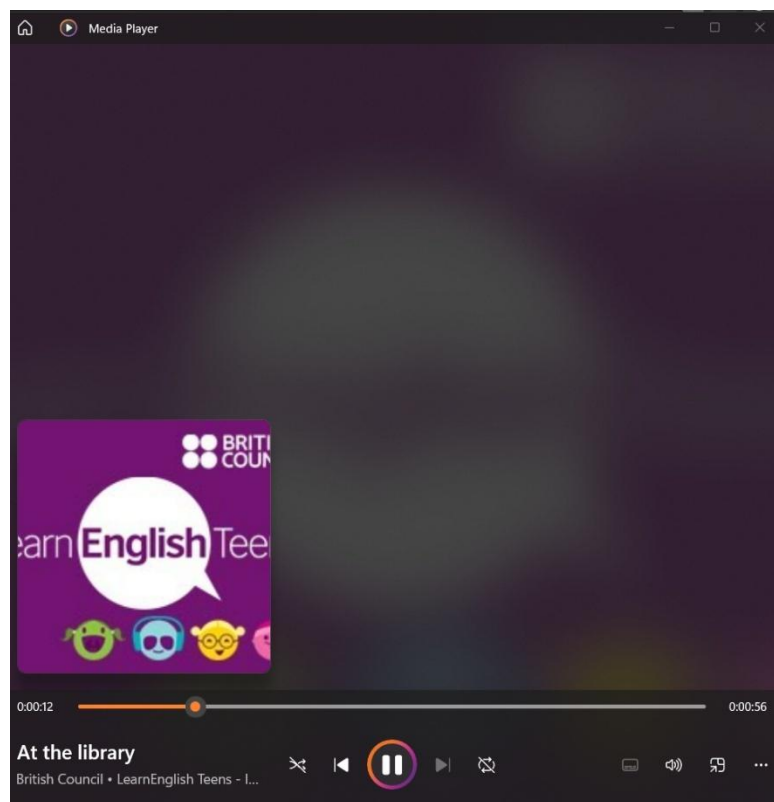


Рис. 3.29. Відтворення оригінального аудіофайлу у плеєрі

На Рис. 3.30 показано розшифрований аудіофайл – він відтворюється повністю, але не містить обкладинки, оскільки вектор ініціалізації (IV) та ключ не включалися у структуру ID3-тегів під час шифрування.

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		64

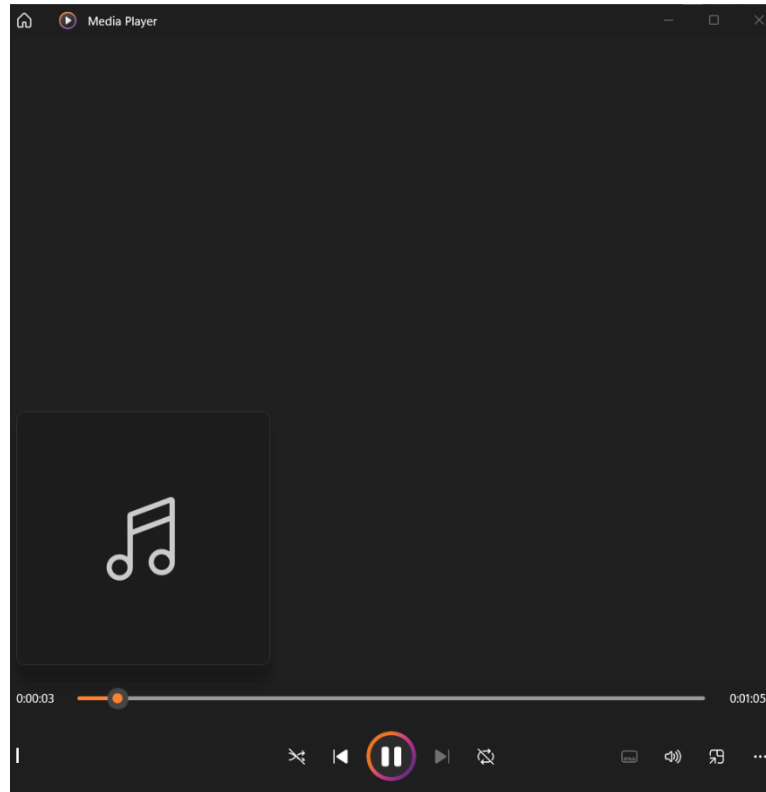


Рис. 3.30. Відтворення розшифрованого аудіофайлу у плеєрі

Після вибору PDF-файлу програма виконує його шифрування, формує зашифрований файл з розширенням .enc і виводить повідомлення про успіх. Це проілюстровано на Рис. 3.31.

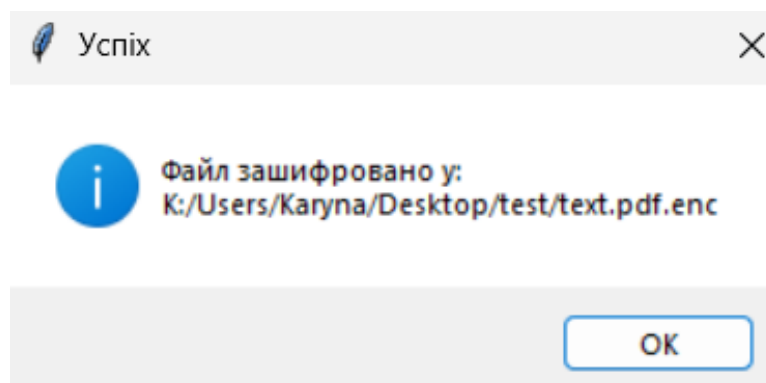


Рис. 3.31. Повідомлення про успішне шифрування PDF-документа

Процес розшифрування PDF-файлу нічим не відрізняється від попередніх прикладів. Після введення ключа користувач обирає, куди зберегти відновлений файл. На Рис. 3.32 зображено вікно збереження розшифрованого PDF-документа.

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		65

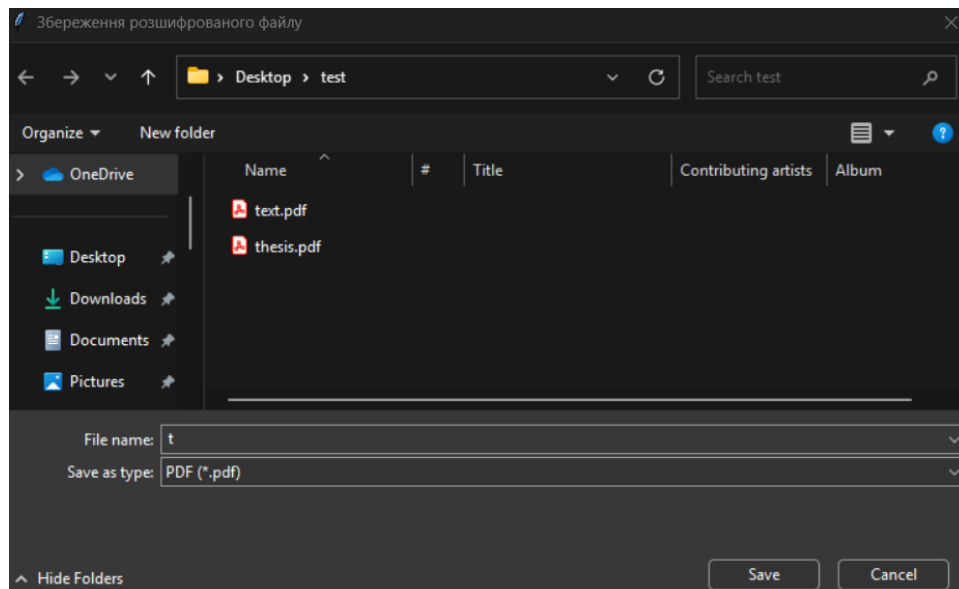


Рис. 3.32. Вибір шляху для збереження розшифрованого PDF-файлу

Після завершення розшифрування виводиться повідомлення про успішне збереження результату (Рис. 3.33).

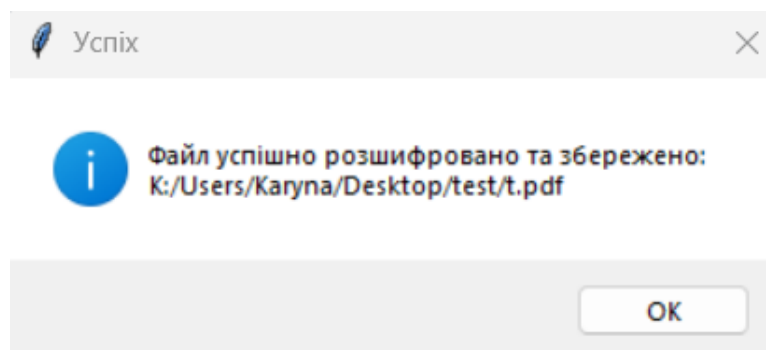


Рис. 3.33. Повідомлення про успішне розшифрування PDF-документа

Для перевірки було відкрито оригінальний PDF-файл (text.pdf) і розшифрований (t.pdf). На Рис. 3.34 зображено вигляд оригінального документа до шифрування, а на Рис. 3.35 – відновленого після розшифрування. Вміст і структура повністю збережені.

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		66

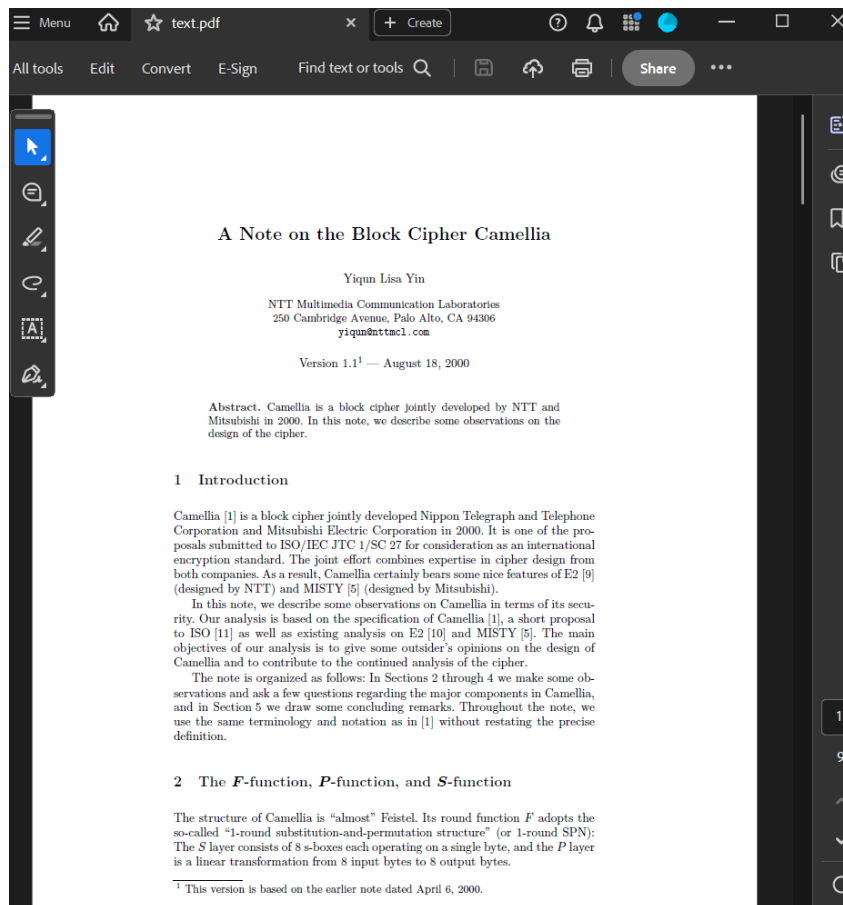


Рис. 3.34. Вигляд оригінального PDF-документа до шифрування

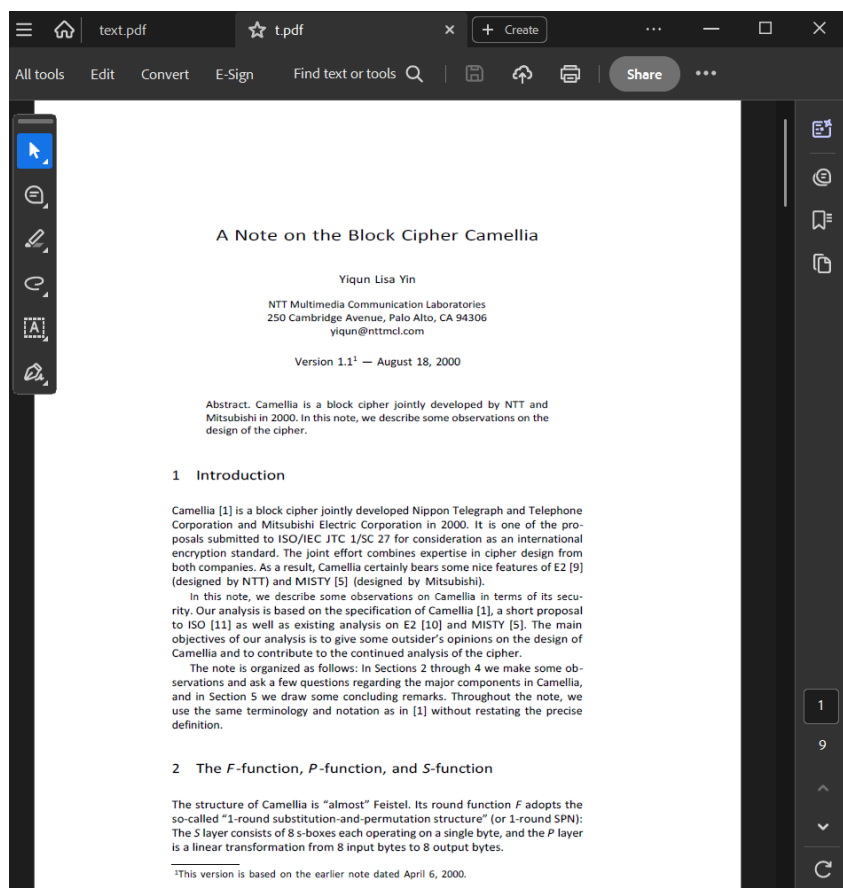


Рис. 3.35. Вигляд розшифрованого PDF-документа

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				67
Змн.	Арк.	№ докум.	Підпис	Дата		

3.2 Аналіз продуктивності шифрування в різних режимах

Під час використання блочних шифрів важливо враховувати не лише рівень безпеки, а й продуктивність обраного режиму шифрування. Для цього у програму було інтегровано модуль вимірювання часу шифрування файлів у різних режимах.

Після натискання кнопки «Побудова графіку» (Рис. 3.36) користувач обирає набір файлів різних типів (текстові, зображення, аудіо, документи) для подальшого вимірювання часу шифрування (Рис. 3.37).

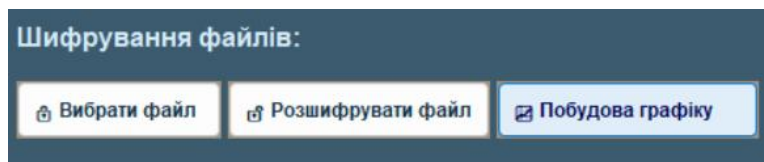


Рис. 3.36. Кнопка запуску модулю продуктивності

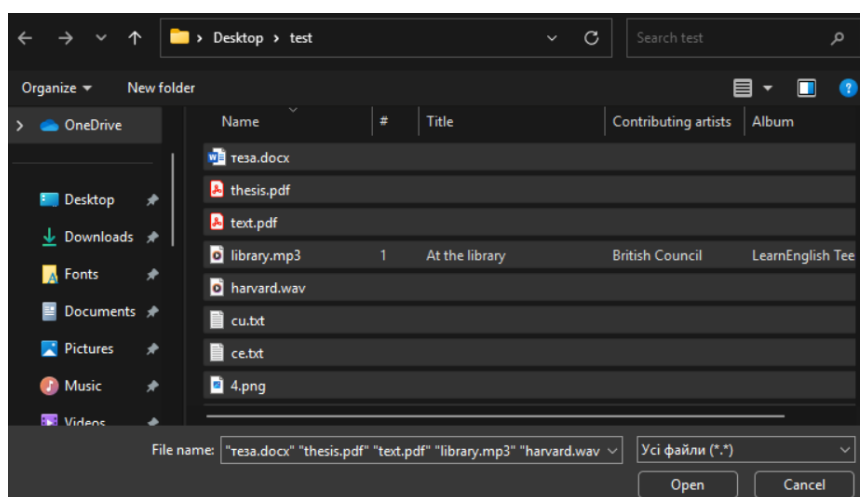


Рис. 3.37. Вибір файлів для тестування в режимах шифрування

Далі відкривається вікно з трьома варіантами побудови графіка. Обрано пункт «Порівняння режимів» (Рис. 3.38), що дозволяє побачити, як змінюється час шифрування в залежності від обраного режиму – ECB, CBC, CFB або OFB.

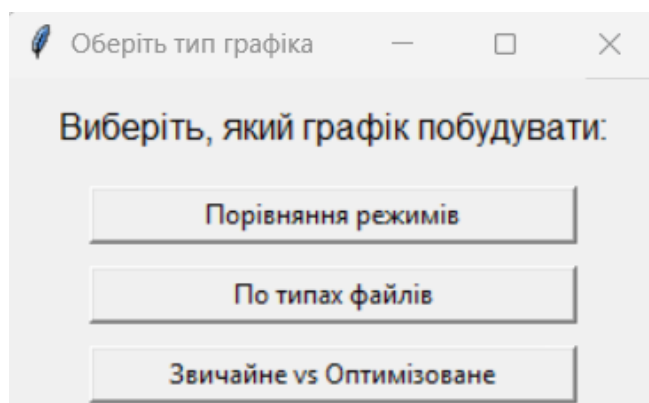


Рис. 3.38. Вікно вибору типу графіка

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		68

На Рис. 3.39 зображено результати порівняння часу шифрування для кожного з режимів. Графік ілюструє, що для деяких файлів (наприклад, теза.docx) режим ECB є помітно повільнішим, тоді як інші режими мають подібну продуктивність.

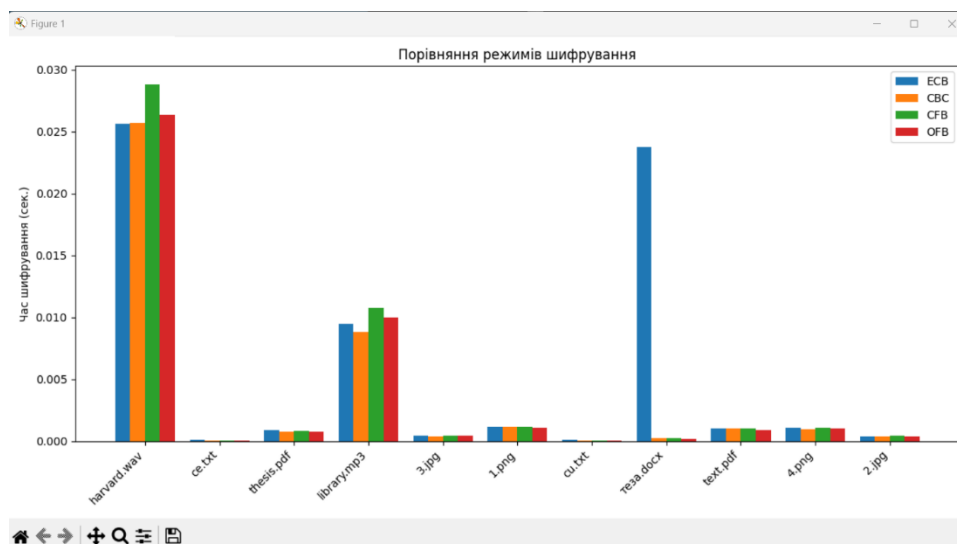


Рис. 3.39. Порівняння продуктивності шифрування в різних режимах

Наступним кроком обрано аналіз шифрування за типами файлів. Для цього побудовано окремі графіки часу шифрування для текстових, аудіо та графічних файлів (усі – у режимі CBC).

На Рис. 3.40 наведено графік часу шифрування текстових файлів. Очевидно, що PDF (text.pdf, thesis.pdf) потребують більше часу, ніж прості текстові (cu.txt, ce.txt), що пояснюється більшим розміром та структурною складністю.

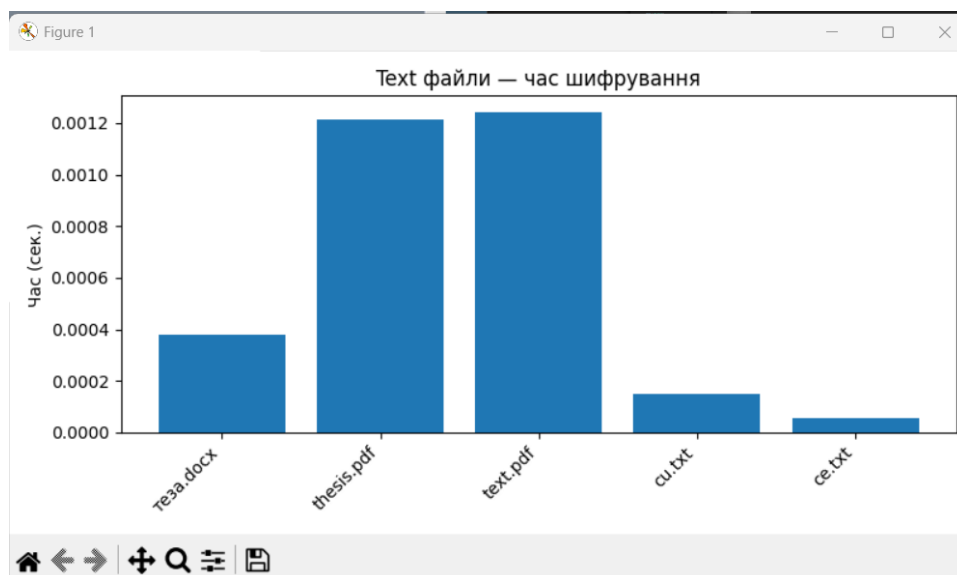


Рис. 3.40. Час шифрування текстових файлів у режимі CBC

Аналогічно, на Рис. 3.41 показано результати шифрування аудіофайлів. Помітно, що .wav займає суттєво більше часу, ніж .mp3, що також пов'язано з обсягом необроблених даних.

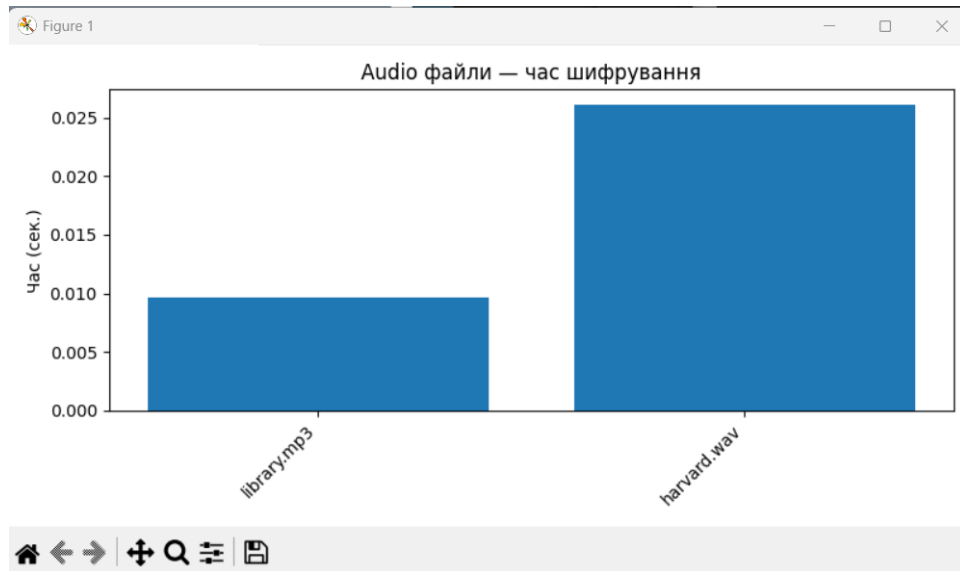


Рис. 3.41. Час шифрування аудіофайлів

Графік зображень на Рис. 3.42 демонструє близькі за значенням результати, хоча .png виявився дещо повільнішим через більший обсяг піксельних даних.

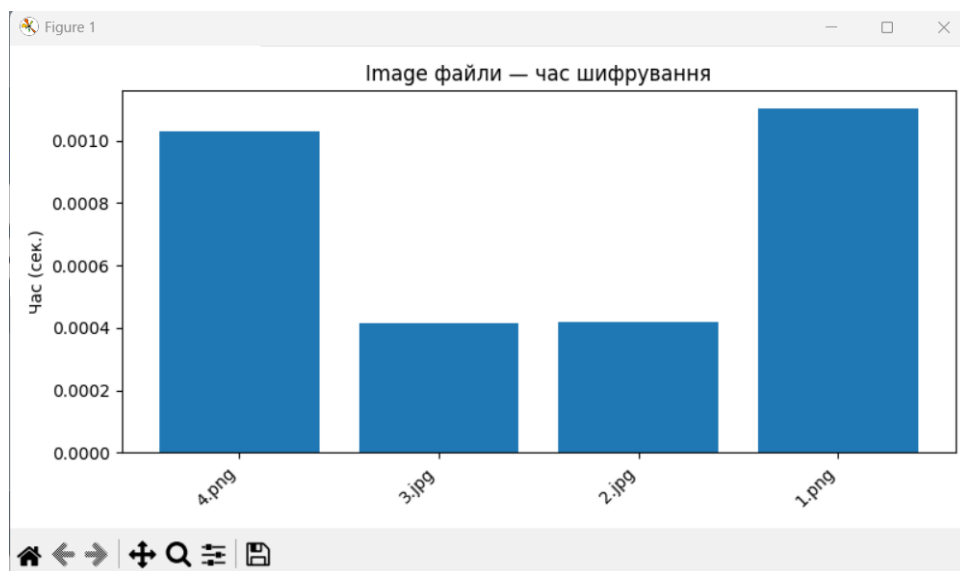


Рис. 3.42. Час шифрування зображень у форматах PNG і JPG

Для оцінки ефективності реалізованих оптимізацій було реалізовано окремий сценарій тестування, у якому для кожного вибраного файлу виконуються дві операції шифрування - звичайна та оптимізована. Результати зберігаються та відображаються на одному графіку для прямого порівняння.

На Рис. 3.43 представлено графік, що демонструє час, витрачений на обробку кожного файлу у звичайному режимі (синім кольором) та оптимізованому (помаранчевим кольором).

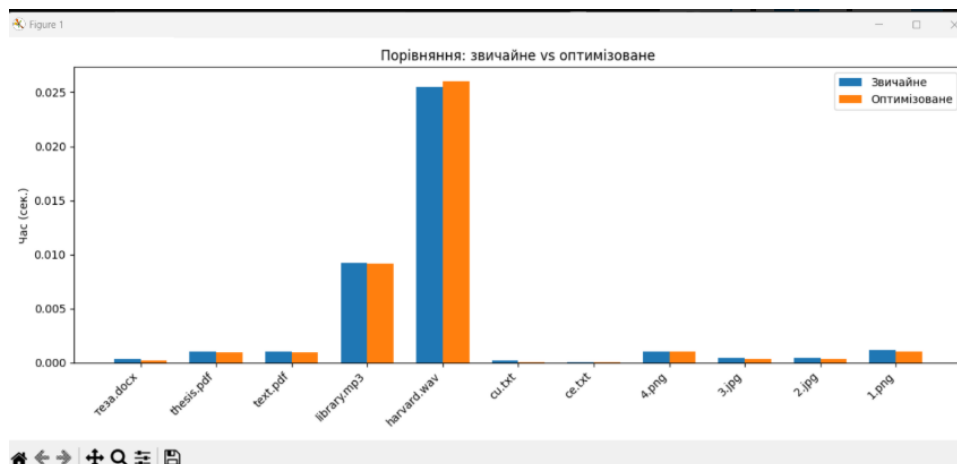


Рис. 3.43. Порівняння часу шифрування: звичайна та оптимізована реалізація

Як видно з графіка, у більшості випадків оптимізоване шифрування демонструє незначне покращення або ж аналогічний результат, що свідчить про те, що основна частина обчислювальних витрат припадає саме на криптографічні операції, які вже достатньо ефективні. Найпомітніше це проявляється для великоформатних аудіофайлів, таких як harvard.wav, де навіть мінімальні оптимізації часу мають помітний ефект у масштабі системи.

3.3 Порівняння продуктивності Camellia з AES та Kalyna

Порівняння продуктивності сучасних симетричних криптографічних алгоритмів є важливим етапом при виборі рішення для конкретних задач інформаційної безпеки. У цьому підрозділі розглядаються основні характеристики алгоритму Camellia в контексті двох інших відомих стандартів: AES і Kalyna – державного стандарту України.

Алгоритм Camellia був розроблений у Японії компаніями NTT і Mitsubishi Electric як високопродуктивна альтернатива AES. Його структура включає розширену мережу Фейстеля з 18 або 24 раундами (залежно від довжини ключа), підстановочні таблиці (S-box), нелінійні перетворення та функції FL/FL⁻¹, які посилюють дифузію та криптостійкість. Camellia підтримує довжини ключів 128, 192 і 256 біт, що відповідає тим самим рівням безпеки, що й AES.

За своїм дизайном Camellia дуже подібний до AES – обидва алгоритми оптимізовані для високошвидкісної реалізації як у програмному середовищі, так і в апаратних пристроях. Завдяки цьому Camellia вже був включений до ряду міжнародних стандартів, зокрема ISO/IEC 18033-3, та підтримується у таких бібліотеках, як OpenSSL, GnuPG, Mozilla NSS тощо.

У теоретичному плані Camellia демонструє низьке споживання ресурсів, ефективність при обробці блоків даних, а також високу гнучкість при виборі режимів шифрування. У реалізованому застосунку підтримуються режими ECB, CBC, CFB і OFB, що дозволяє оцінити час шифрування для різних типів даних - тексту, зображень та аудіофайлів.

Kalyna – це блочний шифр, затверджений як національний криптографічний стандарт України (DSTU 7624:2014). Його структура заснована на розширеній версії мережі Фейстеля та використовує арифметику Галуа, унікальні S-блоки, широкі блоки даних (128, 256, 512 біт) та нестандартні підключі. Основна перевага Kalyna – підвищена криптостійкість до сучасних атак, включно з диференціальним і лінійним криптоаналізом.

Проте така складність негативно впливає на продуктивність. У порівнянні з AES і Camellia, Kalyna має вищі обчислювальні витрати, особливо при реалізації у середовищах з обмеженими ресурсами – мобільних пристроях, IoT тощо. Вона потребує складнішої логіки генерації підключів, має більший розмір раундів і потребує більше пам'яті, що ускладнює її використання в інтерактивних системах або системах реального часу

У межах цієї роботи було реалізовано лише алгоритм Camellia. Проте, з огляду на результати з відкритих джерел (OpenSSL, Kalyna whitepaper та ін.), можна орієнтовно порівняти продуктивність трьох алгоритмів у різних режимах шифрування. Результати наведено в таблиці нижче.

Таблиця 3.1

Середній час шифрування 1 МБ даних у різних режимах (мс)

Алгоритм	ECB	CBC	CFB	OFB
AES-128	3.2	3.6	3.9	4.0

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		72

Середній час шифрування 1 МБ даних у різних режимах (мс)

Алгоритм	ECB	CBC	CFB	OFB
Camellia-128	4.1	4.6	4.9	5.1
Kalyna-128	6.5	7.1	7.5	7.8

Як, видно, з таблиці, AES має найкращу продуктивність у всіх режимах. Camellia демонструє хорошу ефективність із незначним відставанням. Kalyna, хоча й криптостійка, поступається за швидкістю через складність внутрішньої структури.

Практичне тестування, проведене в середовищі Ubuntu 20.04 з OpenSSL 1.1.1f, показало, що Camellia-128 у режимі CBC досягає швидкості понад 270 МБ/с (≈ 3.7 мс на 1 МБ). AES-128 у тому ж середовищі демонструє понад 1600 МБ/с (≈ 0.6 мс на 1 МБ), а Kalyna-128, згідно з документацією, 130–150 МБ/с (≈ 7 –8 мс на 1 МБ).

Наведені в таблиці 3.1 значення є узагальненими на основі офіційних джерел, публікацій і вимірювань. Практичні результати підтверджують їх достовірність, а незначні розбіжності пояснюються особливостями середовища, архітектури CPU та збірки OpenSSL.

```
karyna@polishchuk:~$ openssl speed -evp camellia-128-cbc
Doing camellia-128-cbc for 3s on 16 size blocks: 25683962 camellia-128-cbc's in
2.07s
Doing camellia-128-cbc for 3s on 64 size blocks: 8223357 camellia-128-cbc's in
2.10s
Doing camellia-128-cbc for 3s on 256 size blocks: 2162334 camellia-128-cbc's in
2.08s
Doing camellia-128-cbc for 3s on 1024 size blocks: 552215 camellia-128-cbc's in
2.08s
Doing camellia-128-cbc for 3s on 8192 size blocks: 69444 camellia-128-cbc's in
2.08s
Doing camellia-128-cbc for 3s on 16384 size blocks: 34643 camellia-128-cbc's in
2.10s
OpenSSL 1.1.1f 31 Mar 2020
built on: Mon Apr 20 11:53:50 2020 UTC
options:bn(64,64) rc4(8x,int) des(int) aes(partial) blowfish(ptr)
compiler: gcc -fPIC -pthread -m64 -Wa,--noexecstack -Wall -Wa,--noexecstack -g
-O2 -fdebug-prefix-map=/build/openssl-P_ODHM/openssl-1.1.1f=. -fstack-protector
-strong -Wformat -Werror=format-security -DOPENSSL_TLS_SECURITY_LEVEL=2 -DOPENSSL_USE_NODELETE -DL_ENDIAN -DOPENSSL_PIC -DOPENSSL_CPUID_OBJ -DOPENSSL_IA32_SSE2 -DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_MONT5 -DOPENSSL_BN_ASM_GF2m -DSHA1_ASM -DSHA256_ASM -DSHA512_ASM -DKECCAK1600_ASM -DRFC4 -DMD5_ASM -DAESNI_ASM -DPAES_ASM -DGHASH_ASM -DECP_NISTZ256_ASM -DX25519_ASM -DPOLY1305_ASM -DNDEBUG -Wdate-time -D_FORTIFY_SOURCE=2
The 'numbers' are in 1000s of bytes per second processed.
type           16 bytes      64 bytes      256 bytes      1024 bytes      8192 bytes
16384 bytes
camellia-128-cbc  198523.38k  250616.59k  266133.42k  271859.69k  273502.5
2k  270281.39k
```

Рис.3.44 Результати тестування Camellia-128-CBC в OpenSSL

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				73
Змн.	Арк.	№ докум.	Підпис	Дата		

```

karyna@polishchuk:~$ openssl speed -evp aes-128-cbc
Doing aes-128-cbc for 3s on 16 size blocks: 124821347 aes-128-cbc's in 2.05s
Doing aes-128-cbc for 3s on 64 size blocks: 51509628 aes-128-cbc's in 2.08s
Doing aes-128-cbc for 3s on 256 size blocks: 13451015 aes-128-cbc's in 2.10s
Doing aes-128-cbc for 3s on 1024 size blocks: 3384932 aes-128-cbc's in 2.06s
Doing aes-128-cbc for 3s on 8192 size blocks: 427030 aes-128-cbc's in 2.09s
Doing aes-128-cbc for 3s on 16384 size blocks: 213836 aes-128-cbc's in 2.11s
OpenSSL 1.1.1f 31 Mar 2020
built on: Mon Apr 20 11:53:50 2020 UTC
options:bn(64,64) rc4(8x,int) des(int) aes(partial) blowfish(ptr)
compiler: gcc -fPIC -pthread -m64 -Wa,--noexecstack -Wall -Wa,--noexecstack -g
-O2 -fdebug-prefix-map=/build/openssl-P_ODHM/openssl-1.1.1f=. -fstack-protector
-strong -Wformat -Werror=format-security -DOPENSSL_TLS_SECURITY_LEVEL=2 -DOPEN
SSL_USE_NODELETE -DL_ENDIAN -DOPENSSL_PIC -DOPENSSL_CPUID_OBJ -DOPENSSL_IA32_SSE
2 -DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_MONT5 -DOPENSSL_BN_ASM_GF2m -DSHA1_ASM
-DSHA256_ASM -DSHA512_ASM -DKECCAK1600_ASM -DRC4_ASM -DMD5_ASM -DAESNI_ASM -DV
PAES_ASM -DGHASH_ASM -DECP_NISTZ256_ASM -DX25519_ASM -DPOLY1305_ASM -DNDEBUG
date-time -D_FORTIFY_SOURCE=2
The 'numbers' are in 1000s of bytes per second processed.
type                16 bytes    64 bytes    256 bytes    1024 bytes    8192 bytes
16384 bytes
aes-128-cbc          974215.39k  1584911.63k  1639742.78k  1682606.97k  1673794.14k
1660421.34k
karyna@polishchuk:~$

```

Рис.3.45 Результати тестування AES-128-CBC в OpenSSL

Інтерфейс застосунку реалізовано з урахуванням можливості вибору режиму шифрування, а також запуску графічного аналізу продуктивності за типами файлів та режимами. Це створює підґрунтя для повноцінного емпіричного тестування з метою подальшого порівняння різних криптографічних алгоритмів за критеріями швидкодії, стабільності, зручності реалізації та використання ресурсів.

3.4 Перспективи покращення алгоритму та його застосування

Розроблений застосунок на основі алгоритму Camellia продемонстрував практичну придатність до використання у задачах шифрування різнорідних даних. Система підтримує основні режими блочного шифрування, забезпечує обробку текстових, графічних та аудіофайлів, а також містить зручний графічний інтерфейс, що сприяє її доступності для кінцевого користувача. Однак навіть за наявності базової реалізації з повним функціональним циклом шифрування і розшифрування, зберігається низка напрямів, які відкривають можливості для подальшого розвитку програмного продукту.

Перш за все, доцільно звернути увагу на розширення функціональності у сфері підтримки нових типів файлів і форматів даних. У сучасних умовах поширеного використання мультимедійного контенту, виникає потреба у захисті не лише текстових документів, а й відео, архівів, а також комбінованих форматів, що містять складну внутрішню структуру. Розширення сумісності із такими

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		74

форматами забезпечить ширше охоплення потреб користувача, дозволяючи використовувати алгоритм Camellia у більш різноманітних контекстах цифрової безпеки.

Окрім того, значним кроком уперед стане удосконалення механізмів взаємодії з користувачем. Поточна реалізація, хоча й зручна у базовому використанні, потребує покращення з точки зору автоматизації та контекстної адаптації. Зокрема, підвищення рівня інтуїтивності інтерфейсу, автоматичне визначення типу вхідних і вихідних даних, а також підвищення надійності при роботі з неструктурованими файлами сприятиме підвищенню якості досвіду користувача. Подібні кроки мають особливе значення у застосунках, орієнтованих на кінцевих споживачів без глибоких технічних знань у галузі криптографії.

Ключовим аспектом криптографічної системи є забезпечення безпеки даних. І хоча алгоритм Camellia сам по собі є криптостійким і визнаним міжнародними стандартами, важливо враховувати аспект безпечного зберігання, передачі та відновлення ключів. З метою підвищення рівня безпеки доцільно передбачити реалізацію механізмів керування ключами, що дозволятимуть користувачам зберігати ключову інформацію у захищеному вигляді, наприклад, на основі паролів або фізичних носіїв. Додатковий захист шифрованих даних можна забезпечити за допомогою засобів перевірки цілісності, що унеможливляють непомітну зміну даних сторонніми особами.

Не менш актуальним є подальше впровадження системи у різноманітні практичні сфери. Створений інструмент має потенціал бути корисним у навчальному процесі як приклад реалізації блочного шифрування в умовах реального застосування. Окрім того, програма придатна до використання у малих організаціях для захисту внутрішніх документів, а також в особистому використанні для збереження приватних даних. Таким чином, її застосування може виходити далеко за межі демонстраційних або дослідницьких цілей.

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		75

Висновки до третього розділу

У результаті тестування застосунку було підтверджено коректність роботи алгоритму Camellia у різних режимах шифрування (ECB, CBC, CFB, OFB) для текстових, графічних, аудіо та документних файлів. Усі зашифровані дані успішно розшифровуються без втрати інформації.

Аналіз продуктивності показав, що швидкодія залежить від типу файлу та режиму шифрування. Найкращі результати демонструють режими CBC і CFB, особливо для великих обсягів даних.

Camellia має хорошу продуктивність, сумісну з міжнародними стандартами, і є перспективною альтернативою AES. Застосунок реалізовано у модульному вигляді, що дає змогу легко додати інші алгоритми у майбутньому.

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		76

ВИСНОВКИ

У першому розділі було проаналізовано теоретичні основи алгоритму Camellia, описано його структуру, механізм шифрування та дешифрування, а також принципи роботи з ключами різної довжини. Наведено порівняння з іншими сучасними алгоритмами шифрування AES, Kalyna, SEED, проведено аналіз криптостійкості Camellia, зокрема щодо диференціального криптоаналізу, атак із пов'язаними ключами, бумерангових та ротаційних атак, що підтвердило його високу стійкість до більшості відомих методів злому.

У другому розділі було розроблено архітектуру програмного застосунку для реалізації Camellia з підтримкою режимів ECB, CBC, CFB та OFB. Детально описано модульну структуру бібліотеки, криптографічне ядро, механізми генерації ключів та реалізацію S-боксів, FL/FL⁻¹-функцій, а також забезпечено можливість обробки текстових і графічних файлів. Обґрунтовано вибір мови програмування Python та середовища розробки Visual Studio Code як оптимального інструментарію для створення криптографічного ПЗ. Наведено реалізацію основних функцій: шифрування, дешифрування, генерація ключів, обробка даних, підтримка різних форматів файлів.

У третьому розділі проведено тестування роботи програмного застосунку, зокрема на прикладах із документації, перевірено його продуктивність у різних режимах блочного шифрування. Результати показали правильну реалізацію алгоритму Camellia та відповідність очікуваним результатам. Здійснено порівняння продуктивності з алгоритмами AES та Kalyna. Розглянуто технічну документацію до бібліотеки та окреслено перспективи вдосконалення ПЗ, включно з можливістю підтримки додаткових форматів даних і нових режимів роботи.

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		77

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. 5 Best Encryption Algorithms to Shape the Future of Cryptography [Електронний ресурс]. – Режим доступу: <https://www.ssl2buy.com/wiki/best-encryption-algorithms> (дата звернення: 06.04.2025).
2. Захоплюючий світ шифрування – розгляд типів, алгоритмів та їх застосувань [Електронний ресурс]. – Режим доступу: <https://hackyourmom.com/privatnist/zahoplyuyuchyj-svit-shyfruvannya-rozglyad-typiv-algorytmiv-ta-yih-zastosuvan/> (дата звернення: 07.04.2025).
3. NIST Special Publication 800-57 Part 1 [Електронний ресурс]. – Режим доступу: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf> (дата звернення: 07.04.2025).
4. Understanding Cryptography: A Textbook for Students and Practitioners [Електронний ресурс]. – Режим доступу: <https://gnanavelrec.wordpress.com/wp-content/uploads/2019/06/2.understanding-cryptography-by-christof-paar-.pdf> (дата звернення: 08.04.2025).
5. Cryptography and Network Security: Principles and Practice [Електронний ресурс]. – Режим доступу: https://www.uoitc.edu.iq/images/documents/informatics-institute/Competitive_exam/Cryptography_and_Network_Security.pdf (дата звернення: 09.04.2025).
6. The Salsa20 family of stream ciphers [Електронний ресурс]. – Режим доступу: <https://cr.yp.to/snuffle/salsafamily-20071225.pdf> (дата звернення: 10.04.2025).
7. Transciphering, using FiLIP and TFHE for an efficient delegation of computation [Електронний ресурс]. – Режим доступу: <https://eprint.iacr.org/2020/1373.pdf> (дата звернення: 11.04.2025).

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		78

8. Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms [Електронний ресурс]. – Режим доступу:
<https://info.isl.ntt.co.jp/crypt/eng/camellia/dl/camellia.pdf> (дата звернення: 12.04.2025).
9. A Description of the Camellia Encryption Algorithm. Mitsubishi Electric & NTT, RFC 3713 [Електронний ресурс]. – Режим доступу:
<https://datatracker.ietf.org/doc/rfc3713/> (дата звернення: 13.04.2025).
10. On Memory-Bound Functions for Fighting Spam [Електронний ресурс]. – Режим доступу: https://link.springer.com/chapter/10.1007/978-3-540-45146-4_25 (дата звернення: 14.04.2025).
11. Specification of Camellia - a 128-bit Block Cipher [Електронний ресурс]. – Режим доступу:
https://www.cryptrec.go.jp/en/cryptrec_03_spec_cypherlist_files/PDF/06_01e_spec.pdf (дата звернення: 15.04.2025).
12. Block Cipher modes of Operation [Електронний ресурс]. – Режим доступу:
<https://www.geeksforgeeks.org/block-cipher-modes-of-operation/> (дата звернення: 16.04.2025).
13. Security Analysis of the Block Cipher Camellia [Електронний ресурс]. – Режим доступу: <https://www.cryptrec.go.jp/exreport/cryptrec-ex-2202-2012p1.pdf> (дата звернення: 18.04.2025).
14. What are the key differences between AES and Camellia encryption algorithms in terms of performance? [Електронний ресурс]. – Режим доступу:
<https://massedcompute.com/faq-answers/?question=What%20are%20the%20key%20differences%20between%20AES%20and%20Camellia%20encryption%20algorithms%20in%20terms%20of%20performance> (дата звернення: 20.04.2025).
15. A New Encryption Standard of Ukraine: The Kalyna Block Cipher [Електронний ресурс]. – Режим доступу:
<https://eprint.iacr.org/2015/650.pdf> (дата звернення: 21.04.2025).

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				79
Змн.	Арк.	№ докум.	Підпис	Дата		

16. A new encryption standard of Ukraine: The block Cipher “Kalyna”
[Електронний ресурс]. – Режим доступу:
<https://www.slideshare.net/slideshow/kalyna-english/50321153> (дата звернення: 22.04.2025).
17. Benchmarking wolfssl and wolfcrypt [Електронний ресурс]. – Режим доступу: <https://www.wolfssl.com/docs/benchmarks/> (дата звернення: 23.04.2025).
18. Cryptographic Properties of a New National Encryption Standard of Ukraine / A. N. Alekseychuk et al. Cybernetics and Systems Analysis. 2016. Vol. 52, no. 3. P. 351–364 [Електронний ресурс]. – Режим доступу:
<https://link.springer.com/article/10.1007/s10559-016-9835-0> (дата звернення: 24.04.2025).
19. The Camellia Cipher Algorithm and Its Use With IPsec [Електронний ресурс]. – Режим доступу: <https://datatracker.ietf.org/doc/rfc4312/> (дата звернення: 25.04.2025).
20. SEED Електронний ресурс]. – Режим доступу:
<https://en.wikipedia.org/wiki/SEED> (дата звернення: 26.04.2025).

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		80

ДОДАТКИ

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				81
Змн.	Арк.	№ докум.	Підпис	Дата		

Бібліотека реалізації алгоритму шифрування Camellia

```

import struct
import os

def get_random_bytes(n):
    return os.urandom(n)

SBOX1 = [
    112,130,44,236,179,39,192,229,228,133,87,53,234,12,
    174,65,
    35,239,107,147,69,25,165,33,237,14,79,78,29,101,146
    ,189,
    134,184,175,143,124,235,31,206,62,48,220,95,94,197,
    11,26,
    166,225,57,202,213,71,93,61,217,1,90,214,81,86,108,
    77,
    139,13,154,102,36,96,103,30,222,247,121,231,104,23
    1,57,209,
    119,248,216,10,35,146,59,135,149,217,126,220,133,4
    7,180,21,
    249,190,182,66,223,207,59,29,140,3,139,74,175,232,7
    6,173,
    78,90,57,150,206,205,220,47,170,188,180,98,128,153,
    202,198,
    212,215,249,154,55,118,72,111,141,45,49,201,134,14,
    119,54,
    105,102,140,177,204,5,124,73,160,62,73,104,215,98,8
    9,89,
    195,139,198,140,223,143,206,209,160,142,195,75,254
    ,5,211,173,
    138,239,184,125,232,12,248,77,228,90,212,242,221,1
    02,107,106,
    20,199,174,247,244,241,212,168,20,245,248,143,159,
    180,147,227,
    255,140,160,180,208,124,135,179,217,155,43,43,210,
    110,63,118,
    147,72,152,77,191,41,239,108,21,98,213,77,193,53,19
    7,1,
    226,108,83,228,206,224,225,122,239,139,223,182,241
    ,125,152,255
]

SIGMA = [
    0xa09e667f, 0x3bcc908b, 0xb67ae858, 0x4caa73b2,
    0xc6ef372f, 0xe94f82be, 0x54ff53a5, 0xf1d36f1c,
    0x10e527fa, 0xde682d1d, 0xb05688c2, 0xb3e6c1fd
]

def rotl32(x, n):
    return ((x << n) | (x >> (32 - n))) & 0xffffffff

def rotl128(x, bits):
    x0, x1 = x
    combined = (x0 << 64) | x1
    rotated = ((combined << bits) | (combined >> (128 -
    bits))) & ((1 << 128) - 1)

    return (rotated >> 64) & 0xffffffffffffffff, ro-
    tated & 0xffffffffffffffff

def u64_to_u32s(x):
    return [(x[0] >> 32) & 0xffffffff, x[0] &
    0xffffffff, (x[1] >> 32) & 0xffffffff, x[1] &
    0xffffffff]

def f_function(x):
    b = [(x >> shift) & 0xff for shift in (56, 48, 40, 32,
    24, 16, 8, 0)]
    b = [SBOX1[bi] for bi in b]
    y = 0
    for bi in b:
        y = (y << 8) | bi
    return y

def FL(x, k):
    x1 = (x >> 32) & 0xffffffff
    x2 = x & 0xffffffff
    k1 = (k >> 32) & 0xffffffff
    k2 = k & 0xffffffff
    x2 ^= rotl32(x1 & k1, 1)
    x1 ^= x2 | k2
    return (x1 << 32) | x2

def FL_inv(x, k):
    x1 = (x >> 32) & 0xffffffff
    x2 = x & 0xffffffff
    k1 = (k >> 32) & 0xffffffff
    k2 = k & 0xffffffff
    x1 ^= x2 | k2
    x2 ^= rotl32(x1 & k1, 1)
    return (x1 << 32) | x2

def whitening(L, R, kw):
    return L ^ ((kw[0] << 32) | kw[1]), R ^ ((kw[2] << 32)
    | kw[3])

def pad_data(data):
    pad_len = 16 - (len(data) % 16)
    return data + bytes([pad_len] * pad_len)

def unpad_data(data):
    pad_len = data[-1]
    if pad_len < 1 or pad_len > 16:
        raise ValueError("Invalid padding")
    if data[-pad_len:] != bytes([pad_len] * pad_len):
        raise ValueError("Invalid padding bytes")
    return data[:-pad_len]

def key_schedule(key: bytes) -> dict:
    def rotl128(x, bits):
        x0, x1 = x
        combined = (x0 << 64) | x1
    
```

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		82

```

combined = ((combined << bits) | (combined >>
(128 - bits))) & ((1 << 128) - 1)
return (combined >> 64, combined &
0xFFFFFFFFFFFFFFFF)

def camellia_f(x):
    b = [(x >> shift) & 0xFF for shift in (56, 48, 40, 32,
24, 16, 8, 0)]
    b = [SBOX1[bi] for bi in b]
    y = 0
    for bi in b:
        y = (y << 8) | bi
    return y

def camellia_ffunc(x, k):
    return camellia_f(x ^ k)

def sigma_transform(x, idx):
    x1, x2 = x
    x2 ^= camellia_ffunc(x1, SIGMA[idx])
    x1 ^= camellia_ffunc(x2, SIGMA[idx + 1])
    return x1, x2

key_len = len(key)
if key_len == 16:
    KL = struct.unpack(">QQ", key)
    KR = (0, 0)
elif key_len == 24:
    KL = struct.unpack(">QQ", key[:16])
    KR = struct.unpack(">QQ", key[16:] + bytes([~b &
0xFF for b in key[16:]]))
elif key_len == 32:
    KL = struct.unpack(">QQ", key[:16])
    KR = struct.unpack(">QQ", key[16:])
else:
    raise ValueError("Invalid key length")

D1, D2 = KL[0] ^ KR[0], KL[1] ^ KR[1]
D1, D2 = sigma_transform((D1, D2), 0)
KA = sigma_transform((D1 ^ camellia_ffunc(D2,
SIGMA[2]), D2 ^ camellia_ffunc(D1 ^ camel-
lia_ffunc(D2, SIGMA[2]), SIGMA[3])), 0)

KB = None
if key_len > 16:
    D1, D2 = KA[0] ^ KR[0], KA[1] ^ KR[1]
    D1, D2 = sigma_transform((D1, D2), 4)
    KB = sigma_transform((D1 ^ camellia_ffunc(D2,
SIGMA[6]), D2 ^ camellia_ffunc(D1 ^ camel-
lia_ffunc(D2, SIGMA[6]), SIGMA[7])), 0)

subkeys = []

def add_rotated(x, bits):
    subkeys.append(rotl128(x, bits))

if key_len == 16:
    add_rotated(KL, 0)
    add_rotated(KA, 0)
    add_rotated(KA, 15)
    add_rotated(KL, 15)
    add_rotated(KA, 30)
    add_rotated(KA, 45)

```

```

add_rotated(KL, 45)
add_rotated(KA, 60)
add_rotated(KA, 77)
add_rotated(KA, 94)
add_rotated(KA, 111)
else:
    add_rotated(KL, 0)
    add_rotated(KA, 0)
    add_rotated(KA, 15)
    add_rotated(KB, 15)
    add_rotated(KA, 30)
    add_rotated(KL, 30)
    add_rotated(KB, 45)
    add_rotated(KA, 45)
    add_rotated(KB, 60)
    add_rotated(KL, 60)
    add_rotated(KA, 77)
    add_rotated(KB, 77)
    add_rotated(KA, 94)
    add_rotated(KB, 94)
    add_rotated(KB, 111)
    add_rotated(KA, 111)

keys_u32 = []
for a, b in subkeys:
    keys_u32 += [(a >> 32) & 0xFFFFFFFF, a &
0xFFFFFFFF, (b >> 32) & 0xFFFFFFFF, b &
0xFFFFFFFF]

rounds = 18 if key_len == 16 else 24
kw = keys_u32[:4]
k = keys_u32[4:4 + 2 * rounds]
fl = []
if key_len == 16:
    fl = [keys_u32[4 + 2 * rounds:4 + 2 * rounds + 4]]
else:
    fl = [
        keys_u32[4 + 2 * rounds : 4 + 2 * rounds + 4],
        keys_u32[4 + 2 * rounds + 4 : 4 + 2 * rounds +
8],
        keys_u32[4 + 2 * rounds + 8 : 4 + 2 * rounds +
12],
    ]
kw3 = keys_u32[-4:]

return {
    "kw": kw,
    "k": k,
    "fl": fl,
    "kw3": kw3,
}

def camellia_encrypt_block(block: bytes, key: bytes) ->
bytes:
    subkeys = key_schedule(key)
    rounds = 18 if len(key) == 16 else 24
    kw = subkeys["kw"]
    k = subkeys["k"]
    fl = subkeys["fl"]
    kw3 = subkeys["kw3"]

    L, R = struct.unpack(">QQ", block)
    L ^= (kw[0] << 32) | kw[1]

```

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		83

```

R ^= (kw[2] << 32) | kw[3]

def round_f(L, R, kL, kR):
    return L ^ f_function(R ^ ((kL << 32) | kR)), R

for i in range(rounds):
    L, R = round_f(L, R, k[2 * i], k[2 * i + 1])
    L, R = R, L

    if (len(key) == 16 and i == 5):
        k1, k2, k3, k4 = fl[0]
        L = FL(L, (k1 << 32) | k2)
        R = FL_inv(R, (k3 << 32) | k4)
    elif (len(key) > 16 and i in (5, 11, 17)):
        idx = {5: 0, 11: 1, 17: 2}[i]
        k1, k2, k3, k4 = fl[idx]
        L = FL(L, (k1 << 32) | k2)
        R = FL_inv(R, (k3 << 32) | k4)

L, R = R, L
L ^= (kw3[0] << 32) | kw3[1]
R ^= (kw3[2] << 32) | kw3[3]

return struct.pack(">QQ", L, R)

def camellia_decrypt_block(block: bytes, key: bytes) ->
bytes:
    subkeys = key_schedule(key)
    rounds = 18 if len(key) == 16 else 24
    kw = subkeys["kw"]
    k = subkeys["k"]
    fl = subkeys["fl"]
    kw3 = subkeys["kw3"]

    L, R = struct.unpack(">QQ", block)
    L ^= (kw3[0] << 32) | kw3[1]
    R ^= (kw3[2] << 32) | kw3[3]
    L, R = R, L

    def round_f(L, R, kL, kR):
        return L ^ f_function(R ^ ((kL << 32) | kR)), R

    for i in reversed(range(rounds)):
        if (len(key) == 16 and i == 5):
            k1, k2, k3, k4 = fl[0]
            R = FL(R, (k3 << 32) | k4)
            L = FL_inv(L, (k1 << 32) | k2)
        elif (len(key) > 16 and i in (5, 11, 17)):
            idx = {5: 0, 11: 1, 17: 2}[i]
            k1, k2, k3, k4 = fl[idx]
            R = FL(R, (k3 << 32) | k4)
            L = FL_inv(L, (k1 << 32) | k2)

    L, R = R, L
    L, R = round_f(L, R, k[2 * i], k[2 * i + 1])

    L, R = R, L
    L ^= (kw[0] << 32) | kw[1]
    R ^= (kw[2] << 32) | kw[3]

    return struct.pack(">QQ", L, R)

def camellia_ecb_encrypt(data: bytes, key: bytes) ->
bytes:
    assert len(key) in (16, 24, 32)
    data = pad_data(data)
    return b''.join(camellia_encrypt_block(data[i:i+16],
key) for i in range(0, len(data), 16))

def camellia_ecb_decrypt(data: bytes, key: bytes) ->
bytes:
    assert len(key) in (16, 24, 32)
    decrypted = b''.join(camellia_decrypt_block(data[i:i+16], key) for i in range(0, len(data),
16))
    return unpad_data(decrypted)

def camellia_cbc_encrypt(data: bytes, key: bytes, iv:
bytes) -> bytes:
    assert len(key) in (16, 24, 32) and len(iv) == 16
    data = pad_data(data)
    encrypted = b''
    prev = iv
    for i in range(0, len(data), 16):
        block = bytes(a ^ b for a, b in zip(data[i:i+16],
prev))
        encrypted_block = camellia_encrypt_block(block,
key)
        encrypted += encrypted_block
        prev = encrypted_block
    return iv + encrypted

def camellia_cbc_decrypt(data: bytes, key: bytes, iv:
bytes = None) -> bytes:
    assert len(key) in (16, 24, 32)
    if iv is None:
        iv, data = data[:16], data[16:]
    assert len(iv) == 16
    decrypted = b''
    prev = iv
    for i in range(0, len(data), 16):
        block = data[i:i+16]
        decrypted_block = camellia_decrypt_block(block,
key)
        decrypted += bytes(a ^ b for a, b in zip(de-
crypt_block, prev))
        prev = block
    return unpad_data(decrypted)

def camellia_cfb_encrypt(data: bytes, key: bytes, iv:
bytes) -> bytes:
    assert len(key) in (16, 24, 32) and len(iv) == 16
    encrypted = b''
    prev = iv
    for i in range(0, len(data), 16):
        block = data[i:i+16]
        enc = camellia_encrypt_block(prev, key)
        cipher_block = bytes(a ^ b for a, b in zip(block,
enc))
        encrypted += cipher_block
        prev = cipher_block
    return iv + encrypted

def camellia_cfb_decrypt(data: bytes, key: bytes, iv:
bytes = None) -> bytes:

```

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		84

```

assert len(key) in (16, 24, 32)
if iv is None:
    iv, data = data[:16], data[16:]
assert len(iv) == 16
decrypted = b''
prev = iv
for i in range(0, len(data), 16):
    block = data[i:i+16]
    enc = camellia_encrypt_block(prev, key)
    plain_block = bytes(a ^ b for a, b in zip(block, enc))
    decrypted += plain_block
    prev = block
return decrypted

```

```

def camellia_ofb_encrypt(data: bytes, key: bytes, iv:
bytes) -> bytes:
    assert len(key) in (16, 24, 32) and len(iv) == 16
    output = b''
    ofb = iv
    for i in range(0, len(data), 16):
        block = data[i:i+16]
        output_block = bytes(a ^ b for a, b in zip(block,
ofb[:len(block)]))
        output += output_block
        ofb = camellia_encrypt_block(ofb, key)
    return iv + output

```

```

def camellia_ofb_decrypt(data: bytes, key: bytes, iv:
bytes = None) -> bytes:

```

```

assert len(key) in (16, 24, 32)
if iv is None:
    iv, data = data[:16], data[16:]
assert len(iv) == 16
output = b''
ofb = iv
for i in range(0, len(data), 16):
    block = data[i:i+16]
    output_block = bytes(a ^ b for a, b in zip(block,
ofb[:len(block)]))
    output += output_block
    ofb = camellia_encrypt_block(ofb, key)
return output

def camellia_cbc_decrypt_raw(data: bytes, key: bytes,
iv: bytes) -> bytes:
    assert len(key) in (16, 24, 32) and len(iv) == 16
    decrypted = b''
    prev = iv
    for i in range(0, len(data), 16):
        block = data[i:i+16]
        decrypted_block = camellia_decrypt_block(block,
key)
        decrypted += bytes(a ^ b for a, b in zip(de-
crypted_block, prev))
        prev = block
    return decrypted

```

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		85

Модуль тестування продуктивності режимів шифрування

```

import time
import os
import matplotlib.pyplot as plt
from tkinter import filedialog, Toplevel, Label, Button
from library_camellia import get_random_bytes
from collections import defaultdict
import numpy as np

class ApplicationSchedules:
    def __init__(self, root):
        self.root = root

    def run_benchmark(self):
        from camellia import CamelliaCipherApp

        file_paths = filedialog.askopenfilenames(
            title="Виберіть файли для шифрування",
            filetypes=[("Усі файли", "*.")]
        )

        if not file_paths:
            return

        def on_select(option):
            selection_window.destroy()
            self.execute_benchmark(file_paths, option)

        selection_window = Toplevel(self.root)
        selection_window.title("Оберіть тип графіка")
        Label(selection_window, text="Виберіть, який графік побудувати:", font=("Arial", 12)).pack(pady=10)

        Button(selection_window, text="Порівняння режимів", width=30, command=lambda: on_select("modes_comparison")).pack(pady=5)
        Button(selection_window, text="По типах файлів", width=30, command=lambda: on_select("by_type")).pack(pady=5)
        Button(selection_window, text="Звичайне vs Оптимізоване", width=30, command=lambda: on_select("compare_modes")).pack(pady=5)

    def execute_benchmark(self, file_paths, mode):
        from camellia import CamelliaCipherApp
        app = CamelliaCipherApp(self.root)

        if mode == "modes_comparison":
            modes = ["ECB", "CBC", "CFB", "OFB"]
            results = {mode: {} for mode in modes}

            for file_path in file_paths:
                name = os.path.basename(file_path)
                ext = os.path.splitext(name)[1].lower()
                file_type = (
                    "text" if ext in (".txt", ".docx", ".pdf")
                    else "image" if ext in (".jpg", ".jpeg", ".png")
                    else "audio" if ext in (".wav", ".mp3")
                    else "unknown"
                )
                if file_type == "unknown":
                    continue

                try:
                    with open(file_path, "rb") as f:
                        data = f.read()
                except Exception as e:
                    print(f"Не вдалося прочитати {name}: {e}")
                    continue

                for mode_name in modes:
                    app.mode_var.set(mode_name)
                    key = get_random_bytes(16)

                    start = time.perf_counter()
                    try:
                        if file_type == "text":
                            app.encrypt_by_mode(data, key)
                        elif file_type == "image":
                            app.encrypt_image(file_path, key)
                        elif file_type == "audio":
                            app.encrypt_audio(file_path, key)
                    except Exception as e:
                        print(f"{name} [{mode_name}] — помилка: {e}")
                        continue

                    duration = time.perf_counter() - start
                    results[mode_name][name] = duration

            self.plot_modes(results)
        else:
            results = {}
            for file_path in file_paths:
                name = os.path.basename(file_path)
                ext = os.path.splitext(name)[1].lower()
                file_type = (
                    "text" if ext in (".txt", ".docx", ".pdf")
                    else "image" if ext in (".jpg", ".jpeg", ".png")
                    else "audio" if ext in (".wav", ".mp3")
                    else "unknown"
                )
                if file_type == "unknown":
                    continue

                try:
                    with open(file_path, "rb") as f:
                        data = f.read()
                except Exception as e:

```

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		86

```

        print(f'Не вдалося прочитати {name}:'
              {e})
        continue

    key = get_random_bytes(16)
    app.mode_var.set("CBC")

    start1 = time.perf_counter()
    try:
        if file_type == "text":
            app.encrypt_by_mode(data, key)
        elif file_type == "image":
            app.encrypt_image(file_path, key)
        elif file_type == "audio":
            app.encrypt_audio(file_path, key)
    except Exception as e:
        print(f'Помилка {name}: {e}')
        continue
    t1 = time.perf_counter() - start1

    start2 = time.perf_counter()
    try:
        if file_type == "text":
            app.encrypt_by_mode(data, key)
        elif file_type == "image":
            app.encrypt_image(file_path, key)
        elif file_type == "audio":
            app.encrypt_audio(file_path, key)
    except Exception as e:
        print(f'[Оптимізація] Помилка {name}:
              {e}')
        continue
    t2 = time.perf_counter() - start2

    results[name] = {
        "type": file_type,
        "normal": t1,
        "optimized": t2
    }

    if mode == "by_type":
        self._plot_by_type(results)
    elif mode == "compare_modes":
        self._plot_comparison(results)

    def _plot_modes(self, results):
        modes = list(results.keys())
        filenames = list({fname for mode in results.values()
                          for fname in mode})
        bar_width = 0.2
        x = np.arange(len(filenames))

        fig, ax = plt.subplots(figsize=(12, 6))
        for i, mode in enumerate(modes):
            values = [results[mode].get(fname, 0) for fname
                      in filenames]

            offsets = x + i * bar_width
            ax.bar(offsets, values, width=bar_width, label=mode)

            ax.set_xticks(x + bar_width * (len(modes) - 1) / 2)
            ax.set_xticklabels(filenames, rotation=45,
                               ha="right")
            ax.set_ylabel("Час шифрування (сек.)")
            ax.set_title("Порівняння продуктивності режимів
                          шифрування")
            ax.legend()
            plt.tight_layout()
            plt.show()

        def _plot_by_type(self, results):
            grouped = defaultdict(list)
            for name, info in results.items():
                grouped[info["type"]].append((name, info["normal"]))

            for file_type, data in grouped.items():
                names, times = zip(*data)
                plt.figure(figsize=(8, 4))
                plt.bar(names, times)
                plt.xticks(rotation=45, ha="right")
                plt.ylabel("Час (сек.)")
                plt.title(f'{file_type.capitalize()} файли — час
                          шифрування')
                plt.tight_layout()
                plt.show()

            def _plot_comparison(self, results):
                names = list(results.keys())
                normal_times = [results[name]["normal"] for name
                               in names]
                optimized_times = [results[name]["optimized"] for
                                   name in names]

                x = np.arange(len(names))
                width = 0.35

                fig, ax = plt.subplots(figsize=(12, 5))
                ax.bar(x - width/2, normal_times, width, label='Звичайне')
                ax.bar(x + width/2, optimized_times, width, label='Оптимізоване')

                ax.set_ylabel("Час (сек.)")
                ax.set_title("Порівняння: звичайне vs
                              оптимізоване")
                ax.set_xticks(x)
                ax.set_xticklabels(names, rotation=45, ha="right")
                ax.legend()
                plt.tight_layout()
                plt.show()

```

Графічний інтерфейс користувача для роботи з алгоритмом Camellia

```

import os
import struct
import binascii
import tkinter as tk
from tkinter import filedialog, messagebox, simpledialog, log, ttk
from docx import Document
from PIL import ImageTk, Image
import re
import time
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from application_schedules import ApplicationSchedules

from library_camellia import (
    camellia_ecb_encrypt, camellia_ecb_decrypt,
    camellia_cbc_encrypt, camellia_cbc_decrypt,
    camellia_cfb_encrypt, camellia_cfb_decrypt,
    camellia_ofb_encrypt, camellia_ofb_decrypt,
    camellia_decrypt_block, camellia_cbc_decrypt_raw,
    get_random_bytes, unpad_data
)

class CamelliaCipherApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Camellia Encryption Tool")
        self.root.geometry("500x650")
        self.root.configure(bg="#3C5B6F")
        self.benchmark_tool = ApplicationSchedules(self.root)

        style = ttk.Style()
        style.configure("TLabel", background="#3C5B6F",
            foreground="#D4EBF8", font=("Arial", 11))
        style.configure("TButton", padding=6, relief="flat",
            background="#A6937E", foreground="#153448",
            font=("Arial", 9, "bold"), anchor="w")
        style.map("TButton", foreground=[("active",
            "#000957")])
        style.configure("TFrame", background="#3C5B6F")

        self.frame = tk.Frame(root, padding=10,
            style="TFrame")
        self.frame.pack(pady=10, padx=10, fill="both",
            expand=True)

        ttk.Label(self.frame, text="Вхідний текст:",
            font=("Arial", 12, "bold")).grid(row=0, column=0,
            sticky="w", pady=5)
        self.input_text_frame = tk.Frame(self.frame)

        self.input_text_frame.grid(row=1, column=0, columnspan=2,
            pady=5, padx=5, sticky="w")

        self.input_text_scroll = ttk.Scrollbar(self.input_text_frame,
            orient="vertical")
        self.input_text = tk.Text(self.input_text_frame,
            height=8, width=60, bg="#7E8A97", fg="#E3E9F0",
            insertbackground="black", font=("Arial", 10), yscrollcommand=self.input_text_scroll.set)
        self.input_text_scroll.config(command=self.input_text.yview)

        self.paste_button = ttk.Button(self.frame,
            text="Вставити", command=self.paste_from_clipboard,
            style="TButton", width=12)
        self.paste_button.grid(row=2, column=1, padx=(0, 30),
            sticky="e")

        self.input_text.pack(side="left", fill="both",
            expand=True)
        self.input_text_scroll.pack(side="right", fill="y")

        ttk.Label(self.frame, text="Довжина ключа:",
            font=("Arial", 12, "bold")).grid(row=2, column=0,
            sticky="w", pady=5)
        self.key_size_var = tk.StringVar(value="128")
        self.key_size_menu = ttk.Combobox(self.frame,
            textvariable=self.key_size_var, values=["128", "192",
            "256"], state="readonly", width=5)
        self.key_size_menu.grid(row=2, column=1,
            sticky="w", padx=10)

        ttk.Label(self.frame, text="Режим шифрування:",
            font=("Arial", 12, "bold")).grid(row=3, column=0,
            sticky="w", pady=5)
        self.mode_var = tk.StringVar(value="ECB")
        self.mode_menu = ttk.Combobox(self.frame,
            textvariable=self.mode_var, values=["ECB", "CBC", "CFB",
            "OFB"], state="readonly", width=5)
        self.mode_menu.grid(row=3, column=1,
            sticky="w", padx=10)

        button_width = 25
        self.encrypt_button = ttk.Button(self.frame, text="Зашифрувати",
            command=self.encrypt, style="TButton", width=button_width)
        self.encrypt_button.grid(row=4, column=0, pady=5,
            padx=5, sticky="w")

        self.decrypt_button = ttk.Button(self.frame, text="Розшифрувати",
            command=self.decrypt, style="TButton", width=button_width)
        self.decrypt_button.grid(row=4, column=1, pady=5,
            padx=5, sticky="w")

```

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				88
Змн.	Арк.	№ докум.	Підпис	Дата		

```

self.save_button = ttk.Button(self.frame, text="
Зберегти зашифроване", command=self.save_en-
crypt_text, style="TButton", width=button_width)
self.save_button.grid(row=5, column=0, pady=5,
padx=5, sticky="w")

self.save_decrypted_button = ttk.Button(self.frame,
text="Зберегти розшифроване", com-
mand=self.save_decrypted_text, style="TButton",
width=button_width)
self.save_decrypted_button.grid(row=5, column=1,
pady=5, padx=5, sticky="w")

self.clear_button = ttk.Button(self.frame, text="
Очистити", command=self.clear_fields, style="TBut-
ton", width=12)
self.clear_button.grid(row=3, column=1, padx=(0,
30), sticky="e")

ttk.Label(self.frame, text="Результат:",
font=("Arial", 12, "bold")).grid(row=7, column=0,
sticky="w", pady=5)
self.output_text_frame = ttk.Frame(self.frame)
self.output_text_frame.grid(row=8, column=0, col-
umnspan=2, pady=5, padx=5, sticky="w")

self.output_text_scroll = ttk.Scrollbar(self.out-
put_text_frame, orient="vertical")
self.output_text = tk.Text(self.output_text_frame,
height=8, width=60, bg="#7E8A97", fg="#F0ECE3",
insertbackground="black", font=("Arial", 10), yscroll-
command=self.output_text_scroll.set)
self.output_text_scroll.config(command=self.out-
put_text.yview)

self.output_text.pack(side="left", fill="both", ex-
pand=True)
self.output_text_scroll.pack(side="right", fill="y")

ttk.Label(self.frame, text="Шифрування файлів:",
font=("Arial", 12, "bold")).grid(row=9, column=0,
sticky="w", pady=10)

self.file_button_frame = ttk.Frame(self.frame)
self.file_button_frame.grid(row=10, column=0, col-
umnspan=2, pady=5, sticky="w")

self.file_encrypt_button = ttk.Button(self.file_but-
ton_frame, text=" Вибрати файл", command=self.se-
lect_file_encrypt, style="TButton", width=15)
self.file_encrypt_button.pack(side="left", padx=2)

self.file_decrypt_button = ttk.Button(self.file_but-
ton_frame, text=" Розшифрувати файл", com-
mand=self.select_file_decrypt, style="TButton",
width=20)
self.file_decrypt_button.pack(side="left", padx=2)

self.speed_button = ttk.Button(self.file_but-
ton_frame, text=" Побудова графіку", com-
mand=self.benchmark_tool.run_benchmark,
style="TButton", width=20)
self.speed_button.pack(side="left", padx=2)

```

```

self.image_window = None

def encrypt_by_mode(self, data, key):
    mode = self.mode_var.get()
    iv = get_random_bytes(16) if mode in ("CBC",
"CFB", "OFB") else None

    if mode == "ECB":
        encrypted = camellia_ecb_encrypt(data, key)
    elif mode == "CBC":
        encrypted = camellia_cbc_encrypt(data, key, iv)
    elif mode == "CFB":
        encrypted = camellia_cfb_encrypt(data, key, iv)
    elif mode == "OFB":
        encrypted = camellia_ofb_encrypt(data, key, iv)
    else:
        raise ValueError("Невідомий режим
шифрування")

    return encrypted, iv

def decrypt_by_mode_for_file(self, data: bytes, key:
bytes, filetype: str) -> bytes:
    mode = self.mode_var.get()

    if mode == "ECB":
        decrypted = camellia_ecb_decrypt(data, key) if
filetype == "text" else b"".join(
        camellia_decrypt_block(data[i:i+16], key) for i
in range(0, len(data), 16)
        )
    elif mode in ("CBC", "CFB", "OFB"):
        if len(data) < 16:
            raise ValueError("Дані занадто короткі для
вилучення IV.")
        iv, ciphertext = data[:16], data[16:]

        if mode == "CBC":
            if filetype == "text":
                decrypted = camellia_cbc_decrypt(cipher-
text, key, iv)
            else:
                decrypted = camellia_cbc_decrypt_raw(ci-
phertext, key, iv)
        elif mode == "CFB":
            decrypted = camellia_cfb_decrypt(ciphertext,
key, iv)
        elif mode == "OFB":
            decrypted = camellia_ofb_decrypt(ciphertext,
key, iv)
        else:
            raise ValueError(f"Невідомий режим
шифрування: {mode}")

    return decrypted

@staticmethod
def raw_cbc_decrypt(ciphertext: bytes, key: bytes, iv:
bytes) -> bytes:
    decrypted = b"
    prev = iv
    for i in range(0, len(ciphertext), 16):
        block = ciphertext[i:i+16]

```

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				89
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        decrypted_block = camellia_decrypt_block(block, key)
        decrypted += bytes(a ^ b for a, b in zip(decrypted_block, prev))
        prev = block
        return decrypted

def paste_from_clipboard(self):
    try:
        clipboard_text = self.root.clipboard_get()
        self.input_text.delete("1.0", tk.END)
        self.input_text.insert(tk.END, clipboard_text)
    except tk.TclError:
        messagebox.showwarning("Увага", "У буфері обміну немає тексту.")

def looks_like_text(self, text: str) -> bool:
    sample = text.strip()[:100]
    words = re.findall(r"[A-Za-zA-Яа-яІіЇїЄє]{3,}", sample)
    has_capital = any(c.isupper() for c in sample)
    has_space = ' ' in sample
    has_punctuation = any(p in sample for p in ".-,:;!?"")
    return len(words) >= 2 and has_capital and has_space and has_punctuation

def safe_decode_text(self, decrypted: bytes) -> str | None:
    try:
        text = decrypted.decode("utf-8")
        if self.looks_like_text(text):
            return text
    except UnicodeDecodeError:
        pass

    for i in range(1, min(128, len(decrypted))):
        try:
            text = decrypted[i:].decode("utf-8")
            if self.looks_like_text(text):
                return text
        except UnicodeDecodeError:
            continue
    return None

def generate_key(self):
    try:
        key_size = int(self.key_size_var.get())
        if key_size not in (128, 192, 256):
            raise ValueError("Неправильна довжина ключа")
        return os.urandom(key_size // 8)
    except Exception:
        messagebox.showerror("Помилка", "Неправильна довжина ключа! Виберіть 128, 192 або 256.")
        return None

def encrypt(self):
    key = self.generate_key()
    if key is None:
        return

```

```

        data = self.input_text.get("1.0", tk.END).strip().encode("utf-8")
        mode = self.mode_var.get()

        if mode not in ("ECB", "CBC", "CFB", "OFB"):
            messagebox.showerror("Помилка", "Будь ласка, оберіть режим шифрування.")
            return

    try:
        encrypted, iv = self.encrypt_by_mode(data, key)
    except Exception as e:
        messagebox.showerror("Помилка", f"Помилка шифрування: {str(e)}")
        return

    full_encrypted = encrypted
    if iv:
        full_encrypted = iv + encrypted

    output = f"Ключ: {binascii.hexlify(key).decode()} \n"
    if iv:
        output += f"IV: {binascii.hexlify(iv).decode()} \n"
    output += f"Зашифрований текст: {binascii.hexlify(full_encrypted).decode()}"

    self.output_text.delete("1.0", tk.END)
    self.output_text.insert(tk.END, output)

def decrypt(self):
    try:
        input_data = self.input_text.get("1.0", tk.END).strip()
        lines = [line.strip() for line in input_data.split("\n") if line.strip()]
        if len(lines) < 2:
            messagebox.showerror("Помилка", "Некоректний формат даних!")
            return

        key_line = next((line for line in lines if line.startswith("Ключ:")), None)
        data_line = next((line for line in lines if line.startswith("Зашифрований текст:")), None)

        if not key_line or not data_line:
            messagebox.showerror("Помилка", "Не вдалося знайти ключ або зашифрований текст.")
            return

        key = binascii.unhexlify(key_line.split(":")[1].strip())
        encrypted_data = binascii.unhexlify(data_line.split(":")[1].strip())

        if len(encrypted_data) < 16:
            messagebox.showerror("Помилка", "Зашифровані дані занадто короткі (менше 16 байт)")
            return

```

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				90
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        decrypted = self.decrypt_by_mode(encrypted_data, key)

        if self.mode_var.get() in ("ECB", "CBC") and self.looks_like_text(decrypted.decode('utf-8', errors='ignore')):
            try:
                decrypted = unpad_data(decrypted)
            except Exception:
                pass

            text = self.safe_decode_text(decrypted)
            if text is None:
                self.output_text.delete("1.0", tk.END)
                self.output_text.insert(tk.END, "Розшифровані дані — не текст. Збережіть їх у файл.")
                return

            self.output_text.delete("1.0", tk.END)
            self.output_text.insert(tk.END, f"Розшифрований текст:\n{text}")

        except Exception as e:
            messagebox.showerror("Помилка", f"Не вдалося розшифрувати: {str(e)}")

    def encrypt_file(self, filepath, key):
        ext = os.path.splitext(filepath)[-1].lower()

        try:
            if ext in (".txt", ".docx", ".pdf"):
                with open(filepath, "rb") as file:
                    data = file.read()
            else:
                messagebox.showerror("Помилка", f"Формат файлу {ext} не підтримується!")
                return None, None
        except Exception as e:
            messagebox.showerror("Помилка читання", f"Не вдалося прочитати файл: {str(e)}")
            return None, None

        try:
            encrypted, iv = self.encrypt_by_mode(data, key)
        except Exception as e:
            messagebox.showerror("Помилка шифрування", f"Не вдалося зашифрувати файл: {str(e)}")
            return None, None

        encrypted_filepath = filepath + ".enc"
        try:
            with open(encrypted_filepath, "wb") as file:
                file.write(iv + encrypted if iv else encrypted)
        except Exception as e:
            messagebox.showerror("Помилка", f"Не вдалося зберегти зашифрований файл: {str(e)}")
            return None, None

        return encrypted_filepath, key

    def decrypt_file(self, filepath, key=None):
        if not key:

```

```

            key_hex = simpledialog.askstring("Введіть ключ", "Введіть шістнадцятковий ключ:", initialvalue=self.root.clipboard_get())
            if not key_hex:
                messagebox.showwarning("Скасовано", "Розшифрування скасовано.")
                return
            try:
                key = binascii.unhexlify(key_hex)
            except binascii.Error:
                messagebox.showerror("Помилка", "Неправильний формат ключа!")
                return

            try:
                with open(filepath, "rb") as file:
                    encrypted_data = file.read()
            except Exception as e:
                messagebox.showerror("Помилка", f"Не вдалося прочитати файл: {str(e)}")
                return

            try:
                decrypted = self.decrypt_by_mode_for_file(encrypted_data, key, "binary")
                if decrypted is None:
                    messagebox.showerror("Помилка", "Не вдалося розшифрувати файл: недійсні або пошкоджені дані")
                    return
            except Exception as e:
                messagebox.showerror("Помилка дешифрування", f"Не вдалося розшифрувати: {str(e)}")
                return

            save_path = filedialog.asksaveasfilename(
                title="Зберегти розшифрований файл як",
                defaulttextextension=".txt",
                filetypes=[
                    ("PDF", "*.pdf"),
                    ("Word документ", "*.docx"),
                    ("Зображення JPEG", "*.jpg"),
                    ("Зображення PNG", "*.png"),
                    ("Аудіо MP3", "*.mp3"),
                    ("Аудіо WAV", "*.wav"),
                    ("Текстовий файл", "*.txt"),
                    ("Усі файли", "*.*")
                ]
            )
            if not save_path:
                return

            try:
                ext = os.path.splitext(save_path)[-1].lower()
                if ext in (".txt", ".docx"):
                    text = decrypted.decode("utf-8", errors="replace")
                    if ext == ".txt":
                        with open(save_path, "w", encoding="utf-8") as file:
                            file.write(text)
                    else:

```

```

doc = Document()
for line in text.split("\n"):
    doc.add_paragraph(line)
doc.save(save_path)
else:
    with open(save_path, "wb") as file:
        file.write(decrypted)
    messagebox.showinfo("Успіх", f"Файл успішно
розшифровано: {save_path}")
except Exception as e:
    messagebox.showerror("Помилка", f"Не
вдалося зберегти файл: {str(e)}")

def encrypt_audio(self, filepath, key):
    try:
        with open(filepath, "rb") as file:
            audio_data = file.read()
        except Exception as e:
            messagebox.showerror("Помилка", f"Не
вдалося прочитати аудіофайл: {str(e)}")
            return None, None

    try:
        encrypted, iv = self.encrypt_by_mode(audio_data, key)
    except Exception as e:
        messagebox.showerror("Помилка
шифрування", f"Не вдалося зашифрувати аудіо:
{str(e)}")
        return None, None

    encrypted_filepath = filepath + ".enc"
    try:
        with open(encrypted_filepath, "wb") as file:
            file.write(iv + encrypted if iv else encrypted)
        except Exception as e:
            messagebox.showerror("Помилка збереження",
f"Не вдалося зберегти зашифроване аудіо: {str(e)}")
            return None, None

    return encrypted_filepath, key

def decrypt_audio(self, filepath, key):
    try:
        with open(filepath, "rb") as file:
            encrypted_data = file.read()
        except Exception as e:
            messagebox.showerror("Помилка", f"Не
вдалося прочитати файл: {str(e)}")
            return

    try:
        decrypted = self.decrypt_by_mode_for_file(encrypted_data, key, "binary")
    except Exception as e:
        messagebox.showerror("Помилка", f"Не
вдалося розшифрувати аудіо: {str(e)}")
        return

    save_path = filedialog.asksaveasfilename(defaultextension=".wav",
filetypes=[("Аудіофайли",
"*.*mp3;*.wav")])

```

```

if save_path:
    with open(save_path, "wb") as file:
        file.write(decrypted)
    messagebox.showinfo("Успіх", f"Аудіо
збережено: {save_path}")

def decrypt_image(self, filepath, key):
    try:
        image = Image.open(filepath).convert("RGB")
        width, height = image.size
        img_bytes = image.tobytes()
        size_header = struct.pack(">II", width,
height)
        full_data = size_header + img_bytes
        except Exception as e:
            messagebox.showerror("Помилка", f"Не
вдалося прочитати зображення: {str(e)}")
            return None, None

    try:
        encrypted, iv = self.encrypt_by_mode(full_data,
key)
        except Exception as e:
            messagebox.showerror("Помилка
шифрування", f"Не вдалося зашифрувати
зображення: {str(e)}")
            return None, None

    encrypted_filepath = filepath + ".enc"
    try:
        with open(encrypted_filepath, "wb") as file:
            file.write(iv + encrypted if iv else encrypted)
        except Exception as e:
            messagebox.showerror("Помилка збереження",
f"Не вдалося зберегти зашифроване зображення:
{str(e)}")
            return None, None

    return encrypted_filepath, key

def decrypt_image(self, filepath, key):
    try:
        with open(filepath, "rb") as file:
            encrypted_data = file.read()
        except Exception as e:
            messagebox.showerror("Помилка", f"Не
вдалося прочитати файл: {str(e)}")
            return

    try:
        decrypted = self.decrypt_by_mode_for_file(encrypted_data, key, "binary")
        width, height = struct.unpack(">II", decrypted[:8])
        img_data = decrypted[8:]

        expected_size = width * height * 3
        if len(img_data) != expected_size:
            raise ValueError(f"Некоректні дані
зображення: очікується {expected_size} байтів,
отримано {len(img_data)}")

```

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				92
Змн.	Арк.	№ докум.	Підпис	Дата		

```

img = Image.frombytes("RGB", (width, height),
img_data)
except Exception as e:
    messagebox.showerror("Помилка", f"Не
вдалося зібрати зображення: {str(e)}")
    return

save_path = filedialog.asksaveasfilename(
    defaultextension=".png",
    filetypes=[
        ("PNG зображення", "*.png"),
        ("JPEG зображення", "*.jpg;*.jpeg")
    ],
    title="Зберегти зображення"
)

if save_path:
    try:
        ext = os.path.splitext(save_path)[-1].lower()
        img_format = "PNG" if ext == ".png" else
"JPEG"
        img.save(save_path, format=img_format)
        messagebox.showinfo("Успіх", f"Зображення
збережено: {save_path}")
    except Exception as e:
        messagebox.showerror("Помилка", f"Не
вдалося зберегти зображення: {str(e)}")

def save_encrypted_text(self):
    content = self.output_text.get("1.0", tk.END).strip()
    if not content:
        messagebox.showwarning("Попередження",
"Немає даних для збереження!")
        return

    file_path = filedialog.asksaveasfilename(defaulttex-
tension=".txt", filetypes=[("Текстові файли", "*.txt")])
    if not file_path:
        return

    content_lines = content.split("\n")
    if len(content_lines) < 2:
        messagebox.showerror("Помилка", "Дані для
збереження некоректні!")
        return

    key_line = next((line for line in content_lines if
line.startswith("Ключ:")), None)
    iv_line = next((line for line in content_lines if
line.startswith("IV:")), None)
    encrypted_text_line = next((line for line in con-
tent_lines if line.startswith("Зашифрований текст:")),
None)

    if not key_line or not encrypted_text_line:
        messagebox.showerror("Помилка",
"Неможливо знайти ключ або зашифрований текст!")
        return

    key = key_line.split(":")[1].strip()
    encrypted_text = en-
crypted_text_line.split(":")[1].strip()

```

```

key_file = file_path.replace(".txt", "_key.txt")
encrypted_file = file_path.replace(".txt", "_en-
crypted.txt")

with open(key_file, "w", encoding="utf-8") as file:
    file.write(key)

with open(encrypted_file, "w", encoding="utf-8") as
file:
    file.write(encrypted_text)

if iv_line:
    iv = iv_line.split(":")[1].strip()
    iv_file = file_path.replace(".txt", "_iv.txt")
    with open(iv_file, "w", encoding="utf-8") as file:
        file.write(iv)
    messagebox.showinfo("Успіх", f"Ключ
збережено у: {key_file}\nIV збережено у:
{iv_file}\nЗашифрований текст у: {encrypted_file}")
    else:
        messagebox.showinfo("Успіх", f"Ключ
збережено у: {key_file}\nЗашифрований текст у: {en-
crypted_file}")

def save_decrypted_text(self):
    content = self.output_text.get("1.0", tk.END).strip()
    if not content:
        messagebox.showwarning("Попередження",
"Немає розшифрованого тексту для збереження!")
        return
    file_path = filedialog.asksaveasfilename(defaulttex-
tension=".txt", filetypes=[("Текстові файли", "*.txt")])
    if file_path:
        with open(file_path, "w", encoding="utf-8") as
file:
            file.write(content)
        messagebox.showinfo("Успіх",
"Розшифрований текст збережено успішно!")

def clear_fields(self):
    self.input_text.delete("1.0", tk.END)
    self.output_text.delete("1.0", tk.END)

def select_file_encrypt(self):
    filetypes = [("Всі файли", "*.*"),
        ("Текстові файли", "*.txt"),
        ("Word файли", "*.docx"),
        ("PDF файли", "*.pdf"),
        ("Зображення", "*.png;*.jpg;*.jpeg"),
        ("Аудіофайли", "*.mp3;*.wav")]

    filepath = filedialog.askopenfilename(filetypes=file-
types)

    if filepath:
        key = self.generate_key()
        if key is None:
            return

        ext = os.path.splitext(filepath)[-1].lower()
        if ext in [".png", ".jpg", ".jpeg"]:
            encrypted_path, key = self.encrypt_im-
age(filepath, key)

```

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		93

```

elif ext in [".mp3", ".wav"]:
    encrypted_path, key = self.encrypt_audio(filepath, key)
else:
    encrypted_path, key = self.encrypt_file(filepath, key)

    if encrypted_path:
        def copy_key_and_close():
            self.root.clipboard_clear()
            self.root.clipboard_append(binascii.hexlify(key).decode())
            self.root.update()
            messagebox.showinfo("Копіювання", "Ключ скопійовано в буфер обміну!")
            key_window.destroy()

        key_window = tk.Toplevel(self.root)
        key_window.title("Ключ шифрування")
        ttk.Label(key_window, text=f"Ключ: {binascii.hexlify(key).decode()}").pack(pady=10)
        ttk.Button(key_window, text="Копіювати та закрити", command=copy_key_and_close).pack(pady=5)

        messagebox.showinfo("Успіх", f"Файл зашифровано у: {encrypted_path}")

    def select_file_decrypt(self):
        filetypes = [("Зашифровані файли", "*.enc")]
        filepath = filedialog.askopenfilename(filetypes=filetypes)

        if not filepath:
            return

        key_hex = simpledialog.askstring("Введіть ключ", "Введіть шістнадцятковий ключ (32/48/64 символи):")
        if not key_hex:
            return

        try:
            key = binascii.unhexlify(key_hex)
        except binascii.Error:
            messagebox.showerror("Помилка", "Неправильний формат ключа!")
            return

        if len(key) not in (16, 24, 32):
            messagebox.showerror("Помилка", "Ключ повинен бути довжиною 128, 192 або 256 біт (16, 24, 32 байти)!")
            return

        try:
            with open(filepath, "rb") as file:
                encrypted_data = file.read()
        except Exception as e:
            messagebox.showerror("Помилка", f"Не вдалося прочитати файл: {str(e)}")
            return

```

```

try:
    decrypted = self.decrypt_by_mode_for_file(encrypted_data, key, "binary")
except Exception as e:
    messagebox.showerror("Помилка дешифрування", f"Не вдалося розшифрувати: {str(e)}")
    return

with open("debug_output.bin", "wb") as dbg:
    dbg.write(decrypted[:64])

ext = None

if decrypted[:4] == b"%PDF":
    ext = ".pdf"
elif decrypted[:2] == b"PK" and b"word/" in decrypted:
    ext = ".docx"
elif decrypted[0:4] == b"RIFF" and b"WAVE" in decrypted[8:16]:
    ext = ".wav"
elif decrypted.startswith(b"ID3") or decrypted[0:2] == b"\xFF\xFB":
    ext = ".mp3"
elif len(decrypted) >= 8:
    try:
        width, height = struct.unpack(">II", decrypted[:8])
        expected_size = width * height * 3
        img_data = decrypted[8:]
        if len(img_data) == expected_size:
            ext = ".jpg"
            if self.mode_var.get() == "ECB":
                messagebox.showwarning("Увага", "Шифрування зображень у режимі ECB не рекомендоване. Використовуйте CBC/CFB/OFB.")
            except (struct.error, ValueError):
                pass

        if ext is None:
            messagebox.showwarning("Увага", "Тип файлу не розпізнано. Збереження скасовано.")
            return

    save_path = filedialog.asksaveasfilename(
        title="Зберегти розшифрований файл як",
        defaultextension=ext,
        filetypes=[
            ("Усі файли", "*.*"),
            ("Текстовий файл", "*.txt"),
            ("Word документ", "*.docx"),
            ("PDF", "*.pdf"),
            ("Зображення JPEG", "*.jpg"),
            ("Зображення PNG", "*.png"),
            ("Аудіо MP3", "*.mp3"),
            ("Аудіо WAV", "*.wav")
        ]
    )

    if not save_path:
        return

```

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		94

```

try:
    if ext == ".txt":
        text = decrypted.decode("utf-8", errors="re-
place")
        with open(save_path, "w", encoding="utf-8")
as file:
            file.write(text)

    elif ext in (".jpg", ".png"):
        width, height = struct.unpack(">II", de-
crypted[:8])
        img_data = decrypted[8:]
        expected_size = width * height * 3

        if len(img_data) != expected_size:
            raise ValueError(f"Некоректні дані
зображення: очікується {expected_size} байтів,
отримано {len(img_data)}")

        img = Image.frombytes("RGB", (width,
height), img_data)

```

```

        img_format = "PNG" if ext == ".png" else
"JPEG"
        img.save(save_path, format=img_format)

    else:
        with open(save_path, "wb") as file:
            file.write(decrypted)

        messagebox.showinfo("Успіх", f"Файл успішно
розшифровано: {save_path}")

    except Exception as e:
        messagebox.showerror("Помилка", f"Не
вдалося зберегти файл: {str(e)}")

def main():
    root = tk.Tk()
    app = CamelliaCipherApp(root)
    root.mainloop()

if __name__ == "__main__":
    main()

```

		Поліщук К.Р.			ККІКБ.КР.Б – 125 – 16 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		95

СХЕМА ПРОЦЕДУРИ ЗАШИФРУВАННЯ

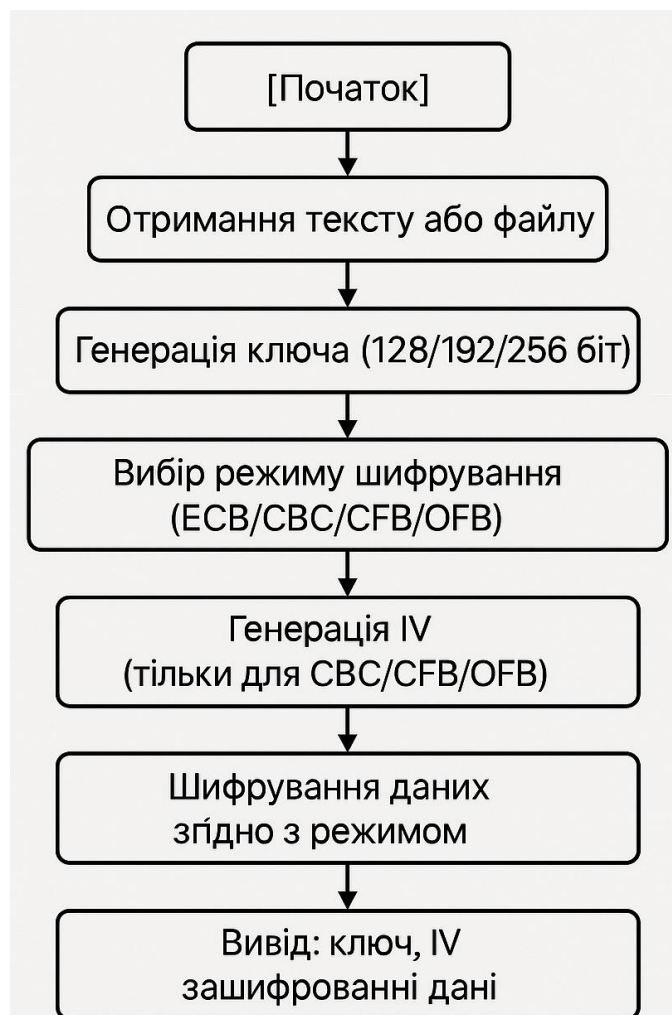


СХЕМА ПРОЦЕДУРИ РОЗШИФРУВАННЯ



СХЕМА РОБОТИ ПРОГРАМИ (ЗАГАЛЬНА ЛОГІКА GUI)

