МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО – НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

перший (бакалаврський)
(рівень вищої освіти)

на тему **Розробка Веб-застосунку для навчання криптографічним алгоритмам засобами платформи Java**

Виконав: студент 4 курсу, групи 6.1211-пзс
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Програмне забезпечення систем
(код і назва освітньої програми)
В.К. Свєрлік
(ініціали та прізвище)
Керівник к.ф.-м.н., доцент Г.П. Коломоєць
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент Директор ТОВ «Дискус»
Р.О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра електроніки, інформаційних систем та програмного забезпечення
Рівень вищої освіти перший (бакалавський)
Спеціальність 121 Інженерія програмного забезпечення
(код та назва)
Освітня програма Програмне забезпечення систем
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри Тетяна КРИТСЬКА
“ 01 ” березня 2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Свєрліку Віктору Костянтиновичу
(прізвище, ім’я, по батькові)

1. Тема роботи Розробка Веб-застосунку для навчання криптографічним алгоритмам засобами платформи Java
керівник роботи Коломоєць Геннадій Павлович, к.ф.-м.н., доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від 27.12.2024 № 2029-с

2. Строк подання студентом кваліфікаційної роботи 02.06.2025
3. Вихідні дані кваліфікаційної роботи бакалавра
   - комплект нормативних документів ;
   - технічне завдання до роботи.
4. Зміст розрахунково – пояснювальної записки (перелік питань, які потрібно розробити)
   - огляд та збір літератури стосовно теми кваліфікаційної роботи;
   - огляд та аналіз існуючих рішень та аналогів;
   - аналіз сучасних технологій для створення клієнтської частини Веб застосунків;
   - проєктування Веб-застосунку;
   - програмна реалізація застосунку.
5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
   слайдів презентації

6. Консультанти розділів бакалаврської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата Завдання прийняв

7. Дата видачі завдання 01.03.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів бакалаврської роботи	Примітка
1	Формулювання основної задачі дипломної роботи, узгодження з науковим керівником	16.12.2024	виконано
2	Вивчення класичних шифрів (Віженера, Гронсфельда) та сучасних криптоалгоритмів (DES, AES, RSA, MD5, SHA)	До 16.02.2025	виконано
3	Ознайомлення з екосистемою Spring, Spring Boot, контекстами та біновою структурою	До 30.03.2025	виконано
4	Участь у конференції «Молода наука - 2025»	14–15.04.2025	виконано
5	Проєктування архітектури Веб-застосунку: модулі, інтерфейси, зв'язки	До 10.05.2025	виконано
6	Реалізація функціоналу застосунку (інтерфейс користувача, алгоритми шифрування/дешифрування)	До 26.05.2025	виконано
7	Тестування та налагодження системи	До 29.05.2025	виконано
8	Розробка рекомендацій щодо використання ПЗ у навчальному процесі	До 31.05.2025	виконано
9	Оформлення пояснювальної записки	До 02.06.2025	виконано
10	Попередній захист	03.06.2025	виконано
11	Коригування зауважень після попереднього захисту	До 13.06.2025	виконано

Студент _____ В.К. Свєрлік
(підпис) (прізвище та ініціали)

Керівник роботи _____ Г.П. Коломоєць
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ І.А. Скрипник
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок – 83

Рисунків – 29

Таблиць – 3

Лістинги – 12

Джерел – 24

Свєрлік В.К. Розробка Веб-застосунку для навчання криптографічним алгоритмам засобами платформи Java: кваліфікаційна робота бакалавра за спеціальністю 121 “Інженерія програмного забезпечення” / науковий керівник Г. П. Коломоєць. Запоріжжя: ЗНУ, 2025. 83 с.

Метою роботи є створення інтерактивного веб-застосунку, який допоможе користувачам вивчати криптографічні алгоритми на практичному досвіді.

У роботі розглядається класифікація криптографічних алгоритмів (класичні, симетричні, асиметричні, хеш-функції), аналізуються системи навчання CrypTool та обґрунтовується архітектура веб-застосунку. Реалізація виконується на платформі Java з використанням Spring Boot для побудови REST API, а також бібліотек JCA та Bouncy Castle для криптографічної логіки.

Веб-інтерфейс створено за допомогою HTML, CSS та JavaScript. Реалізовано підтримку декількох тем оформлення, збереження історії операцій, тестування та пояснень. Було проведено тестування функціональності та оцінку покриття коду за допомогою інструментів JaCoCo.

Ключові слова: *веб-застосунок, Spring Boot, криптографія, шифрування, AES, RSA, цифровий підпис, візуалізація, навчання.*

ABSTRACT

Pages – 83

Drawings – 29

Tables – 3

Listings – 12

Sources – 24

Sverlik V.K. Development of a Web Application for Teaching of Cryptographic Algorithms Using the Java Platform: Bachelor's Qualification Thesis in specialty 121 “Software Engineering” / Scientific supervisor H. P. Kolomoiets. Zaporizhzhia: ZNU, 2025. 83 p.

The aim of this work is to create an interactive web application that helps users learn cryptographic algorithms through hands-on experience.

The thesis examines the classification of cryptographic algorithms (classical, symmetric, asymmetric, hash functions), analyzes CrypTool learning systems, and substantiates the architecture of the web application. The implementation is carried out on the Java platform using Spring Boot to build a REST API and the JCA and Bouncy Castle libraries for cryptographic logic.

The web interface is developed using HTML, CSS, and JavaScript. The system supports multiple themes, operation history tracking, interactive testing, and detailed explanations. Testing of the software and code coverage analysis were performed using the JaCoCo toolkit.

Keywords: *web application, Spring Boot, cryptography, encryption, AES, RSA, digital signature, visualization, education.*

ЗМІСТ

ВСТУП 8
1 ЗАСОБИ JAVA ДЛЯ ОРГАНІЗАЦІЇ БЕЗПЕКИ ЗАСТОСУНКІВ 12
1.1 Визначення предметної області криптографії 12
1.2 Класифікація криптографічних алгоритмів 12
1.3 Частотний аналіз у криптоаналізі 16
1.4 Класичні шифри 17
1.4.1 Шифр Цезаря 17
1.4.2 Шифр Віженера 18
1.4.3 Шифр колонної перестановки 19
1.4.4 Гамування з ПВП (псевдовипадковою послідовністю) 19
1.5 Симетричні шифри 20
1.5.1 DES (Data Encryption Standard) 20
1.5.2 DES (Triple DES – TDES) 22
1.5.3 AES (Advanced Encryption Standard) 23
1.6 Асиметричні шифри 25
1.6.1 RSA (Rivest–Shamir–Adleman) 25
1.6.2 ECDSA (Elliptic Curve Digital Signature Algorithm) 26
1. Генерація пари ключів: 26
2. Підпис повідомлення: 26
3. Перевірка підпису: 26
1.7 Алгоритми отримання хеш-сум 27
1.7.1 MD5 (Message Digest Algorithm 5) 27
1.7.2 SHA-256 (Secure Hash Algorithm 256) 29
1.8 Криптографічні засоби платформи Java 30
1.9 Аналіз аналогічних програмних продуктів 31
1.9.1 CrypTool 31
1.9.2 CrypTool-Online 34
1.9.3 CrypTool 2 (CT2) 36

1.10 Постановка завдання 40
2 ВИВЧЕННЯ ЗАСОБІВ SPRING FRAMEWORK ДЛЯ РОЗРОБКИ ВЕБ-ЗАСТОСУНКІВ 41
2.1 Екосистема Spring та її основні проєкти 41
2.2 Концепції Bean-компонентів. Контекст застосунку та способи додавання Bean-компонентів 42
2.3 Інструменти Spring Framework для створення веб-застосунків 42
2.4 Spring Boot та його можливості 45
2.5 Визначення функціональності застосунку, що реалізується 46
3 ПРОЕКТУВАННЯ ТА ПОБУДОВА ВЕБ-ЗАСТОСУНКУ ДЛЯ НАВЧАННЯ КРИПТОГРАФІЧНИМ АЛГОРИТМАМ 47
3.1 Загальні принципи проектування 47
3.2 Архітектура застосунку 47
3.2.1 Рівень представлення (Presentation Layer) 48
3.2.2 Рівень контролерів (Controller Layer) 48
3.2.3 Рівень бізнес-логіки (Service Layer) 48
3.2.4 Модель (Model / DTO) 49
3.2.5 Взаємодія компонентів 49
3.2.6 Структура проєкту 50
3.3 Проєктування інтерфейсу та функціональних можливостей застосунку 54
3.4 Реалізація обробки криптографічних алгоритмів на прикладі Spring Boot 59
3.5 Організація збереження історії операцій у браузері 63
3.6 Тестування функціональності застосунку 66
3.7 Розробка рекомендацій щодо використання ПЗ у навчальному процесі 77
ВИСНОВКИ 80
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 81

ВСТУП

Актуальність теми

Криптографія вже давно є наріжним каменем інформаційної безпеки, але з кожним роком її роль стає все більш важливою. Ми щодня стикаємося з криптографічними алгоритмами, навіть не помічаючи цього — від банківських операцій до захисту повідомлень у месенджерах. Однак для багатьох людей ця область залишається складною і незрозумілою через велику кількість математичних аспектів і абстрактних понять.

На жаль, більшість навчальних матеріалів з криптографії або занадто теоретичні, або містять багато складних технічних деталей без практичного застосування. Це ускладнює роботу учнів, які тільки починають знайомство з цією темою. Тому потрібні сучасні інтерактивні освітні засоби, які дозволяють не лише читати теорію, а й експериментувати з алгоритмами в реальному часі.

Однак через складність математичних основ криптографію часто важко вивчити, особливо для початківців. Традиційні підходи до навчання здебільшого зосереджені на теоретичних аспектах і обмежують можливість експериментального вивчення алгоритмів у реальному часі. Це створює потребу в розробці інтерактивного веб-застосунку, який дозволить користувачам не тільки читати про криптографічні методи, але й перевіряти їх алгоритми на практиці.

Веб-застосунок, який поєднує візуалізацію процесів шифрування та хешування, може значно полегшити розуміння криптографічних методів. Завдяки інтерактивному підходу користувачі зможуть не тільки побачити, як змінюються дані під час шифрування, але й перевірити самі алгоритми, ввівши власні значення.

Мета дослідження

Метою роботи є створення інтерактивного веб-застосунку, який допоможе користувачам вивчати криптографічні алгоритми на практичному досвіді.

Завдання дослідження

Для досягнення поставленої мети необхідно виконати такі завдання:

- дослідити основні криптографічні алгоритми, що застосовуються для шифрування, дешифрування та хешування даних;
- проаналізувати існуючі навчальні застосунки та визначити їхні переваги й недоліки у навчанні криптографії;
- розробити архітектуру веб-застосунку, що дозволятиме користувачам взаємодіяти з криптографічними алгоритмами в режимі реального часу;
- реалізувати модулі, які пояснюватимуть роботу криптографічних алгоритмів за допомогою інтерактивних прикладів;
- впровадити систему тестування та самоперевірки для оцінки рівня знань користувачів у сфері криптографії.

Об'єкт дослідження

Об’єктом дослідження є процес навчання криптографічним алгоритмам за допомогою інтерактивних технологій.

Предмет дослідження

Предметом дослідження є методи розробки веб-застосунку для навчання криптографії на платформі Java.

Методи дослідження

Для вирішення поставлених завдань були використані такі методи:

- аналіз наукових джерел з криптографії та освітніх технологій;
- дослідження підходів до інтерактивного навчання та вибір оптимального методу впровадження навчального застосунку;
- розробка веб-застосунку на основі Spring Boot ;
- тестування програми на різних типах даних для оцінки її ефективності в процесі навчання.

Практичне значення роботи

Розроблений веб-застосунок можна використовувати як навчальний інструмент у вищих навчальних закладах або як окремий ресурс для тих, хто хоче краще зрозуміти використані у роботі криптографічні алгоритми. Завдяки інтерактивному підходу розроблена програма дозволить користувачам не тільки отримати знання, а й застосувати їх на практиці, що значно підвищить ефективність процесу навчання.

Апробація одержаних результатів

Результати дослідження були представлені на XVIII науково-практичній конференції студентів, аспірантів, докторантів і молодих вчених Запорізького національного університету «Молода наука–2025» [1].

Глосарій

Криптографія — наука про методи захисту інформації шляхом переведення її у форму, недоступну для несанкціонованого доступу (наприклад, за допомогою шифрування).

Веб-застосунок – це програмний продукт, який працює через веб-браузер і виконує інтерактивні функції для користувача в інтернеті.
інтерактивність — властивість програмного засобу дозволяти користувачам взаємодіяти з інтерфейсом у реальному часі (наприклад, вводити текст для шифрування).

Візуалізація алгоритму — спосіб відображення внутрішніх етапів алгоритму за допомогою графіків або інших графічних засобів для кращого розуміння користувачем.

Симетричне шифрування – це тип шифрування, де для шифрування та дешифрування використовується один і той же ключ (наприклад, алгоритм AES).

Асиметричне шифрування — метод шифрування, який використовує

пару ключів: відкритий ключ для шифрування та закритий ключ для дешифрування (наприклад, алгоритм RSA).

Хешування — це процес перетворення довільного набору даних у хеш-суму фіксованої довжини за допомогою спеціальних алгоритмів (наприклад, MD5 або SHA-256).

Цифровий підпис — це криптографічний механізм, що забезпечує достовірність і цілісність електронних документів (наприклад, ECDSA).

Блоковий шифр — криптографічний алгоритм, який шифрує дані блоками фіксованого розміру (наприклад, AES працює з 128-бітними блоками).

Поточний шифр — алгоритм, який шифрує дані по одному біту або байту, створюючи псевдовипадковий потік (наприклад, шумоподавлення RC4 або PVP).

1 ЗАСОБИ JAVA ДЛЯ ОРГАНІЗАЦІЇ БЕЗПЕКИ ЗАСТОСУНКІВ

1.1 Визначення предметної області криптографії

Криптографія є ключовим компонентом інформаційної безпеки, оскільки забезпечує конфіденційність та цілісність інформації за допомогою технологій шифрування даних та аутентифікації користувачів та пристроїв. Вона використовується в багатьох сферах, включаючи захист персональних даних, безпечні фінансові транзакції та комунікації, цифрові підписи та технології блокчейну [3]. Через збільшення кількості кіберзагроз важливість криптографії як науки продовжує зростати (див. рис. 1-2).

Incidents by Severity Level	H2'2023	H1'2024	Change
Critical	31	3	-90%
High	156	45	-71%
Medium	1264	1670	32%
Low	12	21	75%
Total	1463	1739	19%

Рисунок *1 — Статистика інцидентів за рівнем серйозності. (24.03.2025)* [2]

Однак важливо зазначити, що ця статистика відносна і залежить від різних факторів. Окрім змін у тенденціях атак у кіберпросторі, на ці цифри також впливає підвищення можливості виявлення загроз, зростання обізнаності (більше організацій, які шукають консультацію/допомогу) та інші фактори [2].

1.2 Класифікація криптографічних алгоритмів

Криптографічні алгоритми класифікуються за принципом їхньої роботи та призначенням. На основі цього їх можна поділити на такі основні категорії (див. рис. 3) [3].

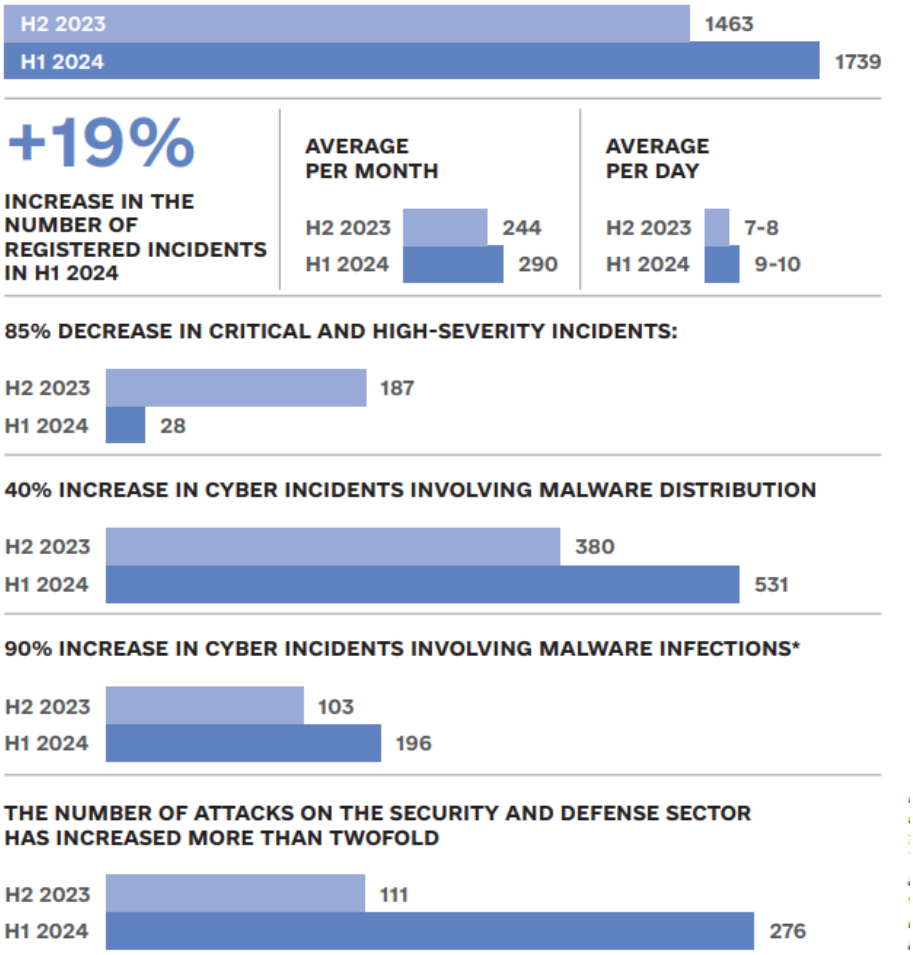


Рисунок *2 — Кількісні показники для оброблених інцидентів. (24.03.2025)* [2]

Криптографічні алгоритми – це математичні методи та процедури, що використовуються для забезпечення конфіденційності, цілісності та автентифікації даних [3]. Вони застосовуються для захисту інформації від несанкціонованого доступу та модифікації.

Безключові алгоритми не використовують секретного ключа. Вони застосовуються здебільшого для перевірки цілісності даних, а не для їхнього шифрування.

Хешування – це процес перетворення вхідних даних будь-якої довжини у фіксоване значення (хеш). Його основне призначення – перевірка цілісності повідомлень і безпечне зберігання паролів. Приклади хеш-алгоритмів:

- MD5 (Message Digest 5).
- SHA-1 (Secure Hash Algorithm 1).

- SHA-224 (Secure Hash Algorithm 224).
- SHA-256 (Secure Hash Algorithm 256).
- SHA-384 (Secure Hash Algorithm 384).
- SHA-512 (Secure Hash Algorithm 512).
- SHA-3 (Keccak Algorithm).
- RIPEMD (RACE Integrity Primitives Evaluation Message Digest).

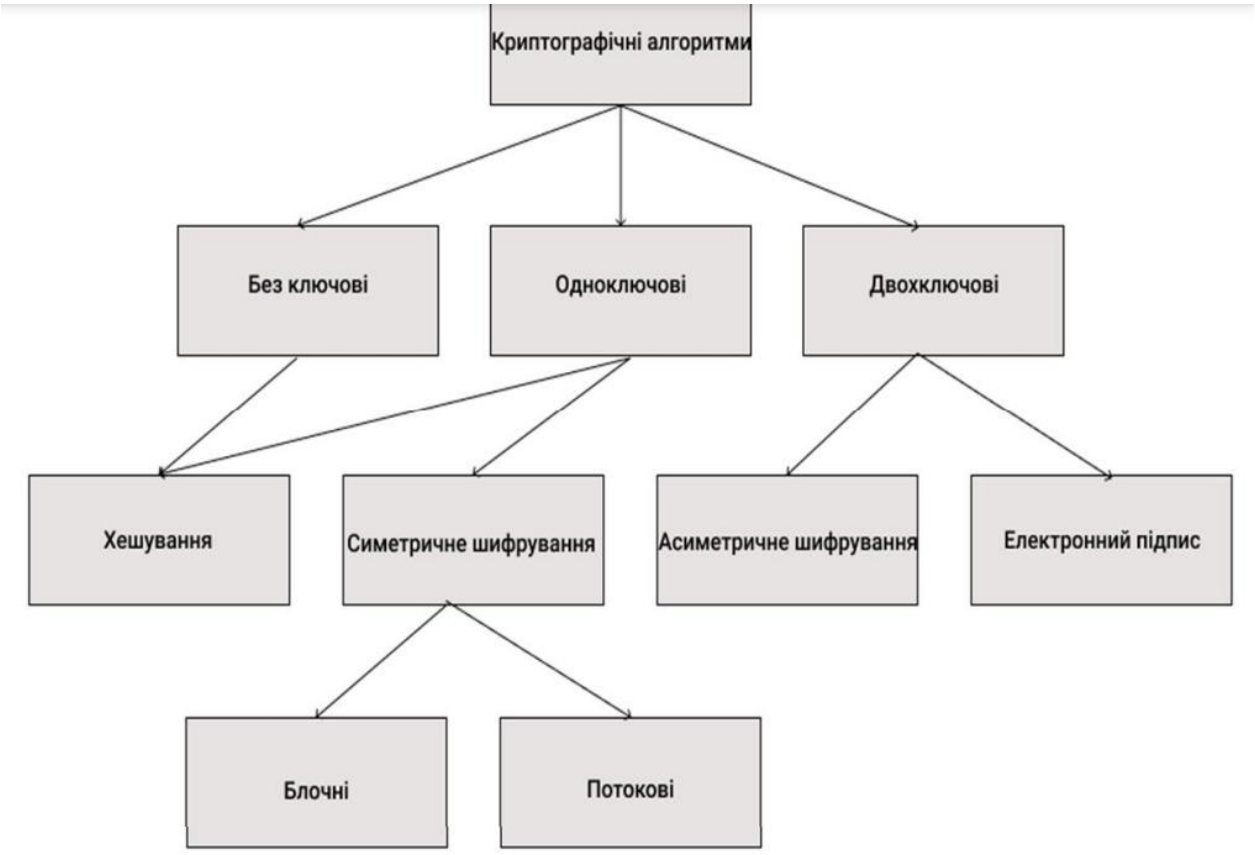


Рисунок *3 — Класифікація криптографічних алгоритмів*

Одноключові алгоритми (хешування та симетричне шифрування) використовують один і той самий ключ як для шифрування, так і для розшифрування. Вони забезпечують швидке шифрування, але вимагають безпечного обміну ключами.

Симетричне шифрування – це метод, при якому відправник і отримувач використовують один і той самий секретний ключ. Використовується для захисту переданих даних у системах зв'язку. Поділяються на два підтипи:

блокові шифри та потокові шифри. Приклади класичних симетричних шифрів:

- Шифр Цезаря.
- Шифр Віженера.
- Шифр перестановки .
- Шифр Вернама.

Блочні шифри оперують фіксованими блоками даних (наприклад, 64 або 128 біт). Дані розбиваються на блоки, кожен з яких шифрується окремо.

Приклади блочних шифрів:

- DES (Data Encryption Standard).
- Triple DES (3DES).
- AES (Advanced Encryption Standard).
- IDEA (International Data Encryption Algorithm).
- Blowfish.
- Twofish.

Потокові шифри обробляють потік даних побітово або побайтово. Вони особливо ефективні для роботи з потоками інформації, такими як відео або аудіо.

Приклади потокових шифрів:

- Гамування з псевдовипадковою послідовністю.
- RC4 (Rivest Cipher 4).
- ChaCha20.
- Salsa20.
- SEAL (Software-Optimized Encryption Algorithm).

Двоключові алгоритми (асиметричне шифрування та цифровий підпис) використовують пару ключів: відкритий та закритий. Один з них використовується для шифрування, інший – для розшифрування. У цьому випадку закритий ключ не передається, зберігається в таємниці власником і використовується лише ним. Перехоплення відкритого ключа не дозволяє зловмиснику розшифрувати зашифрований текст, що забезпечує додаткову

безпеку передачі даних.

Асиметричне шифрування базується на використанні пари ключів: відкритого та закритого. Використовується для захищеного обміну даними та цифрових підписів. Приклади алгоритмів асиметричного шифрування:

- RSA (Rivest-Shamir-Adleman).
- Diffie-Hellman (DH).
- ElGamal.
- ECDSA (Elliptic Curve Digital Signature Algorithm).

Електронний підпис – це криптографічний механізм, що дозволяє перевірити автентичність документів або повідомлень. Він використовується для захисту від підробки.

Приклади електроних підписів:

- RSA Digital Signature (RSA-DSA).
- DSA (Digital Signature Algorithm).
- ElGamal Digital Signature.
- EdDSA (Edwards-curve Digital Signature Algorithm).
- SPHINCS+ (Stateless Post-Quantum Signature).
- CRYSTALS-DILITHIUM.

Ця класифікація допомагає зрозуміти принципи роботи криптографічних алгоритмів і їхнє застосування у сфері інформаційної безпеки. У дипломному проєкті будуть реалізовані наступні алгоритми.

1.3 Частотний аналіз у криптоаналізі

Частотний аналіз — метод криптоаналізу, заснований на статистичному дослідженні частоти появи символів у зашифрованому тексті [5]. Вперше він був описаний арабським вченим Аль-Кінді в 9 столітті і широко використовувався для зламу замінних шифрів. Метод передбачає порівняння частоти символів у зашифрованому тексті з відомими статистичними частотами символів у мові (наприклад, в англійській мові найпоширенішою є

буква «Е»).

Цей метод ефективно працює проти моноалфавітних шифрів, таких як шифр Цезаря, де кожна літера замінюється тією самою іншою буквою в тексті. Наприклад, якщо найпоширенішим символом у зашифрованому тексті є "K", а в англійській мові найчастіше зустрічається "E", то можна вважати, що "K" є "E", а потім розшифрувати, вибравши зсув.

У багатоалфавітних шифрах, таких як шифр Віженера, ефективність частотного аналізу значно знижена, але все ще можлива - за допомогою аналізу повторів і пошуку довжини ключа.

Частотний аналіз є одним з основних інструментів для дослідження слабких місць криптографічних алгоритмів і підвищення їх стійкості шляхом ускладнення структури шифру.

1.4 Класичні шифри

1.4.1 Шифр Цезаря

Історія: один із найдавніших шифрів, приписуваний Юлію Цезарю, який використовував його для таємного листування під час своїх військових кампаній приблизно в 58 році до нашої ери [3].

Як це працює: шифр Цезаря — це моноалфавітний шифр заміни, у якому кожна літера замінюється іншою літерою на фіксовану кількість позицій нижче в алфавіті.

Формула шифрування: $E(x) = (x + k) \bmod N$,

де:

- x – порядковий номер літери в алфавіті;
- k — ключ (зсув);
- N – кількість літер в алфавіті.

Розшифровка: $D(x) = (x - k) \bmod N$.

Переваги:

- легко реалізувати;

— легендарна історична цінність.

Недоліки:

— низька стійкість до криптоаналізу;

— легко зламати за допомогою частотного аналізу.

Як обійти: достатньо проаналізувати частоту появи літер у зашифрованому тексті, щоб вгадати ключ.

Застосування: сьогодні використовується лише як навчальний приклад або для розваги.

1.4.2 Шифр Віженера

Історія: запропонований Блезом де Віженером у 16 столітті [6]. Відомий як «незламний шифр» до 19 століття.

Принцип роботи: багатоалфавітний шифр підстановки. Кожна літера шифрується за допомогою окремого зсуву, визначеного ключовим словом.

Формула: $E(i) = (P(i) + K(i \bmod m)) \bmod N$,

де:

— P(i) – і-та літера відкритого тексту;

— K(i) – і-та літера ключа;

— m – довжина ключа;

— N – кількість символів в алфавіті.

Розшифровка: $D(i) = (C(i) - K(i \bmod m)) \bmod N$.

Переваги:

— більш стійкий до частотного аналізу у порівнянні з Цезарем;

— простий у реалізації.

Недоліки:

— слабкий проти методів Каскіна (повторний аналіз).

Як обійти: метод Каскіна дозволяє знайти довжину ключа та розкласти шифр на кілька шифрів Цезаря.

Застосування: як і шифр Цезаря, використовується для навчання.

1.4.3 Шифр колонної перестановки

Історія: використовувався у військових цілях з давніх часів і в середні віки [6].

Принцип роботи: літери записуються в таблицю по рядках, потім зчитуються по стовпцях відповідно до послідовності клавіш.

Переваги:

— простий у виконанні;

— легко поєднується з іншими шифрами.

Недоліки:

— легко криптоаналізується методом грубої перестановки з коротким ключем.

Як обійти: за допомогою аналізу перестановки та частотного аналізу.

Застосування: застарілий метод, але використовується для навчання або в нестандартних системах.

1.4.4 Гамування з ПВП (псевдовипадковою послідовністю)

Історія: його почали використовувати в 20 столітті з появою електронних потокових шифрів [5].

Принцип роботи: кожен біт відкритого тексту обробляється XOR з бітами псевдовипадкової послідовності, згенерованої генератором PVP (наприклад, на основі LZS або блочного шифру в режимі CTR).

Формула шифрування: $C(i)=P(i)\oplus S(i)$,

де:

— C(i) - і-й біт (або символ) зашифрованого повідомлення (шифртексту);

— P(i) - і-й біт (або символ) відкритого тексту (відкритого тексту);

— S(i) — і-й біт (або символ) псевдовипадкової послідовності (гама).

Формула розшифровки: $P(i)=C(i)\oplus S(i)$.

Операція XOR (⊕) має властивість оборотності: $a\oplus b\oplus b=a$ тобто, застосовуючи XOR знову з тим самим S(i), ми повернемося до початкового P(i). S(i) — це послідовність бітів, згенерована генератором псевдовипадкової

послідовності (PSR), наприклад:

- Лінійний регістр зсуву (LFSR).
- Блочний шифр у режимі CTR або OFB (наприклад, AES у режимі CTR).
- Спеціалізовані генератори потокового шифру (Salsa20, ChaCha20, RC4 тощо).

Ключ шифрування використовується як початкове число для генератора PSR, який генерує потік S(i).

Переваги:

- висока швидкість шифрування;
- підходить для потокової передачі даних (аудіо, відео).

Недоліки:

- небезпечний під час повторному використанню ключів (проблема "повторення ключів").

Як обійти: якщо використовувати той самий PVP для кількох повідомлень, відкритий текст можна знайти шляхом XOR між двома зашифрованими текстами.

Застосування: основний принцип у сучасних потокових шифрах (RC4, Salsa20 тощо).

1.5 Симетричні шифри

1.5.1 DES (Data Encryption Standard)

Історія: Алгоритм DES (стандарт шифрування даних) був розроблений компанією IBM у 1975 році та затверджений як федеральний стандарт шифрування в Сполучених Штатах у 1977 році (FIPS PUB 46) [3]. DES широко використовувався в банківських системах, урядових установах та комерційному програмному забезпеченні до кінця 1990-х років.

Як це працює: DES — це симетричний блоковий шифр, який працює з 64-бітними блоками та використовує 56-бітний ключ (фактично 64 біти, 8 з яких — біти парності). DES базується на мережі Фейстеля та складається з 16

раундів.

У кожному раунді виконуються такі кроки:

— початкова перестановка IP;

— поділ блоку на дві половини (L та R);

— функція f(R, K), яка включає:

- розширення до 48 бітів (E-розширення);
- XOR з підключем;
- S-блокова заміна (8 блоків по 6 → 4 біти);
- P-перестановка;

— заміна половин: $L_i = R_{i-1}$, $R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$;

— фінальна перестановка (FP), яка є оберненою до IP.

Схема шифрування: C = DES_Encrypt(P, K),

де:

— P – відкритий текст;

— K – симетричний ключ (56 бітів);

— C – зашифрований текст.

Формула дешифрування: P = DES_Decrypt(C, K).

Переваги:

— простота впровадження;

— колись вважалася фактичним стандартом.

Недоліки:

— мала довжина ключа (56 біт) — вразливість до грубої сили;

— недостатня стійкість до сучасних криптоаналітичних атак;

— не відповідає сучасним стандартам безпеки.

Як обходять:

— метод “грубої сили” (brute-force): у 1998 році спеціалізоване обладнання зламало DES приблизно за 3 дні;

— диференціальний та лінійний криптоаналіз.

Застосування:

- історично — захист банківських транзакцій;
- у сучасних системах — лише в обмежених сценаріях (наприклад, у застарілих системах або як частина Triple DES).

1.5.2 DES (Triple DES – TDES)

Історія: алгоритм DES (Стандарт шифрування даних) був розроблений у 1975 році компанією IBM і стандартизований урядом США в 1977 році як федеральний стандарт шифрування [3]. Оскільки DES із 56-бітним ключем став уразливим до атак грубої сили, у 1998 році була запропонована вдосконалена версія, Triple DES (TDES). Вона заснована на застосуванні алгоритму DES тричі до кожного блоку даних.

Як це працює: TDES — це блоковий симетричний алгоритм, який працює в 64-бітних блоках і використовує три послідовні етапи DES (шифрування-дешифрування-шифрування або потрійне шифрування) з трьома ключами. Це збільшує довжину ключа до 168 біт (3 ключа по 56 біт).

*Схема шифрування:*C = E_K3(D_K2(E_K1(P))),

де:

- P — відкритий текст;
- E_K — шифрування за допомогою DES з ключем K;
- D_K — дешифрування за допомогою DES з ключем K;
- C — шифртекст.

Формула дешифрування: P = D_K1(E_K2(D_K3(C))).

У деяких реалізаціях використовують лише два ключі (двохключовий режим), але це трохи знижує стійкість [7].

Переваги:

- набагато міцніший, ніж класичний DES;
- відповідає стандартам захисту фінансових операцій (банківські системи, платіжні термінали).

Недоліки:

- повільний через потрійний DES;

— вважається застарілим порівняно з сучасними шифрами (наприклад, AES);
— не рекомендується використовувати після 2023 року відповідно до рекомендацій NIST.

Як обходять:

— методом "зменшення до DES" при використанні слабких ключів (наприклад, коли всі три ключі однакові);
— частково вразливий до атаки "meet-in-the-middle", що дозволяє скоротити перебір при використанні двох ключів.

Застосування:

— раніше використовувався в протоколах SSL/TLS;
— банківські мережі (зокрема картки EMV та SWIFT);
— захист банкоматів і POS терміналів.

1.5.3 AES (Advanced Encryption Standard)

Історія: алгоритм AES був прийнятий як стандарт симетричного шифрування в 2001 році Національним інститутом стандартів і технологій США (NIST) після відкритого конкурсу, який тривав з 1997 по 2000 рік. Переможцем став алгоритм Rijndael, розроблений бельгійськими криптографами Вінсентом Райманом і Джоан Даймен. AES замінив застарілий DES/TDES [3].

Як це працює: AES — це симетричний блоковий шифр, який працює на 128-бітних блоках і підтримує три розміри ключа: 128, 192 і 256 бітів.

Шифрування складається з кількох раундів:

— 10 раундів для 128-бітного ключа;
— 12 раундів для 192-бітного ключа;
— 14 раундів для 256-бітного ключа.

Кожен раунд включає:

— SubBytes — нелінійна заміна байтів через S-box.
— ShiftRows — циклічне зміщення рядків.

— MixColumns — змішування даних у стовпцях.

— AddRoundKey — XOR з круглим ключем.

Схема шифрування: C = AES_Encrypt (P, K),

де:

— P — відкритий текст;

— K — секретний симетричний ключ;

— C — зашифрований текст.

Формула розшифровки: P = AES_Decrypt(C, K).

Переваги:

— висока швидкість на програмному та апаратному рівнях;

— захист від усіх відомих видів атак (лінійний криптоаналіз, диференціальний аналіз);

— визнано міжнародним стандартом.

Недоліки:

— вразливість до атак на сторонньому каналі (наприклад, аналіз споживання енергії при поганій реалізації);

— вимагає ретельного застосування режимів шифрування (наприклад, CBC, CTR).

Як обійти:

— атаки на стороні каналу (power analysis, timing attacks) при ненадійній реалізації апаратної криптографії;

— інші слабкі сторони можливі лише у разі неправильного застосування (наприклад, повтор IV в режимах CBC або CTR).

Застосування:

— захист веб-трафіку (TLS/SSL);

— бездротові мережі (WPA2/WPA3);

— VPN, дискові файли, блокчейн-системи (наприклад, у криптогаманцях).

1.6 Асиметричні шифри

1.6.1 RSA (Rivest–Shamir–Adleman)

Історія: алгоритм RSA був створений у 1977 році Рональдом Рівестом, Аді Шаміром і Леонардом Адлеманом у Массачусетському технологічному інституті. Це перший практичний асиметричний (з відкритим ключем) алгоритм шифрування, заснований на складності розкладання великих чисел. RSA швидко став стандартом безпечної передачі даних і цифрових підписів [3].

Як це працює: RSA використовує пару ключів:

— Відкритий ключ (n, e) для шифрування.

— Закритий ключ (n, d) для розшифровки.

— n — добуток двох великих простих чисел (p і q), які зберігаються в секреті.

Схема шифрування: C = P^e mod n,

де:

— P — числове представлення відкритого тексту;

— e – експонента відкритого ключа;

— n – модуль, спільний для відкритого та закритого ключів;

— C - це зашифрований текст.

Формула розшифровки: P = C^d mod n.

де d — експонента закритого ключа.

Переваги:

— не вимагає спільного секретного ключа;

— використовується для цифрових підписів і шифрування.

Недоліки:

— повільний порівняно з симетричними шифрами;

— обмеження розміру зашифрованих даних (не більше розміру ключа в бітах).

Як обійти:

— атаки на погано згенеровані прості числа (факторізація слабкого n);

— атаки через “low exponent” (наприклад, при e=3 без padding).

Застосування:

— протоколи TLS/SSL;

— електронна комерція (захист банківських операцій);

— цифрові підписи (PKI).

1.6.2 ECDSA (Elliptic Curve Digital Signature Algorithm)

Історія: алгоритм ECDSA був схвалений як стандарт у 1999 році Національним інститутом стандартів і технологій США (NIST). Це адаптація класичного алгоритму цифрового підпису DSA до криптографії на еліптичній кривій (ECC). Розвиток ECDSA пов'язаний із загальною тенденцією переходу до еліптичних кривих через їхню високу криптографічну міцність із меншою довжиною ключа [8].

Як це працює: ECDSA базується на обчислювальній складності задачі дискретного логарифмування на еліптичних кривих. Алгоритм дозволяє створювати та перевіряти цифрові підписи для забезпечення цілісності та автентичності даних.

Основні етапи роботи ECDSA:

1. Генерація пари ключів:

— приватний ключ - випадкове число d;

— відкритий ключ - точка $Q = d \times G$, де G - твірна точка на еліптичній кривій.

2. Підпис повідомлення:

— розраховується хеш повідомлення H(m);

— генерується випадкове число k;

— обчислюються дві сигнатурні складові: r і s, які залежать від k, d і H(m).

3. Перевірка підпису:

— відкритий ключ Q і підпис (r, s) використовуються для перевірки автентичності H(m).

Формули (спрощені):

— $r = (k \times G)_x \bmod n$,

— $s = k^{-1}(H(m) + d \times r) \bmod n$,

де:

— k випадкове число;

— n - порядок групи точок на кривій.

Переваги:

— коротші ключі з однаковою міцністю (наприклад, 256-бітний ключ ECDSA ≈ 3072-бітний ключ RSA);

— швидке створення та перевірка підпису;

— ефективний для вбудованих систем і технологій блокчейн.

Недоліки:

— більш складний математичний апарат;

— критично залежить від надійності випадкової генерації k (злом можливий, коли k повторюється).

Як обійти:

— атаки на погано реалізований генератор k (приклад: історичний злом PlayStation 3);

— атаки на генератор ключів зі сторонніх каналів.

Застосування:

— криптовалюти (Bitcoin, Ethereum) [4];

— протоколи TLS/SSL;

— електронні сертифікати та PKI.

1.7 Алгоритми отримання хеш-сум

1.7.1 MD5 (Message Digest Algorithm 5)

Історія: алгоритм MD5 був розроблений Рональдом Рівестом у 1991 році як вдосконалена версія MD4. Його метою було створити хеш-функцію для забезпечення цілісності даних і захисту паролем. Протягом багатьох років

MD5 був одним з найпопулярніших хеш-алгоритмів у сфері інформаційної безпеки [22].

Як це працює: MD5 — це криптографічна хеш-функція, яка приймає повідомлення довільної довжини та повертає хеш-значення 128 біт (16 байт). MD5 працює, розбиваючи вхідні дані на 512-бітні блоки, які потім піддаються чотирьом раундам перетворень, кожен з яких включає кілька нелінійних функцій і побітових операцій.

Основні етапи MD5:

1. Доповнення повідомлення до 512 біт.
2. Розбиття на 512-бітні блоки.
3. Ініціалізація 4 регістрів A, B, C, D (по 32 біта).
4. 64 раунди побітових перетворень з використанням нелінійних логічних функцій, констант і циклічних зрушень.
5. Об’єднання результатів у 128-бітний хеш.

Переваги:

- дуже швидка обробка;
- легко реалізувати;
- широка підтримка у всіх популярних бібліотеках.

Недоліки:

- уразливість до колізій (можливість вибору різних даних, які дадуть однаковий хеш);
- недостатня криптографічна стійкість для сучасних систем.

Як обійти:

- підбір колізій з використанням атак Ван Різенена (1996) і подальших покращених атак;
- використання Rainbow Tables для зворотного пошуку паролів, захищених лише MD5.

Застосування:

- перевірка цілісності файлу (контрольні суми);
- внутрішні системи безпеки в застарілих системах;

— у сучасних системах MD5 не рекомендується для криптографічного захисту паролів або підписів.

1.7.2 SHA-256 (Secure Hash Algorithm 256)

Історія: алгоритм SHA-256 був розроблений Національним інститутом стандартів і технологій США (NIST) у 2001 році як частина сімейства алгоритмів SHA-2. Він був створений для заміни застарілого SHA-1 після виявлення його криптографічної вразливості. Сьогодні SHA-256 є одним із найпоширеніших хеш-алгоритмів у криптографії [23].

Як це працює: SHA-256 — це криптографічна хеш-функція, яка приймає вхідне повідомлення довільної довжини та генерує хеш із 256 біт (32 байти). Алгоритм обробляє дані блоками по 512 бітів і проходить 64 цикли з використанням нелінійних функцій, констант і порозрядних операцій (XOR, AND, OR, зсув і обертання).

Основні кроки:

1. Доповнення повідомлення до кратних 512 біт.
2. Розбиття повідомлення на блоки по 512 біт.
3. Ініціалізація 8 регістрів (A, B, C, D, E, F, G, H).
4. 64 раунди перетворень з використанням констант і функцій SHA-256.
5. Вивід хешу довжиною 256 біт.

Переваги:

— висока стійкість до колізій;

— використовується у всіх сучасних протоколах безпеки (TLS, IPsec, VPN, блокчейн);

— великий вихідний розмір хешу (256 біт) забезпечує високий рівень захисту.

Недоліки:

— повільніше порівняно з MD5 або SHA-1;

— вразливий до атак із сторонніх каналів із поганою реалізацією (наприклад, на вбудованих пристроях).

Як це обійти:

- наразі немає ефективних атак проти SHA-256 для практичних застосувань;

Застосування:

- системи блокчейн (Bitcoin, Ethereum);
- цифрові підписи та електронні сертифікати;
- SSL/TLS, IPsec, VPN.

1.8 Криптографічні засоби платформи Java

У процесі дослідження криптографічних засобів платформи Java було проаналізовано ряд літературних джерел, що стосуються як загальної теорії криптографії, так і особливостей реалізації криптографічних алгоритмів мовою Java.

У монографії C. Paar та J. Pelzl "Understanding Cryptography: A Textbook for Students and Practitioners" розглядаються фундаментальні поняття криптографії, а також класифікація та принципи роботи сучасних симетричних та асиметричних алгоритмів, алгоритмів хешування, цифрових підписів тощо. Значну увагу приділено реалізації таких алгоритмів, як AES, RSA, ECC та SHA-2, який безпосередньо пов’язаний з розробкою цього веб-застосунку [8].

Джерелом, що поглиблює знання про реалізацію криптографічних інструментів у середовищі Java, стала книга Knudsen "Java Cryptography", в якій детально розглядаються Java Cryptography Architecture (JCA), Java Cryptography Extension (JCE), інтерфейси API для шифрування, генерації ключів, хешування та цифрових підписів. Видання також містить практичні приклади роботи з актуальною для даного проекту бібліотекою Bouncy Castle [9].

Практичний аспект використання криптографії в Java висвітлюється в книзі David Hook “Beginning Cryptography with Java”, де розглядаються типові

приклади використання криптографічних алгоритмів у захищених програмних рішеннях на платформі Java. Автор акцентує увагу на інтеграції бібліотеки Bouncy Castle у застосунки та практичній реалізації базових алгоритмів шифрування та цифрових підписів [10].

Офіційні документи та специфікації від Oracle щодо безпеки Java SE також важливі, включаючи документацію для JCA та JCE. Ці документи містять стандарти для реалізації криптографічних API, приклади використання класів ключів і механізмів для створення безпечних веб-застосунків [11].

Таким чином, наведені вище джерела стали основою для вивчення можливостей платформи Java у сфері криптографії та безпосередньо вплинули на вибір засобів і бібліотек для реалізації захисту даних у рамках розробленого веб-застосунок.

1.9 Аналіз аналогічних програмних продуктів

Для оцінки аналогічних програмних продуктів, що використовуються для навчання криптографічних алгоритмів, були обрані наступні рішення: CrypTool, CrypTool-Online та CrypTool 2. Основним критерієм відбору є їх популярність для навчальних цілей та функціональність.

1.9.1 CrypTool

CrypTool — це популярне безкоштовне освітнє програмне забезпечення з відкритим кодом, призначене для демонстрації та вивчення різних методів криптографії та криптоаналізу [12]. CrypTool призначений як для студентів, так і для викладачів, а також для фахівців з інформаційної безпеки.

CrypTool підтримує велику кількість алгоритмів:

- класичні алгоритми шифрування (шифр Цезаря, шифр Віженера, перестановки, шифр Вернама);
- симетричні алгоритми (AES, DES, Blowfish);

— асиметричні алгоритми (RSA, DSA, ECC);

— хеш-алгоритми (MD5, SHA-1, SHA-256, Whirlpool та ін.);

— цифрові підписи та інші криптографічні протоколи.

CrypTool забезпечує візуалізацію криптографічних процесів і дозволяє спостерігати етапи шифрування та дешифрування даних на практиці. Програма має інтерактивний інтерфейс із прикладами, які супроводжуються діаграмами та покроковими демонстраціями.

Аналіз плаваючої частоти символів у шифротексті представлено на рисунку 4, що дозволяє виявити закономірності та слабкі місця в алгоритмах шифрування.

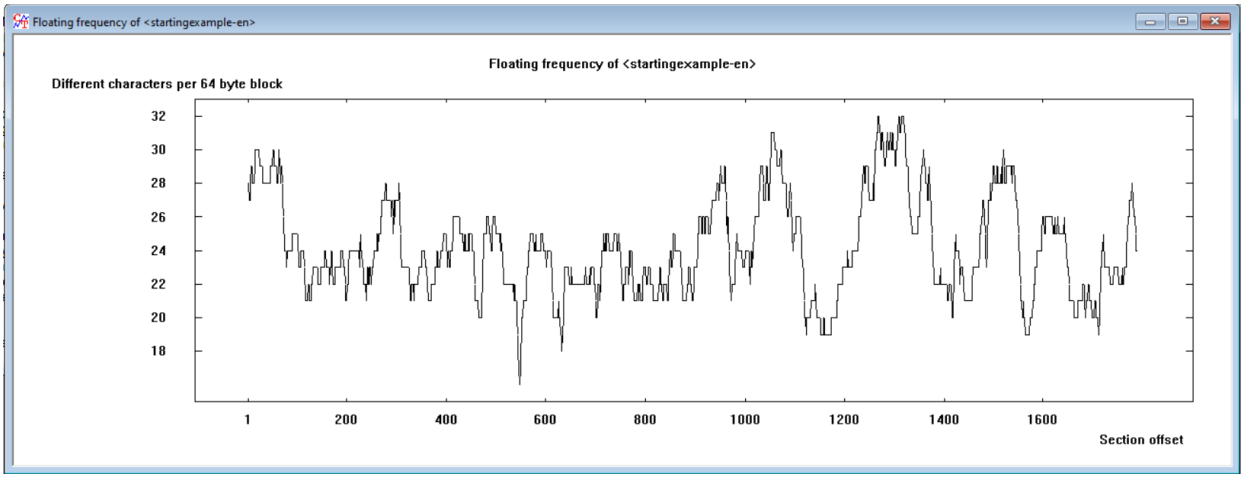


Рисунок *4 — Аналіз плаваючої частоти символів у шифротексті*

CrypTool має вбудовану модель для імітації роботи шифрувальної машини Enigma, яка представлена на рисунку 5, що дозволяє наочно побачити рух сигналу через ротори та зрозуміти принципи її роботи.

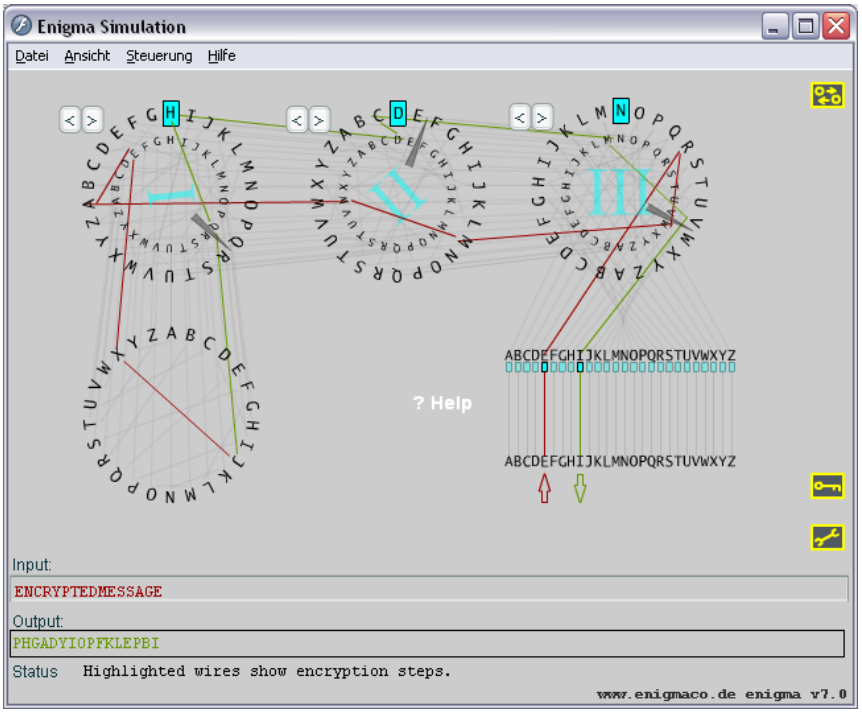


Рисунок *5* — *Симуляція роботи шифрувальної машини Enigma у CrypTool*

CrypTool підтримує 3D-візуалізацію процесів шифрування, яка представлена на рисунку 6.

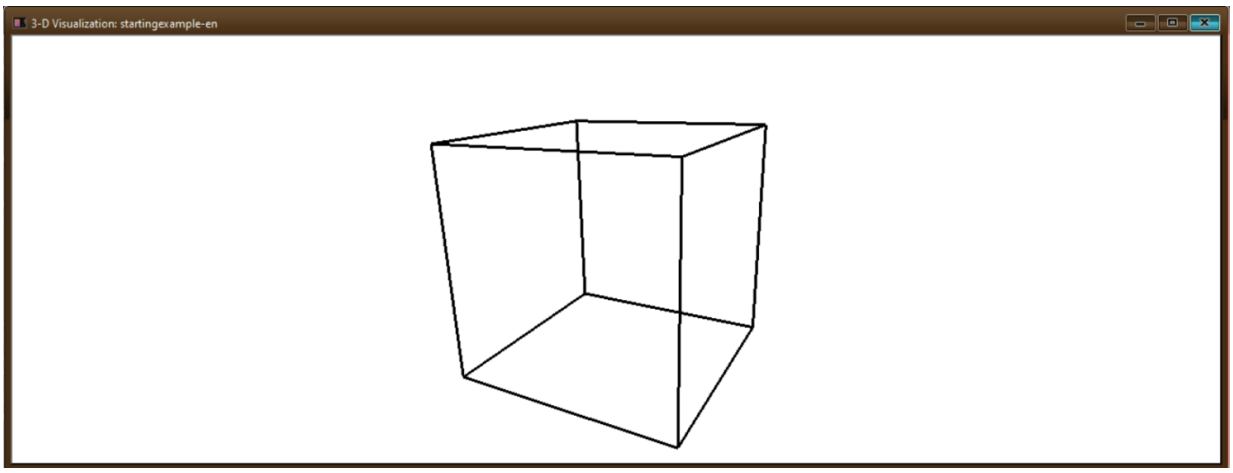


Рисунок *6* — *3D-візуалізація процесу шифрування у CrypTool*

Програма надає можливість візуалізувати роботу алгоритму RSA, як показано на рисунку 7, демонструючи процеси генерації ключів, шифрування та дешифрування даних.

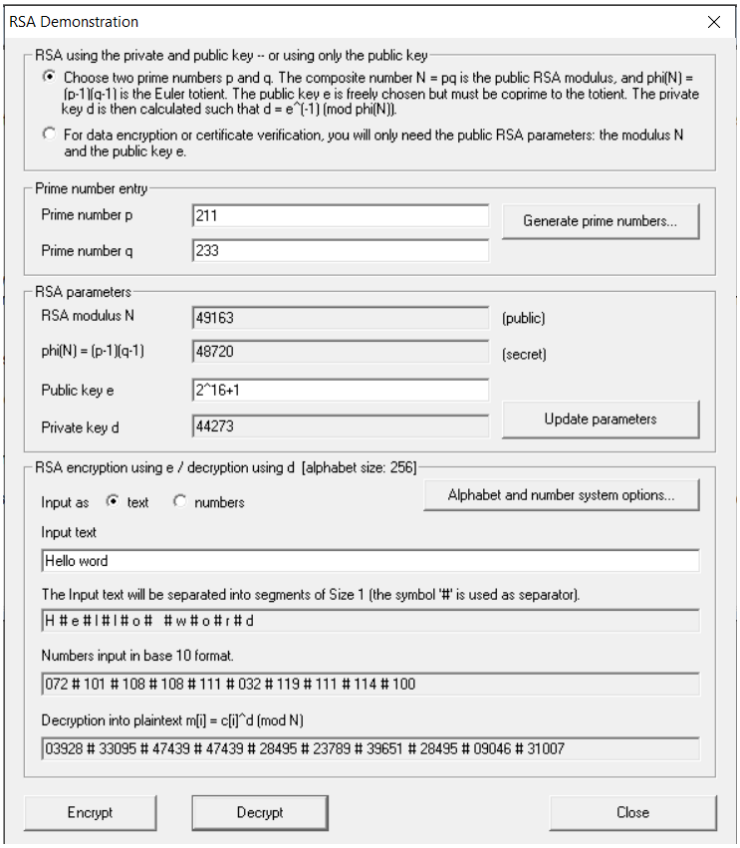


Рисунок *7 — Візуалізація роботи алгоритму RSA у CrypTool*

Детальніше про можливості програми можна дізнатися на офіційному сайті [13].

1.9.2 CrypTool-Online

CrypTool-Online — це веб-версія відомої платформи CrypTool, яка не потребує встановлення програмного забезпечення та працює безпосередньо в браузері. Основною метою CrypTool-Online є надання інтерактивних інструментів для вивчення криптографії онлайн, що дозволяє користувачам швидко тестувати різні криптографічні алгоритми.

CrypTool-Online підтримує широкий спектр алгоритмів:

- класичні шифри: Caesar cipher, ROT13, Enigma, Playfair;
- сучасні симетричні шифри: AES, RC4;
- асиметричні алгоритми: RSA;
- хешування: MD5, SHA-1, SHA-256;

— цифровий підпис: RSA.

Користувачі можуть вибрати потрібний алгоритм, ввести відкритий текст і спостерігати за результатами шифрування і дешифрування в реальному часі.

Особливістю CrypTool-Online є можливість перегляду вихідного коду кожного алгоритму, реалізованого на мові програмування Python. Це дозволяє учням не тільки побачити результат шифрування, а й вивчити логіку алгоритму на рівні коду.

Приклад роботи з класичними шифрами

Одним із популярних класичних алгоритмів, які підтримує CrypTool-Online, є шифр Цезаря. Після введення тексту користувач може вибрати зміщення (наприклад, +3), переглянути результат шифрування та переглянути код Python, який реалізує цей алгоритм. Модуль шифру Цезаря з прикладом коду представлено на рисунку 8.

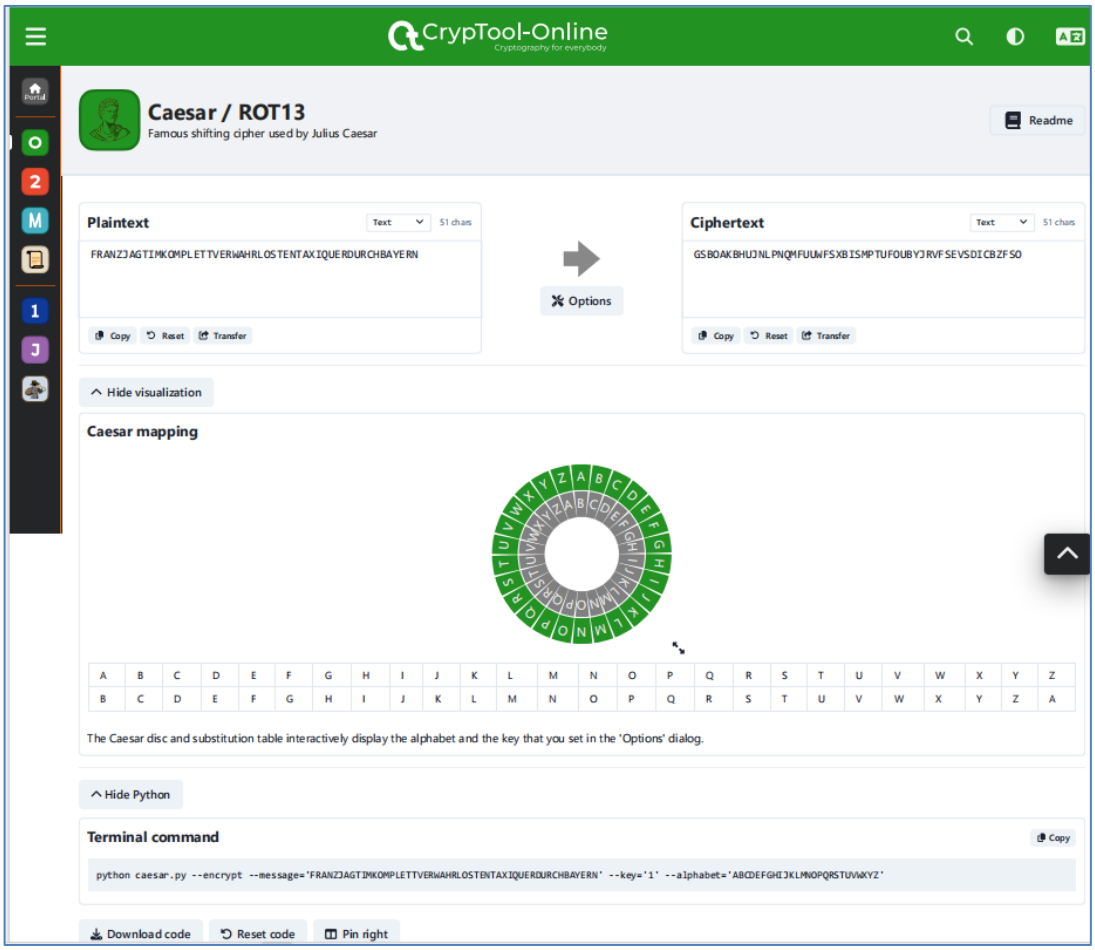


Рисунок *8 — Модуль шифру Цезаря в CrypTool-Online із кодом на Python*

Симулятор Enigma

CrypTool-Online також містить емулятор шифрувальної машини Enigma. У цьому модулі користувач може вибрати налаштування ротора, рефлектора, стартову позицію та перевірити шифрування повідомлень, як це робили оператори Enigma у 20 столітті. Крім того, модуль надає фрагмент коду Python, що демонструє базову реалізацію принципів Enigma. Цей функціонал показано на рисунку 9.

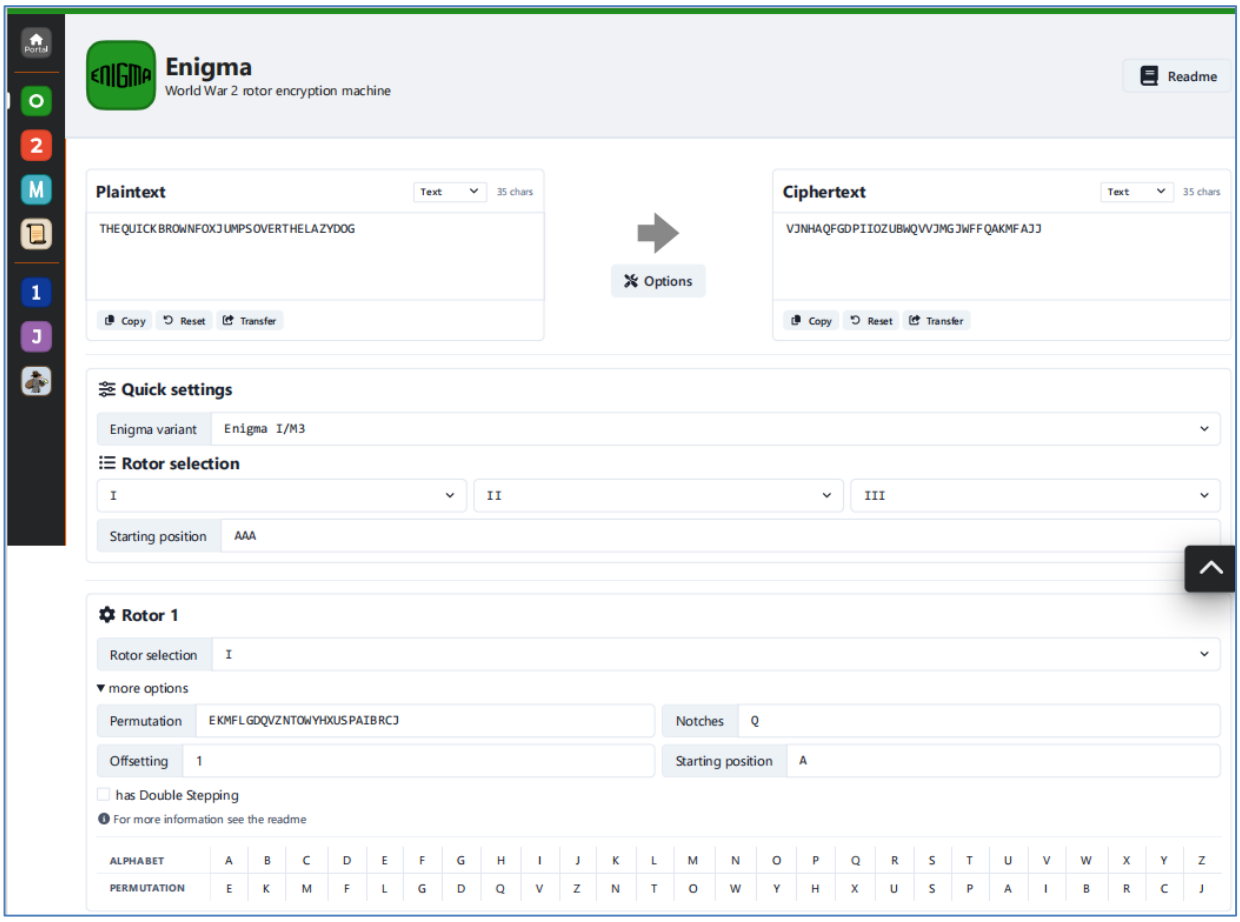


Рисунок *9 — Симуляція шифру Enigma в CrypTool-Online із доступом до коду на Python*

Детальніше про можливості програми можна дізнатися на офіційному сайті [14].

1.9.3 CrypTool 2 (CT2)

CrypTool 2 (CT2) — це сучасна версія CrypTool з модульною

архітектурою та графічним інтерфейсом для побудови криптографічних сценаріїв [15]. CT2 дозволяє візуально моделювати процеси шифрування, дешифрування та криптоаналізу за допомогою інтерактивних блок-схем.

Основні можливості CrypTool 2:

- підтримка блокових і потокових шифрів: AES, DES, RC4;
- асиметричні алгоритми: RSA, DSA, ECC;
- хеш-функції: MD5, SHA-1, SHA-256, Whirlpool;
- підписання та перевірка цифрових підписів (RSA, ECDSA);
- впровадження сліпого підпису;
- побудова візуальних діаграм для криптографічних процесів;
- візуалізація роботи класичних алгоритмів (Caesar, Vigenère, Enigma).

Особливості CrypTool 2:

- графічний редактор блок-схем для побудови криптографічних процесів;
- покрокова візуалізація кожного етапу обробки даних;
- розширені модулі для інтерактивного налаштування параметрів алгоритму.

Приклади роботи CrypTool 2

Модуль Enigma cipher Machine

У CrypTool 2 реалізовано модуль симуляції роботи шифрувальної машини Enigma з детальним відображенням налаштувань роторів, плугборду та історичної інтерпретації процесу шифрування. Цей функціонал представлено на рисунку 10.

Модуль Vigenère cipher

У CT2 також інтегровано модуль поліалфавітного шифру Віженера з можливістю налаштування ключа та демонстрації всіх етапів шифрування та дешифрування. Цей модуль показано на рисунку 11.

Blind signature with RSA

CrypTool 2 підтримує демонстрацію сліпого підпису на основі RSA. Модуль демонструє процес генерації закритого та відкритого ключів, підписання сліпого повідомлення та подальшої його розшифровки. Цей

процес візуалізовано на рисунку 12.

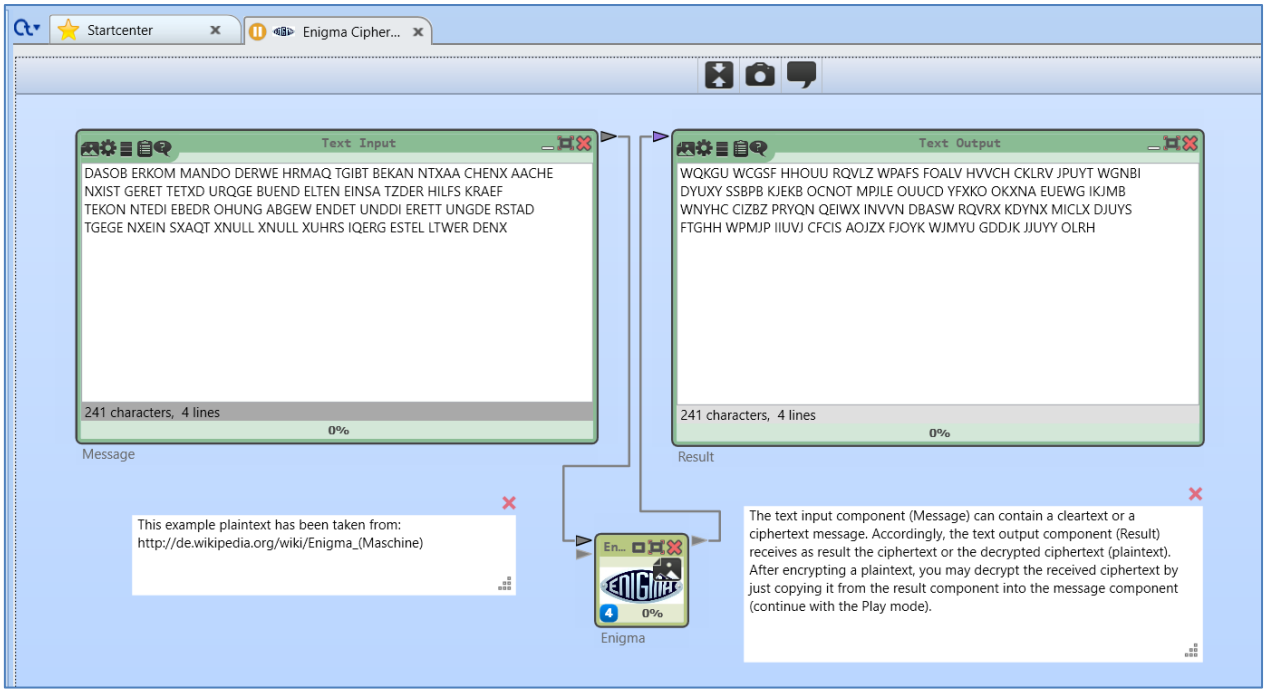


Рисунок *10 — Симуляція шифрувальної машини Enigma у CrypTool 2*

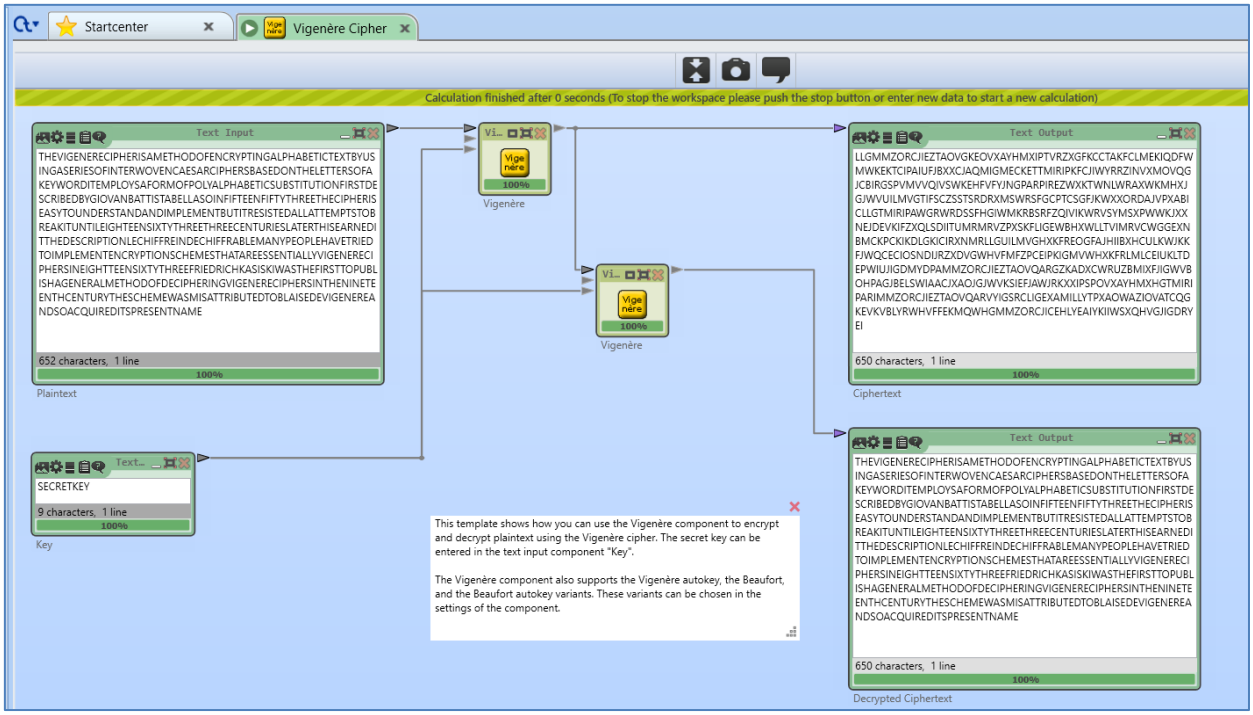


Рисунок *11 — Модуль шифру Віженера в CrypTool 2*

SHA-256 Hash function

CrypTool 2 має окремий модуль для хешування даних алгоритмом SHA-256, який відображає хеш введених даних.(рисунок 13).

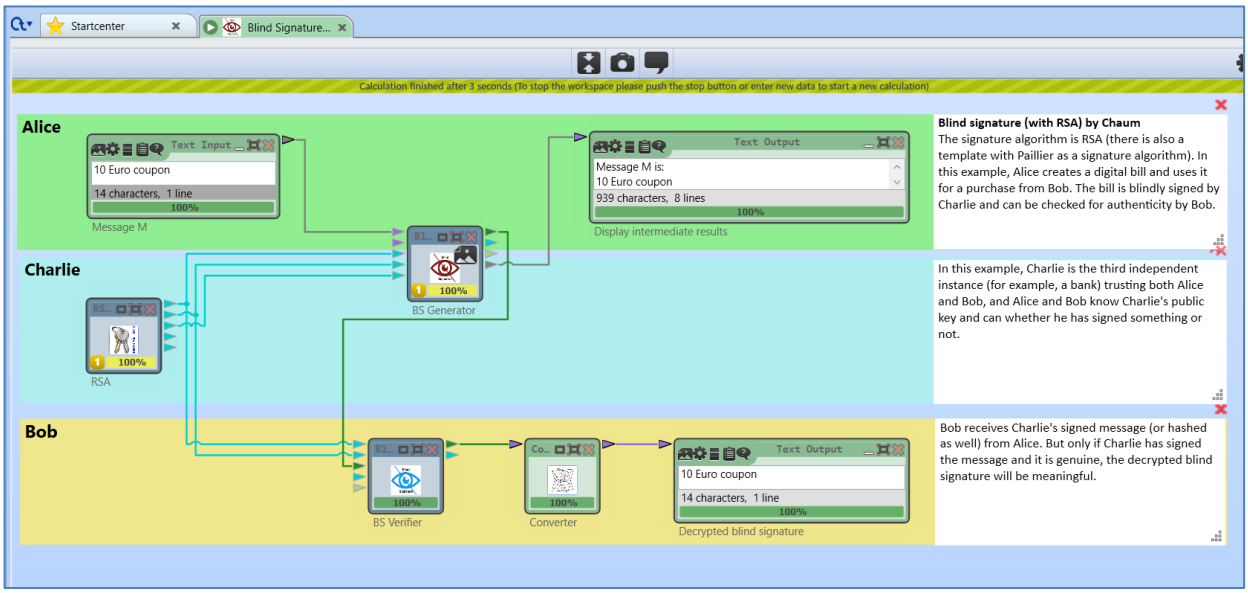


Рисунок *12 — Демонстрація сліпого підпису RSA у CrypTool 2*

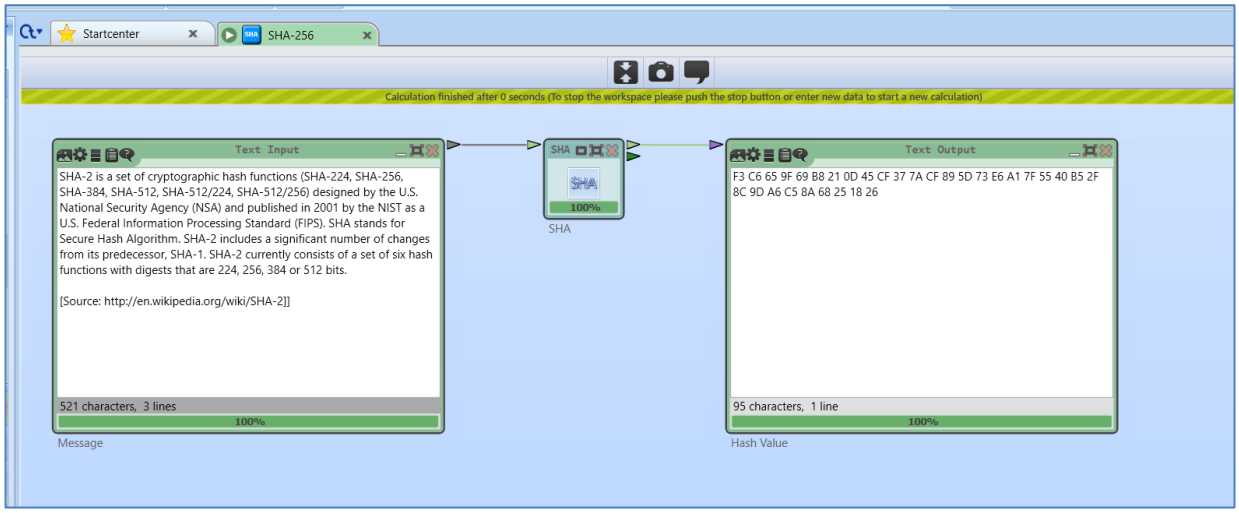


Рисунок *13 — Візуалізація роботи алгоритму SHA-256 у CrypTool 2*

CrypTool 2 — це інструмент, який дозволяє не лише проводити криптографічні експерименти, а й будувати повноцінні інтерактивні сценарії для дослідження процесів шифрування в реальному часі.

Детальніше про можливості програми можна дізнатися на офіційному сайті [15].

1.10 Постановка завдання

На основі проведеного аналізу визначено наступні основні завдання реалізації веб-застосунку:

1. Розробити веб-інтерфейс, що містить навчальні модулі, щоб пояснити, як працюють криптографічні алгоритми.
2. Створити інтерактивні інструменти, які дозволяють користувачам обирати криптографічні алгоритми, вводити дані та спостерігати за результатами.
3. Забезпечити візуалізацію криптографічних процесів, що допоможе користувачам краще зрозуміти принципи криптографії.
4. Впровадити підтримку основних криптографічних алгоритмів відповідно до їх класифікації.
5. Забезпечити безпеку введених користувачами даних шляхом реалізації локальної обробки інформації без передачі її на сервер.
6. Визначити способи реалізації алгоритмів: шляхом самостійної розробки або використання існуючих бібліотек (наприклад, Java Cryptography Architecture (JCA), Bouncy Castle).

2 ВИВЧЕННЯ ЗАСОБІВ SPRING FRAMEWORK ДЛЯ РОЗРОБКИ ВЕБ-ЗАСТОСУНКІВ

2.1 Екосистема Spring та її основні проєкти

Spring Framework — це потужна платформа Java з відкритим кодом, яка спрощує розробку корпоративних, веб та хмарних застосунків [16]. Гнучкість, модульність та масштабованість зробили Spring одним із найпопулярніших фреймворків у середовищі Java. Його ядро базується на концепціях інверсії керування (IoC) та аспектно-орієнтованого програмування (AOP).

Основні проєкти, що входять до екосистеми Spring:

- Spring Core — це базовий модуль, який забезпечує контейнер IoC та управління залежностями.
- Spring MVC — це реалізація архітектурного шаблону Model-View-Controller для веб-застосунків.
- Spring Security забезпечує механізми автентифікації, авторизації та безпеки.
- Spring Data абстрагує роботу з базами даних, підтримує JPA, MongoDB, Redis та інші.
- Spring Boot спрощує створення самодостатніх застосунків завдяки автоматичному налаштуванню.
- Spring Cloud — це набір інструментів для побудови мікросервісної архітектури.
- Spring Test — це інструмент для написання та запуску модульних, інтеграційних та системних тестів.

Spring підтримує інтеграцію з іншими технологіями, такими як Kafka, RabbitMQ, GraphQL, WebSockets, а також з хмарами такими як (AWS, GCP, Azure), що робить його універсальним інструментом.

2.2 Концепції Bean-компонентів. Контекст застосунку та способи додавання Bean-компонентів

У Spring кожен керований об'єкт є Bean-компонентом. Bean-компонент створюється, налаштовується та керується контейнером Spring [17]. Керування включає ініціалізацію, впровадження залежностей, життєвий цикл та знищення (за потреби).

Контекст застосунку (ApplicationContext) – це середовище виконання, яке відповідає за створення та зв'язування Bean-компонентів [18]. Він є розширенням BeanFactory та підтримує такі функції, як події, інтернаціоналізація, AOP тощо.

Основні способи додавання Bean-компонентів:

- Через анотацію @Component — для автоматичного виявлення компонентів під час сканування пакетів.
- Спеціалізовані анотації @Service, @Repository, @Controller — вказують логічну роль компонента.
- Через конфігурацію Java: класи з @Configuration та методи з @Bean дозволяють створювати Bean-компоненти вручну.
- Конфігурація XML (застарілий підхід, але все ще підтримується).

Життєвий цикл Bean-компонента включає такі фази: створення, впровадження залежностей, ініціалізація, використання та знищення.

2.3 Інструменти Spring Framework для створення веб-застосунків

Spring Framework надає розробникам потужні інструменти для створення веб-застосунків, зокрема через модуль Spring Web, який підтримує створення RESTful API за допомогою анотацій @RestController, @RequestMapping, @PostMapping тощо. У своєму класичному вигляді цей модуль використовується в рамках шаблону Model-View-Controller (MVC), де:

- Модель (Model) — бізнес-логіка, включаючи маніпулювання даними та

обчислення.

- Контролер (Controller) — обробляє HTTP-запити та координує взаємодію між моделлю та представленням.
- Представлення (View) — відображає дані користувачеві (наприклад, HTML-сторінки, згенеровані сервером за допомогою шаблонізаторів, таких як Thymeleaf).

Однак веб-застосунок “Crypto Training App” реалізує іншу архітектурну схему: RESTful API з “товстим клієнтом”. У цій моделі:

- Представлення повністю реалізовано на стороні клієнта за допомогою HTML, CSS та JavaScript.
- Серверна сторона (Spring Boot) діє як API, який обробляє запити, виконує криптографічну логіку та повертає результат у фрагментах JSON або HTML.
- Контролери не координують безпосередньо між моделлю та представленням, а радше надають дані клієнту, який відповідає за їх відображення.

Spring Web забезпечує:

- Створення REST API через @RestController.
- Автоматичну обробку JSON-запитів та відповідей.
- Підтримку обробки винятків через @ExceptionHandler.
- Можливість розширення логіки через класи сервісів (Service Layer.;
- Інтеграцію з бібліотеками Java (наприклад, JCA, Bouncy Castle) для реалізації криптографії [19].

Приклад використання Spring Web у проєкті Crypto Training App

Одна з реалізованих функцій – обчислення хешу SHA-256. Для цього було створено REST-контролер HashController, який приймає текст, обчислює хеш та повертає HTML-розмітку з поясненням.

Приклад коду наведенно в лістингу 1.

Лістинг *1 Контролер для обчислення SHA-256 хешу у Spring Boot*

```
package com.cryptoapp.cryptotrainingapp;
```

```
import org.springframework.web.bind.annotation.*;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;

@CrossOrigin
@RestController
@RequestMapping("/api/crypto")
public class HashController {

    @PostMapping("/sha256")
    public String hashSha256(@RequestBody String input)
throws Exception {
        MessageDigest digest =
MessageDigest.getInstance("SHA-256");

        byte[] inputBytes =
input.getBytes(StandardCharsets.UTF_8);
        byte[] hashBytes = digest.digest(inputBytes);

        StringBuilder hexHash = new StringBuilder();
        for (byte b : hashBytes) {
            String hex = Integer.toHexString(0xff & b);
            if (hex.length() == 1) hexHash.append('0');
            hexHash.append(hex);
        }

        StringBuilder explanation = new StringBuilder();
        explanation.append("<div class='explanation'>");

        // Вхідний текст
        explanation.append("<p><b>Вхідне повідомлення:</b>
").append(input).append("</p>");

        // Байтове представлення
        explanation.append("<p><b>Байтове представлення
(UTF-8):</b></p><pre>");
        for (byte b : inputBytes) {
            explanation.append(String.format("%02x ", b));
        }
        explanation.append("</pre>");

        // Хеш результат
        explanation.append("<p><b>Результат хешування (SHA-
256, hex):</b></p>");
        explanation.append("<div
class='mono'>").append(hexHash).append("</div>");

        explanation.append("</div>");
```

```
            return explanation.toString();
        }
}
```

У цьому прикладі демонструється:

- @RestController — позначає клас як REST-контролер.
- @PostMapping(“/sha256”) — обробка POST-запиту.
- MessageDigest — стандартна бібліотека Java для хешування.
- HTML-вивід — генерується динамічно на стороні сервера, але без використання серверного шаблонізатора.

Таким чином, Spring Boot забезпечує реалізацію логіки API, яка обробляється окремим JavaScript-інтерфейсом на клієнті, що відповідає архітектурі “товстого клієнта” з REST-взаємодією. Цей підхід є гнучким, масштабованим та добре адаптованим до сучасних підходів SPA (Single Page Application) або JAMstack.

2.4 Spring Boot та його можливості

Spring Boot спрощує створення веб-застосунків, мінімізуючи конфігурацію та надаючи засоби запуску залежностей [20]. Ключові переваги:

- Автоматичне налаштування залежностей (AutoConfiguration).
- Вбудований веб-сервер (Tomcat, Jetty, Undertow) - окреме розгортання не потрібне.
- Налаштування через application.properties або application.yml.
- Підтримка DevTools - гаряче перезавантаження під час розробки.
- Інтеграція з Spring Security, Spring Data, Spring Web без додаткового налаштування.

Spring Boot дозволяє створювати виконуваний файл .jar або .war, що забезпечує повністю самодостатній застосунок.

2.5 Визначення функціональності застосунку, що реалізується

В рамках дипломного проекту створюється навчальний веб-застосунок на тему криптографії. Його функціональність включає:

- Навчальні модулі: сторінки з описом алгоритмів (AES, RSA, MD5 тощо) та пояснення.
- Інтерактивні інструменти: введення даних, вибір алгоритму та результат у режимі реального часу.
- Візуалізація: графічне відображення процесів шифрування, підстановок, перестановок тощо.
- Тестування: інтерактивні тести для перевірки знань користувачів.
- Безпечна локальна обробка даних на клієнті (JavaScript) без збереження на сервері.
- Мікросервісна архітектура з REST API, реалізована через Spring Boot.

Використання інструментів Spring Boot та бібліотек JCA та Bouncy Castle дозволяє реалізувати надійний, безпечний та гнучкий інструмент для вивчення криптографії.

3 ПРОЕКТУВАННЯ ТА ПОБУДОВА ВЕБ-ЗАСТОСУНКУ ДЛЯ НАВЧАННЯ КРИПТОГРАФІЧНИМ АЛГОРИТМАМ

3.1 Загальні принципи проектування

На сучасному етапі розвитку інформаційних технологій особливу актуальність має необхідність забезпечення доступності та інтерактивності освітніх ресурсів, зокрема у сфері вивчення криптографії.

Для досягнення цієї мети в рамках цієї роботи було розроблено веб-застосунок, орієнтований на демонстрацію принципів роботи криптографічних алгоритмів різних типів: класичних, симетричних, асиметричних та хеш-функцій.

Основними принципами, яких дотримувався при розробці застосунку, були:

- модульність - окремі криптографічні алгоритми реалізуються як незалежні компоненти;
- розширюваність - структура проєкту дозволяє легко додавати нові алгоритми або функціональність;
- інтерактивність - користувач може безпосередньо вводити дані, вибирати алгоритм та отримувати візуалізований результат;
- безпека - обробка даних здійснюється на стороні клієнта без збереження введених даних на сервері;
- використання сучасних інструментів та підходів до розробки веб-застосунків, зокрема фреймворків Spring та Spring Boot.

3.2 Архітектура застосунку

Архітектура веб-застосунку побудована на принципах трирівневої взаємодії клієнт-сервер, що забезпечує розподіл обов'язків між представленням, обробкою запитів та бізнес-логікою.

3.2.1 Рівень представлення (Presentation Layer)

Цей рівень реалізовано за допомогою HTML, CSS та JavaScript. Його завданням є отримання даних від користувача, генерація відповідних запитів до API та відображення результатів. Для кожного криптографічного алгоритму було створено окрему HTML-сторінку, яка містить інтерактивну форму з полями введення даних, кнопками керування та блоком відображення результатів, а також елементи, що відображають етапи криптографічних алгоритмів. Користувач взаємодіє з формами, які надсилають HTTP-запити на сторону сервера методами POST (шифрування, підпис, перевірка) або GET (генерація ключів).

3.2.2 Рівень контролерів (Controller Layer)

Контролери реалізовані за допомогою Spring Boot з використанням анотацій @RestController та @PostMapping/@GetMapping. Вони відповідають за:

- прийняття вхідних HTTP-запитів від клієнта;
- розпакування вхідних даних (JSON, form-data);
- виклик відповідного методу сервісу;
- генерацію та надсилання відповіді клієнту у форматі JSON або HTML.

Приклад: контролер CaesarController приймає текст та зміщення, викликає сервіс CaesarService, отримує об'єкт CaesarResult та повертає його клієнту разом із поясненням.

3.2.3 Рівень бізнес-логіки (Service Layer)

Сервіси реалізують криптографічну логіку: шифрування, дешифрування, генерацію ключів, створення підпису, перевірку підпису тощо. Для реалізації використовуються:

- Бібліотеки Java Cryptography Architecture (JCA) [11].
- Бібліотеки Bouncy Castle сторонніх розробників [19].
- Власні реалізації класичних шифрів (Caesar, Vigenère тощо).

Усі сервіси отримують запити у вигляді об'єктів моделі, обробляють їх та генерують об'єкти результату. Такий розділ забезпечує повторне використання логіки сервісу в тестах та контролерах.

3.2.4 Модель (Model / DTO)

У проєкті реалізовано набір класів, які можна розглядати як модель. Ці класи містять поля для вхідних та вихідних даних (відкритий текст, ключі, параметри шифрування, результат, підпис тощо). Наприклад:

- TripleDesResult — запит із відкритим текстом та ключем.
- CaesarResult — результат шифрування/дешифрування з поясненням.
- VigenereResult — відповідь з результатом та проміжними значеннями.
- TextWithKeyRequest — базовий запит, який передає текст та ключ.

Такі об'єкти серіалізуються/десеріалізуються у формат JSON під час взаємодії клієнт-сервер.

Покращенням було б розділення моделей запиту/результату та пояснення, що дозволило б краще структурувати код та повторно використовувати об'єкти моделі між сервісами та контролерами.

3.2.5 Взаємодія компонентів

Взаємодію між компонентами веб-застосунку під час обробки запиту на шифрування даних представлено на рисунку 14.

Взаємодія між компонентами виглядає так:

1. Користувач вводить дані у форму на HTML-сторінці.
2. JavaScript генерує POST-запит до відповідного контролера.
3. Контролер викликає метод сервісу з об'єктом запиту.
4. Сервіс повертає об'єкт результату.
5. Контролер повертає відповідь у форматі JSON або HTML.
6. Представлення обробляє результат та відображає його користувачеві.

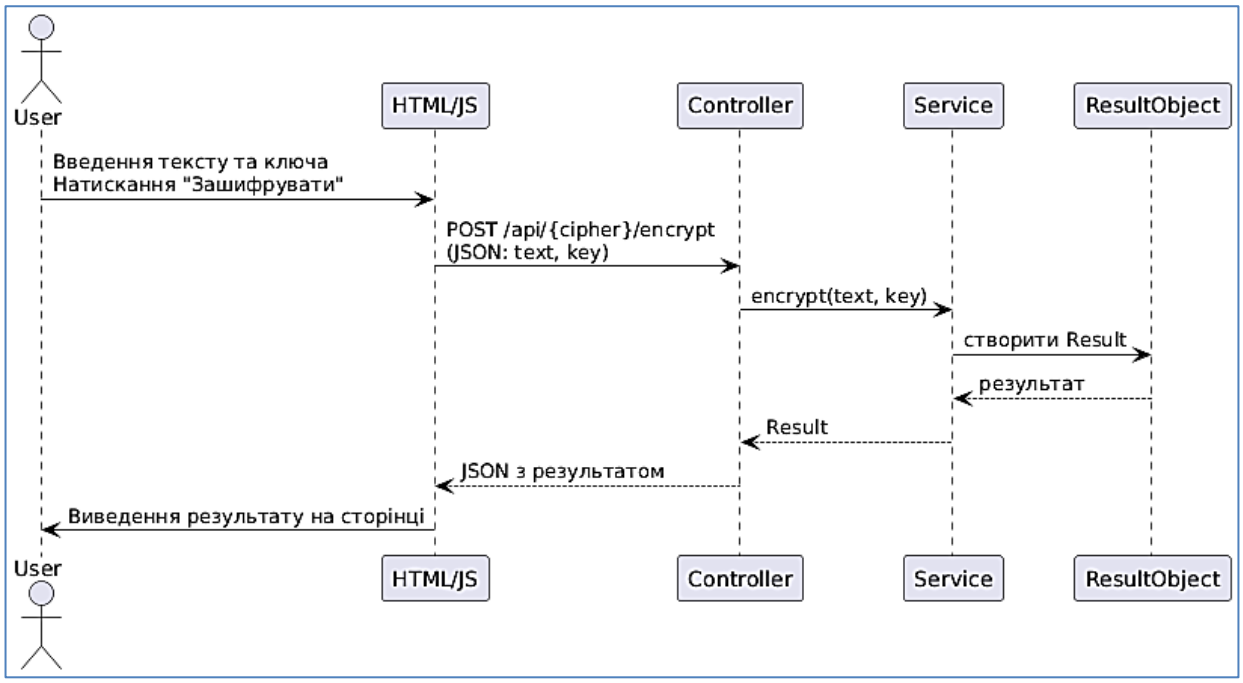


Рисунок *14 — UML-діаграма послідовностей обробки запиту шифрування*

3.2.6 Структура проєкту

Веб-застосунок “Crypto training app” має чітку організацію коду відповідно до багаторівневої архітектури, що дозволяє розділити логіку шифрування, обробку запитів та відображення клієнту.

1. Організація класів та контролерів у проєкті

Проєкт побудовано з поділом на пакети за принципом функціональної відповідальності (див. рис. 15).

Основні компоненти:

- Контролери — отримують HTTP-запити від клієнта та викликають відповідні сервіси.
- Сервіси — реалізують криптографічну логіку за допомогою JCA та Bouncy Castle.
- Класи результатів (Result) — передають вхідні параметри та результати між шарами.

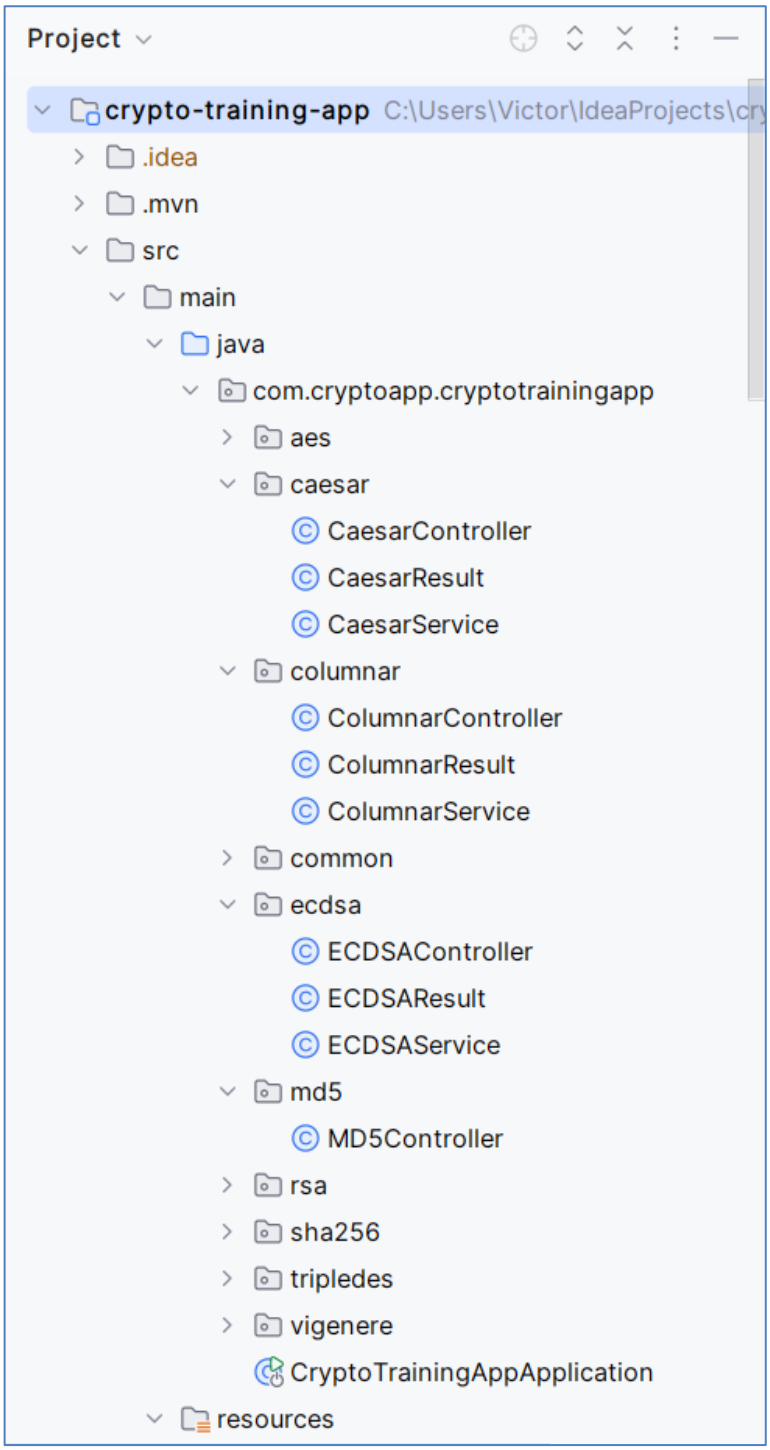


Рисунок *15 — Організація класів і контролерів у проєкті*

2. Структура клієнтських ресурсів (CSS та JavaScript)

Усі клієнтські ресурси (HTML, CSS, JS) розміщуються в статичному каталозі, який автоматично обслуговується Spring Boot (див. рис. 16).

Основні типи ресурсів:

— CSS: файли дизайну інтерфейсу, розділені за типами сторінок (історія,

практика, тестування, теорія).

— JavaScript: скрипти, що забезпечують взаємодію з контролерами, обробку форм та візуалізацію результатів.

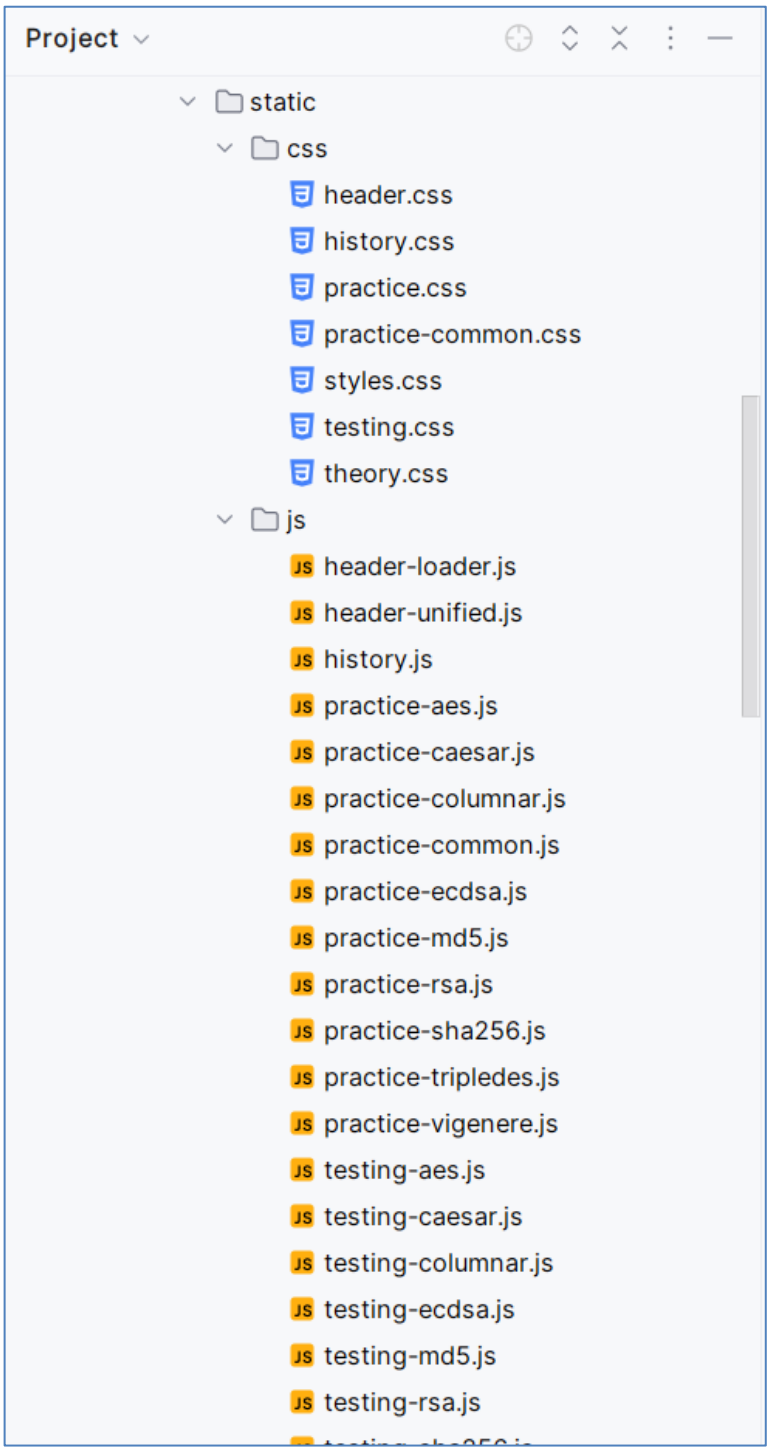


Рисунок *16 — Структура клієнтських ресурсів (CSS та JavaScript) веб-застосунку*

3. Тестова інфраструктура та налаштування проєкту

Для тестування функціональності застосунку використовуються модульні тести, організовані окремо для контролерів та сервісів (див. рис. 17).

Кожен криптоалгоритм має свій власний набір тестів, реалізований за допомогою JUnit 5 [24].

— Тестовий код знаходиться в src/test/java.

— Основні тести охоплюють контролери та сервіси для алгоритмів AES, Caesar, Columnar, RSA, MD5, SHA256, ECDSA, TripleDES, Vigenere.

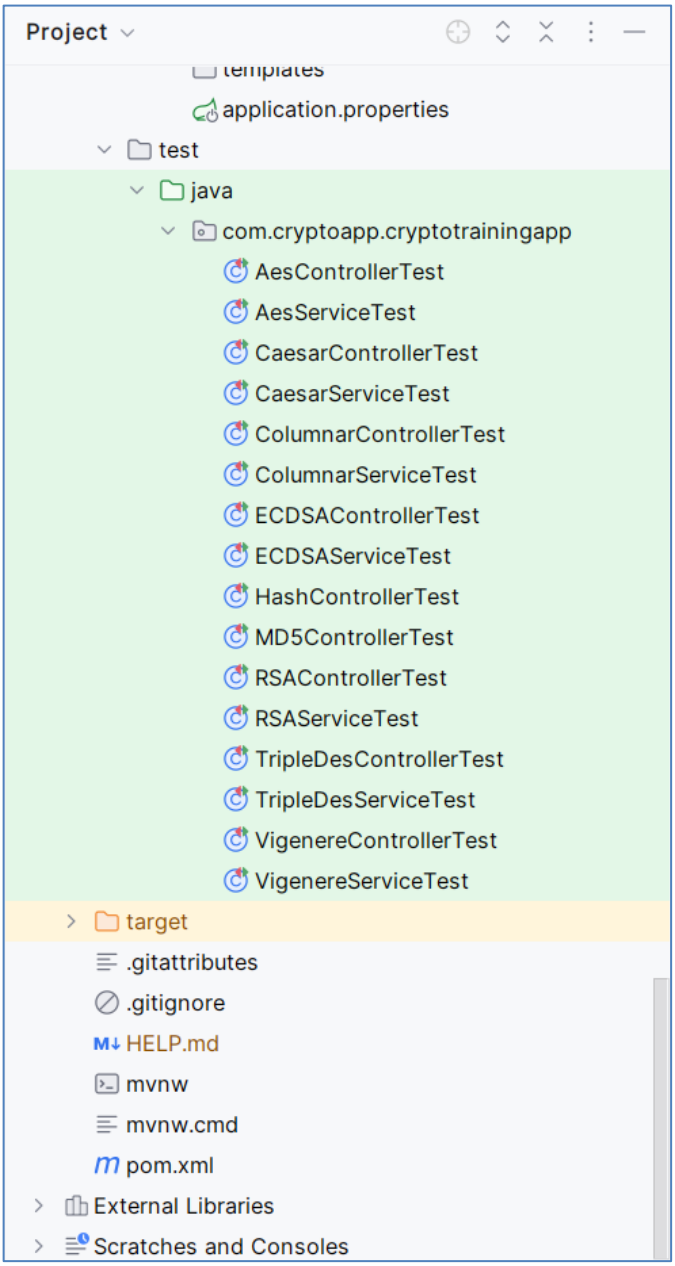


Рисунок *17 — Тестова інфраструктура та налаштування проєкту*

3.3 Проєктування інтерфейсу та функціональних можливостей застосунку

Графічний інтерфейс користувача (GUI) є ключовим елементом навчального застосунку “Crypto training app”, оскільки саме через нього відбувається основна взаємодія користувача з функціональністю. інтерфейс був розроблений з урахуванням принципів зручності використання, доступності, естетики та функціональності.

HTML-сторінки в проекті реалізовані як окремі статичні файли, що зберігаються в каталозі resources/static. Навігація між сторінками здійснюється шляхом завантаження відповідних HTML-файлів за посиланнями. Для динамічного оновлення вмісту сторінок (наприклад, відображення результатів шифрування або дешифрування) використовується JavaScript, який взаємодіє з сервером через HTTP-запити до REST-контролерів. Додаток не використовує шаблонізатори, такі як Thymeleaf, Vue або React, а натомість реалізує легкий клієнтський рендеринг за допомогою JavaScript.

Головна сторінка (рис. 18 та 19) є інформативною та містить розділ “Актуальність теми”, що обґрунтовує доцільність вивчення криптографії. Користувач також має доступ до перемикання теми оформлення – реалізовано світлу та темну теми, між якими можна перемикатися за допомогою інтерактивного перемикача у верхньому правому куті. Обраний стиль зберігається в локальному сховищі браузера.

Сторінка вибору алгоритму (рис. 20) структурована за блоками: класичні, симетричні, асиметричні шифри та алгоритми хешування. Кожна група представлена кнопками з назвами алгоритмів, які забезпечують перехід до відповідних теоретичних, практичних або тестових модулів для роботи з вибраним шифром. Це дозволяє швидко орієнтуватися в структурі програми.

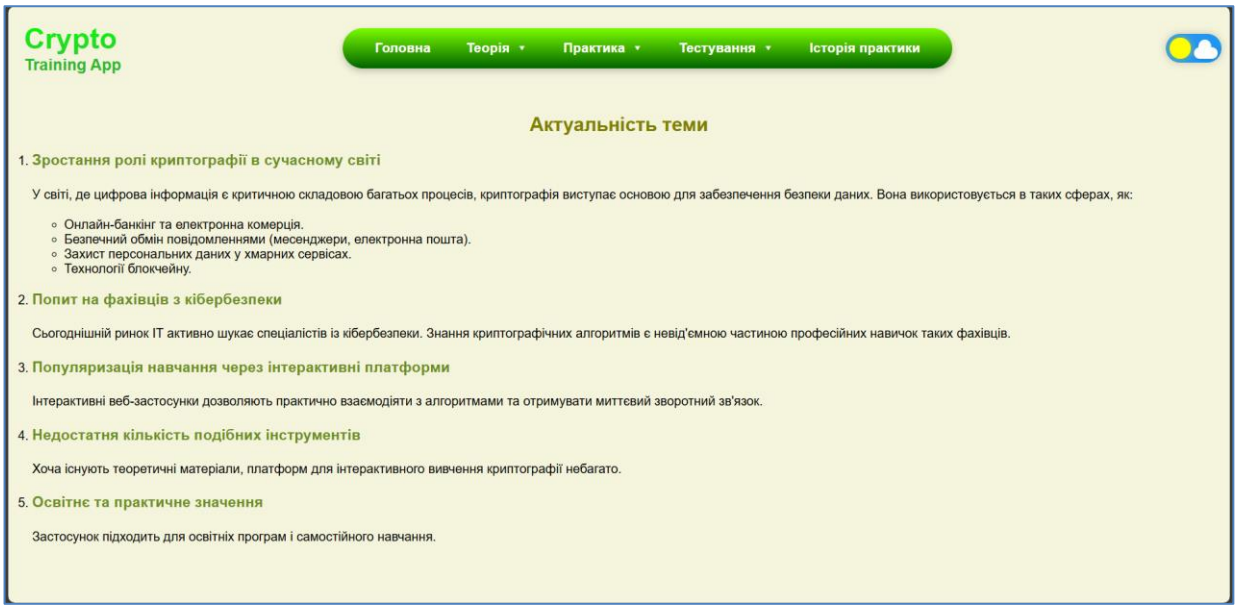


Рисунок *18 — Головна сторінка (світла тема)*

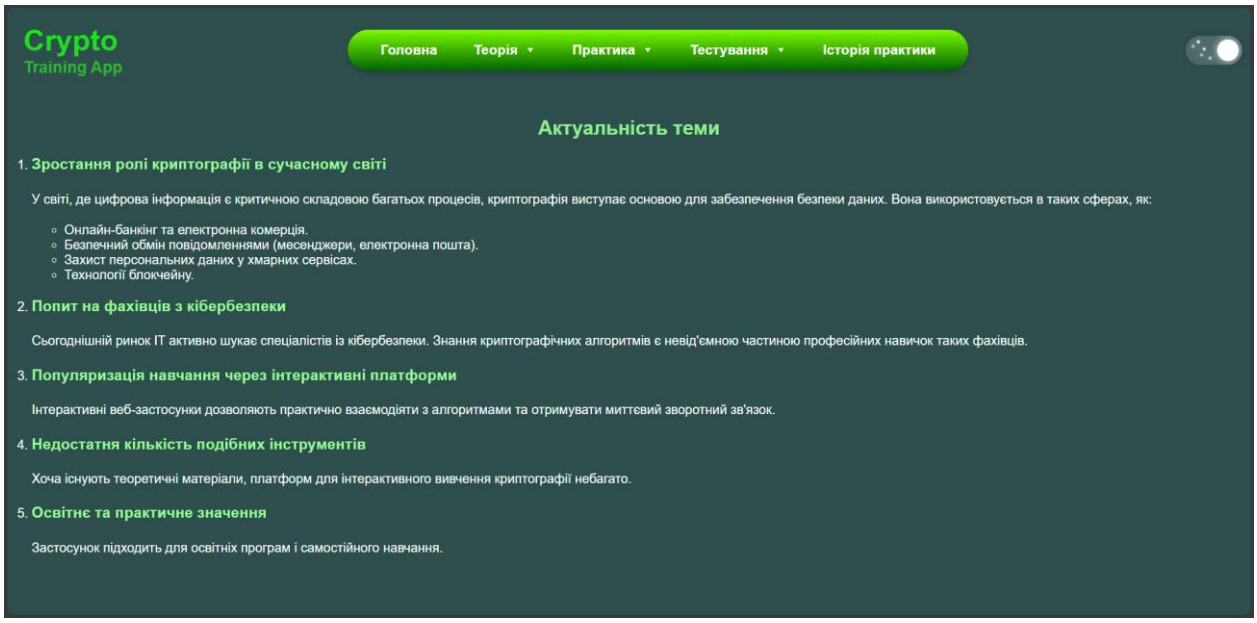


Рисунок *19 — Головна сторінка (темна тема)*

Теоретична сторінка (рис. 21 на прикладі шифру Цезаря) містить повний опис алгоритму, включаючи історичну довідку, принцип роботи, переваги та недоліки, математичні формули для шифрування та дешифрування, кроки виконання алгоритму, а також приклад реалізації на Java. Такий підхід надає користувачеві цілісне уявлення про алгоритм та готує до практичного застосування знань.



Рисунок *20 — Частина сторінки вибору криптографічних алгоритмів*



Рисунок *21 — Частина сторінки теорії шифру Цезаря*

Практичні сторінки (рис. 22 на прикладі шифру Цезаря) дозволяють користувачеві вводити вхідні дані (текст і ключ), вибирати дію (шифрування або розшифрування) та миттєво отримувати результат. Крім того, результат супроводжується детальним поясненням кроків виконання алгоритму, що сприяє глибшому розумінню принципу його роботи.

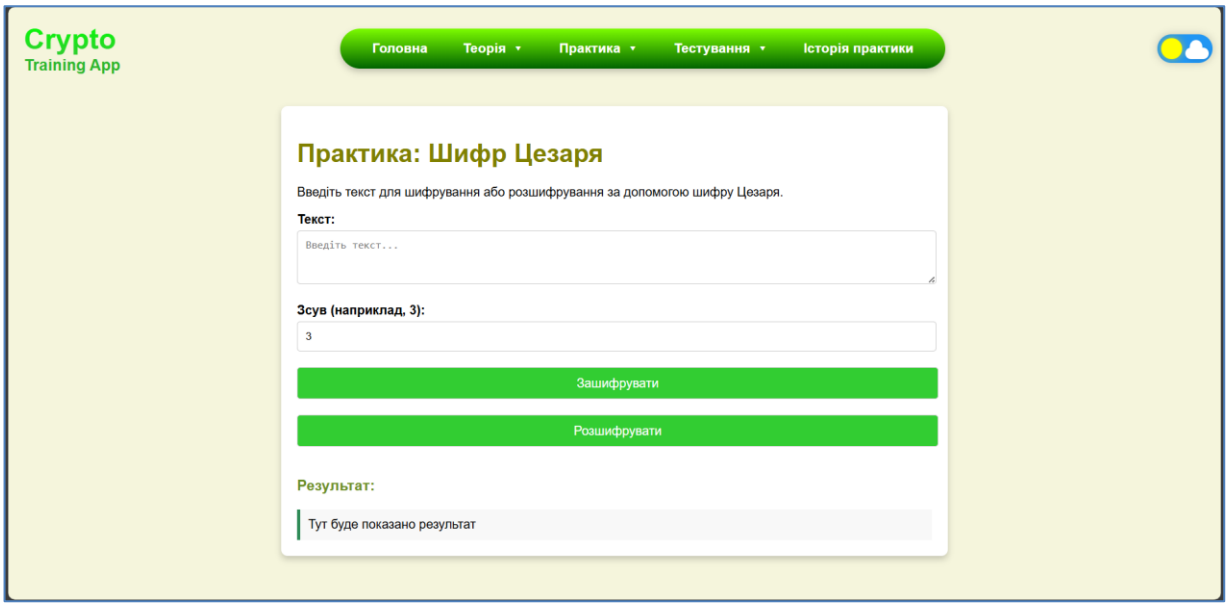


Рисунок *22 — Сторінка практики шифру Цезаря*

Тестова сторінка (рис. 23 на прикладі шифру Цезаря) містить набір із п'яти запитань із вибором однієї правильної відповіді, що охоплюють ключові аспекти алгоритму, включаючи його визначення, історію, слабкі сторони, кількість ключів та принцип роботи. Тест побудовано на теоретичному матеріалі відповідної сторінки, що дозволяє студенту не лише закріпити знання, а й перевірити своє розуміння безпосередньо у веб-інтерфейсі. інтерактивний блок дозволяє вибрати відповіді та отримати оцінку результату одразу після натискання кнопки “Перевірити”.

Веб-застосунок також забезпечує функціональність збереження історії виконаних операцій (рис. 24). Усі результати практичних вправ, включаючи вхідні дані, ключі, отриманий результат та детальне пояснення обчислень, зберігаються в локальному сховищі браузера. Кожен запис відображається з датою, часом, назвою алгоритму, вхідними параметрами, результатом та кроками обробки. Користувач може переглянути всі записи, видалити окремі результати або очистити всю історію. Це не тільки дозволяє зручно відстежувати виконані дії, але й сприяє аналізу помилок, глибшому розумінню роботи алгоритмів та контролю власного прогресу.

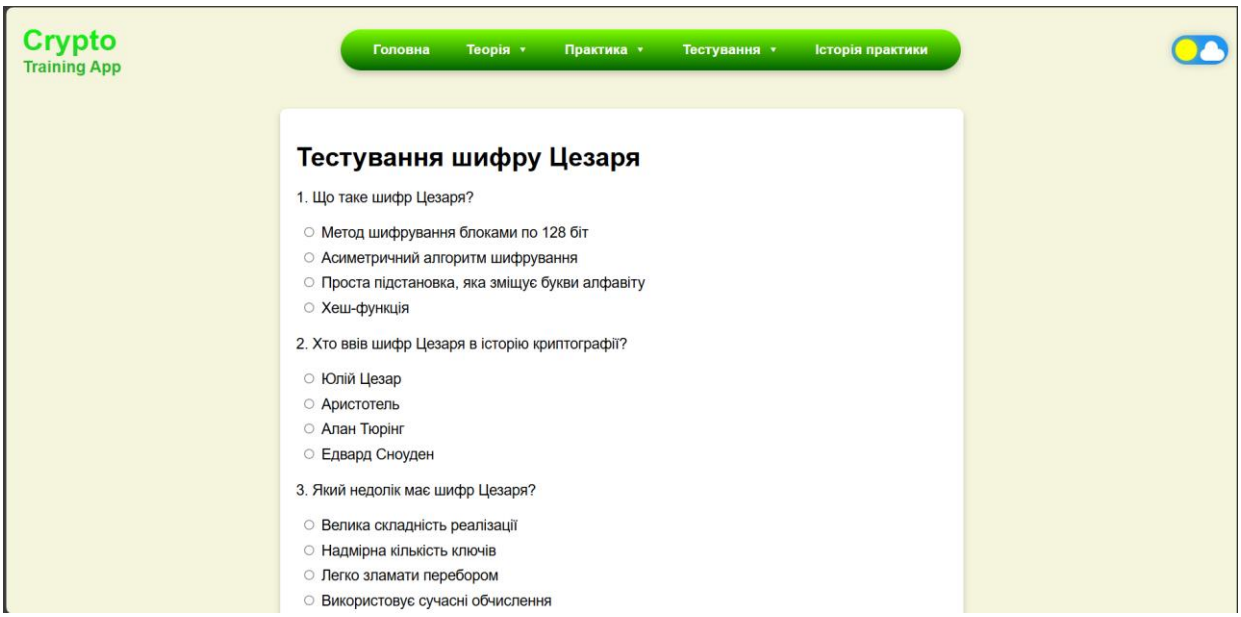


Рисунок *23 — Частина сторінки тестування шифру Цезаря*

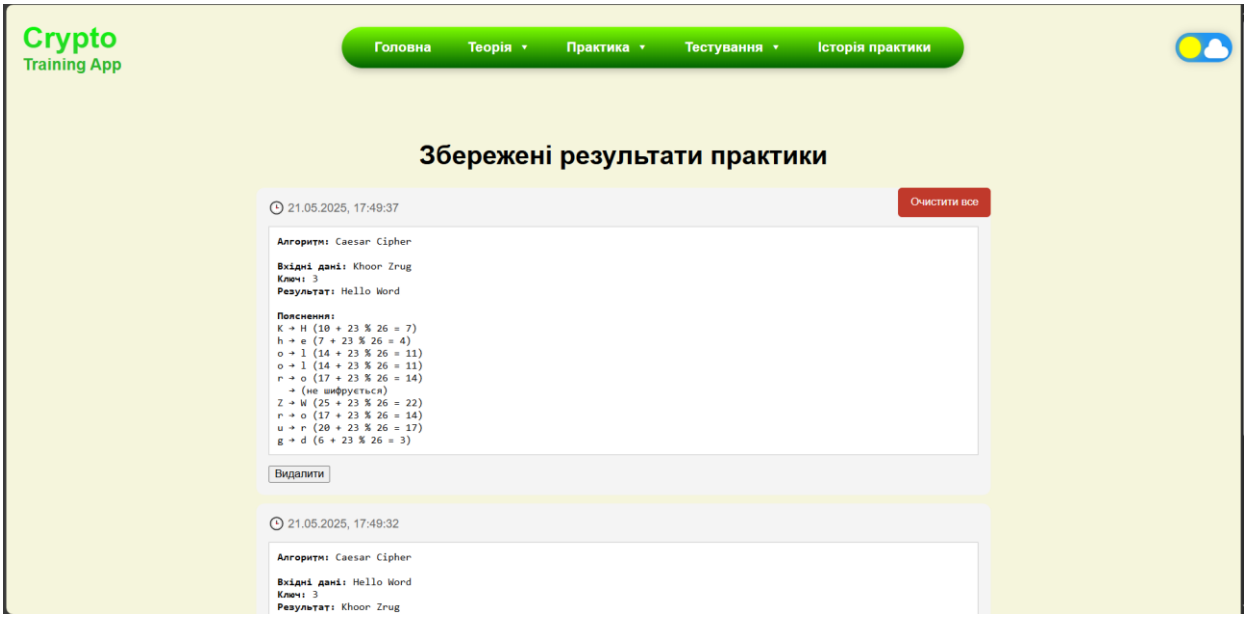


Рисунок *24 — Частина сторінки історії практики*

Таким чином, інтерфейс застосунку побудовано за принципами простоти, зручності та інтерактивності. Завдяки підтримці двох тем, зрозумілій навігації та збереженню історії, застосунок забезпечує повноцінний навчальний процес, що включає як теоретичне опанування, так і практичне освоєння криптографічних алгоритмів.

3.4 Реалізація обробки криптографічних алгоритмів на прикладі Spring Boot

Реалізація логіки обробки криптографічного алгоритму у веб-застосунку здійснюється за допомогою фреймворку Spring Boot, який забезпечує ефективну організацію контролерів, сервісів та передачі даних через REST API.

Кожен криптографічний алгоритм представлений окремим контролером та сервісом, що реалізують логіку шифрування та дешифрування. Розглянемо приклад реалізації одного з найпростіших класичних алгоритмів – шифру Цезаря.

Клас CaesarService відповідає за виконання криптографічних операцій. Метод encryptDetailed приймає текст та числовий ключ-зсув, шифрує кожен символ (як великі, так і малі літери), генеруючи пояснення для кожного кроку (Лістинг 2).

Лістинг *2 Сервіс шифрування та розшифрування методом Цезаря у Spring*

```
@Service
public class CaesarService {
public CaesarResult encryptDetailed(String text, int shift)
{
StringBuilder result = new StringBuilder();
StringBuilder explanation = new StringBuilder();
    for (char ch : text.toCharArray()) {
        if (Character.isUpperCase(ch)) {
            int originalIndex = ch - 'A';
            int shiftedIndex = (originalIndex + shift) %
26;
            char encrypted = (char) ('A' + shiftedIndex);
            explanation.append(ch)
                    .append(" → ").append(encrypted)
                    .append(" (").append(originalIndex)
```

```
                    .append(" + ").append(shift)
                    .append(" % 26 =
").append(shiftedIndex)
                    .append(")").append("\n");
            result.append(encrypted);
        } else if (Character.isLowerCase(ch)) {
            int originalIndex = ch - 'a';
            int shiftedIndex = (originalIndex + shift) %
26;
            char encrypted = (char) ('a' + shiftedIndex);
            explanation.append(ch)
                    .append(" → ").append(encrypted)
                    .append(" (").append(originalIndex)
                    .append(" + ").append(shift)
                    .append(" % 26 =
").append(shiftedIndex)
                    .append(")").append("\n");
            result.append(encrypted);
        } else {
            explanation.append(ch).append(" → (не
шифрується)").append("\n");
            result.append(ch);
        }
    }

    return new CaesarResult(text, shift, result.toString(),
explanation.toString());
}

public CaesarResult decryptDetailed(String text, int shift)
{
    int adjustedShift = 26 - (shift % 26);
    return encryptDetailed(text, adjustedShift);
```

```
}
}
```

Контролер CaesarController приймає POST-запити від клієнта, передає параметри сервісу CaesarService та повертає результат як об'єкт CaesarResult (Лістинг 3).

Лістинг *3 Контролер CaesarController для обробки HTTP-запитів шифру Цезаря у Spring Boot*

```
@RestController
@RequestMapping("/api/caesar")
public class CaesarController {
private final CaesarService caesarService;

@Autowired
public CaesarController(CaesarService caesarService) {
    this.caesarService = caesarService;
}

@PostMapping("/encrypt")
public CaesarResult encrypt(@RequestBody Map<String,
String> request) {
    String text = request.get("text");
    int shift = Integer.parseInt(request.get("shift"));
    return caesarService.encryptDetailed(text, shift);
}

@PostMapping("/decrypt")
public CaesarResult decrypt(@RequestBody Map<String,
String> request) {
    String text = request.get("text");
    int shift = Integer.parseInt(request.get("shift"));
    return caesarService.decryptDetailed(text, shift);
```

```
}
}
```

Відповідь на запит формується як об'єкт CaesarResult, який містить вхідний текст, ключ зсуву, результат операції та пояснення (Лістинг 4).

Лістинг *4 DTO-клас CaesarResult для передачі результатів шифрування*

```
public class CaesarResult {
private String inputText;
private int shift;
private String resultText;
private String explanation;
public CaesarResult(String inputText, int shift, String
resultText, String explanation) {
    this.inputText = inputText;
    this.shift = shift;
    this.resultText = resultText;
    this.explanation = explanation;
}

// Геттери
}
```

На стороні клієнта HTML-сторінка (practice-caesar.html) містить поля для введення тексту та зміни ключа, кнопки шифрування/дешифрування, а також блок для відображення результату та пояснення.

За допомогою JavaScript запити надсилаються до відповідної кінцевої точки API (Лістинг 5).

Лістинг *5 Використання JavaScript для виклику API шифрування шифром Цезаря*

```
fetch("/api/caesar/encrypt", {
method: "POST",
headers: { "Content-Type": "application/json" },
```

```
body: JSON.stringify({ text, shift })
});
```

Результат відображається на екрані, а також зберігається в localStorage для подальшого перегляду в розділі “Історія практики”.

Цей підхід демонструє ефективне використання Spring Boot для створення масштабованої структури веб-застосунку з чітким розділенням логіки, зручною інтеграцією REST API, підтримкою багатьох алгоритмів та можливістю візуалізації проміжних кроків, що особливо важливо для освітнього контексту.

3.5 Організація збереження історії операцій у браузері

Однією з важливих функцій веб-застосунку “Crypto training app” є збереження історії виконаних криптографічних операцій безпосередньо в браузері користувача. Це робиться за допомогою технології Web Storage API – зокрема, localStorage, яка дозволяє зберігати дані між сеансами без використання бази даних сервера.

Після кожної успішної операції шифрування або дешифрування функція saveResult() створює об'єкт результату, який містить такі поля:

- algorithm — назва використаного алгоритму (наприклад, “Caesar Cipher”);
- input — вихідний текст, введений користувачем;
- key — значення ключа або зміщення;
- result — результат шифрування або дешифрування;
- explanation — текстове пояснення з обчисленнями;
- timestamp — позначка часу у форматі дати та часу виконання операції.

Цей об'єкт додається до масиву попередніх записів, який зберігається в localStorage як рядок JSON з ключем "cryptoPracticeResults" (Лістинг 6).

Лістинг *6 Збереження результатів шифрування в localStorage браузера*

```
function saveResult(algorithm, input, key, result,
```

```
explanation) {
const previous =
JSON.parse(localStorage.getItem("cryptoPracticeResults"))
|| [];
const entry = {
algorithm,
input,
key,
result,
explanation,
timestamp: new Date().toLocaleString()
};
previous.push(entry);
localStorage.setItem("cryptoPracticeResults",
JSON.stringify(previous));
}
```

Для відображення історії на відповідній сторінці (history.html) використовується функція loadResults(), яка зчитує записи з localStorage, сортує їх у зворотному порядку (від останнього до першого) та формує HTML-блоки для виводу (Лістинг 7).

Лістинг *7 Відображення історії результатів з localStorage у HTML-розмітці*

```
function loadResults() {
const container = document.getElementById("results-
container");
const entries =
JSON.parse(localStorage.getItem("cryptoPracticeResults"))
|| [];
container.innerHTML = ""; // очищення перед новим
завантаженням

if (entries.length === 0) {
```

```
    container.innerHTML = "<p>Немає збережених
результатів.</p>";
    return;
}

entries.reverse().forEach((entry, index) => {
    const actualIndex = entries.length - 1 - index;
    const div = document.createElement("div");
    div.className = "entry";
    div.innerHTML = `
    <div class="timestamp">🕒 ${entry.timestamp}</div>
    <pre><strong>Алгоритм:</strong>
${entry.algorithm}\n\n<strong>Вхідні дані:</strong>
${entry.input}\n<strong>Ключ:</strong>
${entry.key}\n<strong>Результат:</strong>
${entry.result}\n\n<strong>Пояснення:</strong>\n${entry.exp
lanation}</pre>
    <button
onclick="deleteResult(${actualIndex})">Видалити</button>
`;
    container.appendChild(div);
});
}
```

Користувач може:

— переглядати збережені результати з детальними поясненнями;

— видаляти окремі записи за допомогою функції deleteResult(index) (Лістинг 8);

— повністю очищати історію за допомогою функції clearResults(), яка додатково підтверджує дію (Лістинг 9).

Лістинг *8 Видалення окремого запису результату з localStorage*

```
function deleteResult(index) {
```

```
const entries =
JSON.parse(localStorage.getItem("cryptoPracticeResults"))
|| [];
if (index >= 0 && index < entries.length) {
entries.splice(index, 1);
localStorage.setItem("cryptoPracticeResults",
JSON.stringify(entries));
loadResults();
}
}
```

Лістинг *9 Очищення всіх збережених результатів із localStorage*

```
function clearResults() {
if (confirm("Ви впевнені, що хочете видалити всі збережені
результати?")) {
localStorage.removeItem("cryptoPracticeResults");
loadResults();
}
}

loadResults();
```

Такий підхід не лише забезпечує швидке та локальне зберігання, але й повністю виключає передачу даних на сервер, що підвищує конфіденційність. Історія може бути використана студентом для аналізу помилок, відстеження прогресу та повторного використання попередніх даних без необхідності повторного введення. Система зручна, інтуїтивно зрозуміла та підходить для навчального середовища.

3.6 Тестування функціональності застосунку

Під час розробки веб-застосунку для навчання криптографічним

алгоритмам було проведено два основних типи тестування:

— Ручне тестування, яке дозволяє перевірити поведінку інтерфейсу та інтеграцію модулів.

— Автоматизоване тестування (модульне тестування), зосереджене на тестуванні окремих методів, сервісів та контролерів за допомогою бібліотек JUnit, Spring Boot Test та MockMvc.

Метою тестування була перевірка правильності реалізації шифрування, дешифрування, генерації ключів, валідації введених даних, а також правильності роботи відповідей сервера.

Приклади модульних тестів

1. Тестування шифрування та дешифрування AES (Лістинг 10)

Файл: AesControllerTest.java

Лістинг *10 Юніт-тест для перевірки шифрування та розшифрування AES через REST API*

```
@Test
    public void testEncryptDecrypt() throws Exception {
        //  1. Encrypt
        Map<String, String> encryptRequest = new
HashMap<>();
        encryptRequest.put("key", KEY);
        encryptRequest.put("iv", IV);
        encryptRequest.put("plaintext", PLAINTEXT);

        String encryptJson =
objectMapper.writeValueAsString(encryptRequest);

        String ciphertext =
mockMvc.perform(post("/api/aes/encrypt")

.contentType(MediaType.APPLICATION_JSON)
                        .content(encryptJson))
```

```
                .andExpect(status().isOk())

.andExpect(jsonPath("$.ciphertext").exists())
                .andReturn()
                .getResponse()
                .getContentAsString();

        Map<String, String> encryptResponse =
objectMapper.readValue(ciphertext, Map.class);
        String cipherHex =
encryptResponse.get("ciphertext");

        assertThat(cipherHex).isNotEmpty();

        //  2. Decrypt
        Map<String, String> decryptRequest = new
HashMap<>();
        decryptRequest.put("key", KEY);
        decryptRequest.put("iv", IV);
        decryptRequest.put("ciphertext", cipherHex);

        String decryptJson =
objectMapper.writeValueAsString(decryptRequest);

        String decrypted =
mockMvc.perform(post("/api/aes/decrypt")

.contentType(MediaType.APPLICATION_JSON)
                        .content(decryptJson))
                .andExpect(status().isOk())

.andExpect(jsonPath("$.plaintext").value(PLAINTEXT))
                .andReturn()
```

```
                .getResponse()
                .getContentAsString();

        Map<String, String> decryptResponse =
objectMapper.readValue(decrypted, Map.class);

assertThat(decryptResponse.get("plaintext")).isEqualTo(PLAI
NTEXT);
}
```

Опис тесту: testEncryptDecrypt() з AesControllerTest

Цей модульний тест перевіряє правильність роботи API шифрування та дешифрування, реалізованого в контролері AES (/api/aes/encrypt та /api/aes/decrypt). Він імітує повний цикл шифрування та дешифрування, імітуючи HTTP-запити через Spring MockMvc.

Логіка тестування:

1. Підготовка вхідних даних:
— Текст для шифрування: “Hello AES!”.
— Ключ: “abcdefghijklmnop” (16 символів - допустима довжина для AES-128).
— IV: “1234567890abcdef” (16 символів - початковий вектор для AES-CBC).
2. Надсилання POST-запиту до /api/aes/encrypt:
— Дані передаються у форматі JSON.
— Очікується, що відповідь матиме поле шифротексту, яке міститиме результат шифрування.
3. Перевірка, що зашифрований текст не є порожнім.
4. Далі отриманий шифротекст розшифровується:
— POST-запит до /api/aes/decrypt з тим самим ключем та IV.
— Очікується, що відповідь міститиме поле відкритого тексту зі значенням, що дорівнює вхідному тексту “Hello AES!”.

Очікуваний результат:

— Запит на шифрування має повернути правильно зашифрований текст у форматі Base64 або шістнадцятковому форматі.
— Запит на розшифрування має повернути оригінальний текст “Hello AES!”.
— Усі відповіді HTTP мають статус 200 OK.
— У відповіді на запит на розшифрування ключ відкритого тексту має точно відповідати PLAINTEXT.

Таким чином, тест підтверджує:

— Правильність алгоритму AES-CBC в обох напрямках.
— Правильність обробки вхідних параметрів (ключ, iv, відкритий/зашифрований текст).
— Безперервність циклу «шифрування → дешифрування» без втрати даних.

Результат тесту AesControllerTest представленно на рисунку 25.

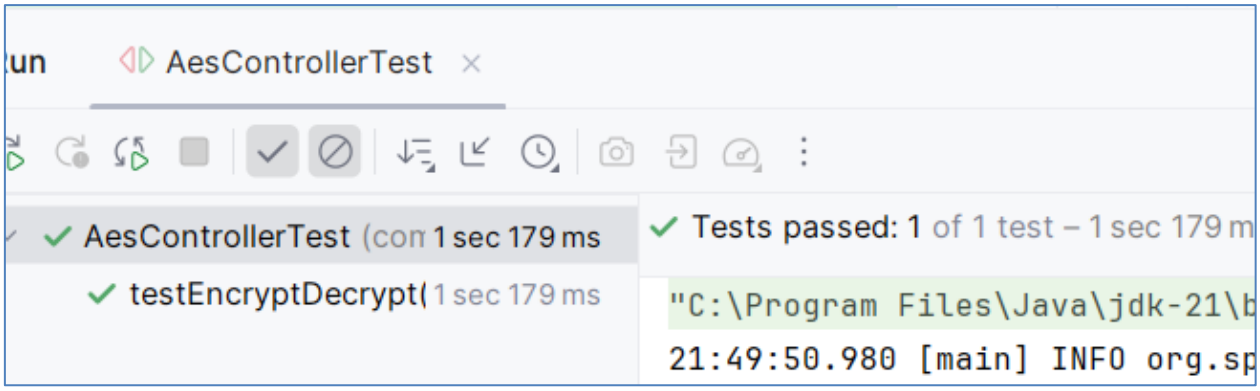


Рисунок *25 — Результат виконання тесту testEncryptDecrypt*

2. Тестування обробки некоректного зсуву шифру Цезаря (Лістинг 11)

Файл: CaesarServiceTest.java

Лістинг *11 Юніт-тест для обробки помилки при неправильному зсуві у шифрі Цезаря (Caesar)*

```
@Test
    void testInvalidShift() throws Exception {
        mockMvc.perform(post("/api/caesar/encrypt")

.contentType(MediaType.APPLICATION_JSON)
```

```
.content("{\"text\":\"abc\",\"shift\":\"notANumber\"}"))
                .andExpect(status().isBadRequest())
                .andExpect(content().string("Невірний
формат числа: For input string: \"notANumber\""));

}
```

Опис тесту: testInvalidShift() з класу CaesarControllerTest

Цей тест перевіряє стійкість контролера /api/caesar/encrypt до недійсних вхідних даних, зокрема, ситуації, коли значення параметра “shift” не є числом.

Логіка тесту:

1. POST-запит надсилається до кінцевої точки /api/caesar/encrypt з тілом JSON:

— текст: “abc” (дійсний текст);

— shift: “notANumber” (недійсне значення, яке не може бути перетворене на число).

2. Очікувана поведінка:

— Статус відповіді HTTP має бути 400 Bad Request.

— Тіло відповіді має містити повідомлення про помилку: “Invalid number format: For input string: “notANumber””.

Очікуваний результат:

— Запит обробляється на рівні обробки помилок - він не досягає шифрування.

— Контролер повертає повідомлення про недійсне значення параметра “shift”.

— Підтверджено, що сервіс може працювати з недійсними даними без винятків, які можуть спричинити збій.

Результат тесту CaesarControllerTest представленно на рисунку 26.

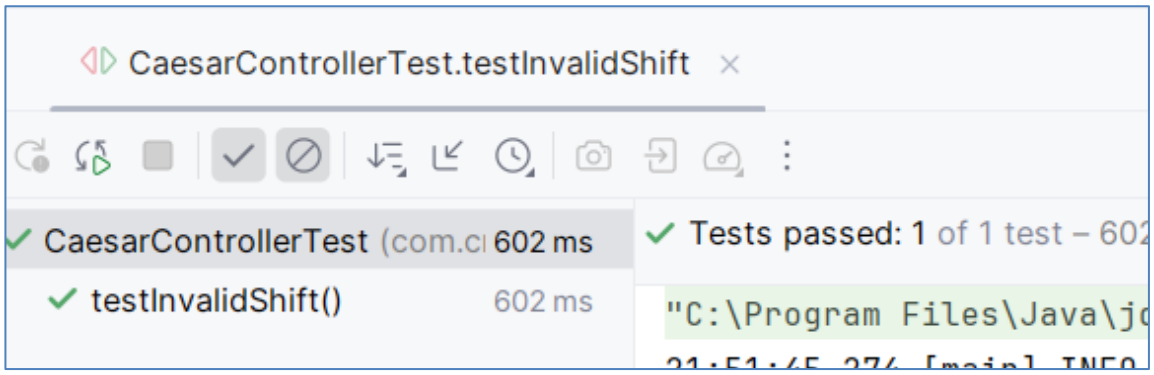


Рисунок *26 — Результат виконання тесту testInvalidShift*

3. Тестування шифрування та розшифрування Triple DES на рівні контролера (Лістинг 12)

Лістинг *12 Інтеграційний тест для шифрування/дешифрування за алгоритмом TripleDES через HTML API*

```
@Test
    public void testEncryptDecrypt() throws Exception {
        // ♡ Шифрування
        TripleDesRequest encryptRequest = new
TripleDesRequest();
        encryptRequest.setText(plaintext);
        encryptRequest.setKey(key);

        String encryptResponse =
mockMvc.perform(post("/api/tripledes/encrypt")

.contentType(MediaType.APPLICATION_JSON)

.content(objectMapper.writeValueAsString(encryptRequest)))
                .andExpect(status().isOk())

.andExpect(content().contentTypeCompatibleWith(MediaType.TE
XT_HTML))
                .andReturn()
```

```
                .getResponse()
                .getContentAsString();

        // Витягти зашифрований текст з <code>...</code>
        Document htmlDoc = Jsoup.parse(encryptResponse);
        Elements codeTags = htmlDoc.select("code");
        assertThat(codeTags).isNotEmpty();
        String encryptedText = codeTags.last().text();
        assertThat(encryptedText).isNotBlank();

        // Розшифрування
        TripleDesRequest decryptRequest = new
TripleDesRequest();
        decryptRequest.setText(encryptedText);
        decryptRequest.setKey(key);

        String decryptResponse =
mockMvc.perform(post("/api/tripledes/decrypt")

.contentType(MediaType.APPLICATION_JSON)

.content(objectMapper.writeValueAsString(decryptRequest)))
                .andExpect(status().isOk())
                .andReturn()
                .getResponse()
                .getContentAsString();

        // Перевірити, що розшифрований текст збігається з
початковим
        Document decryptedDoc =
Jsoup.parse(decryptResponse);
        String decryptedText =
decryptedDoc.select("code").last().text();
```

```
        assertThat(decryptedText).isEqualTo(plaintext);
    }
```

Цей модульний тест перевіряє повний цикл шифрування та дешифрування тексту за допомогою алгоритму Triple DES у контролері TripleDesController. Тест реалізовано за допомогою Spring Boot Test з використанням реального HTTP-запиту до API.

Логіка тесту:

— Вхідний текст: “Test TripleDES!”.

— Ключ: “123456789012345678901234” (24 символи - відповідно до вимог алгоритму).

Кроки тестування:

1. POST-запит надсилається до /api/tripledes/encrypt з текстом та ключем.
2. Перевіряється, що відповідь має статус 200 OK та тип text/html.
3. Зашифрований текст витягується з відповіді з HTML-розмітки (елемент <code>).
4. Отриманий шифр знову надсилається до /api/tripledes/decrypt разом з тим самим ключем.
5. Перевіряється, що результат розшифрування є оригінальним текстом “Test TripleDES!”.

Очікуваний результат:

— Зашифрований текст успішно витягнуто з HTML-відповіді.

— Результат розшифрування точно відповідає оригінальному тексту.

— Усі перевірки assertThat(...) проходять без винятків.

Результат тесту TripleDesControllerTest представленно на рисунку 27.

Опис загального покриття коду

Покриття коду проєкту було виміряно за допомогою інструменту JaCoCo, який був інтегрований у процес збірки Maven [21]. На основі результатів запуску тестів та створення звітів про покриття було отримано такі ключові

показники, представлені в Таблиці 1.

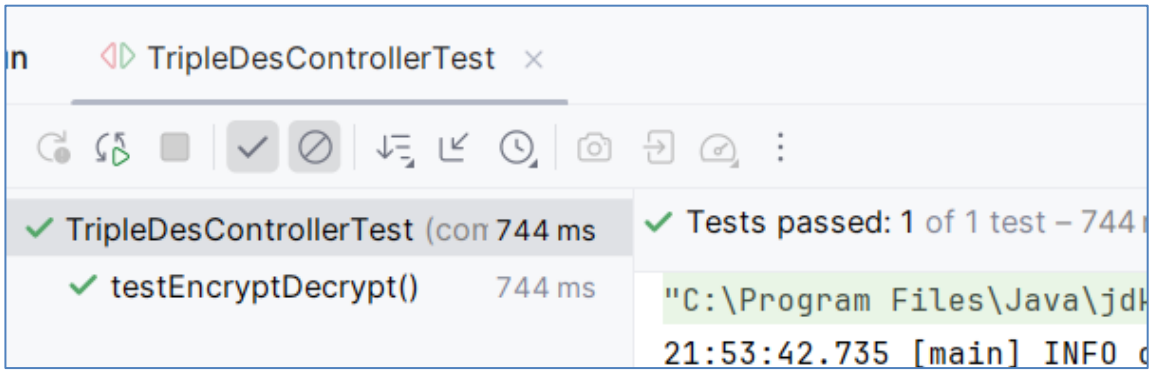


Рисунок *27 — Результат виконання тесту testEncryptDecrypt*

Таблиця *1 — Основні метрики покриття коду у проєкті*

Метрика	Значення
Покриття інструкцій (Instructions)	92 % (2 114 з 2 277)
Покриття умов/гілок (Branches)	95 % (65 з 68)
Покриття методів	71 % (113 з 159)
Покриття класів	96 % (27 з 28)
Покриття рядків коду (Lines)	88 % (462 з 525)
Цикломатична складність (Cxty)	75 % (144 з 193)

Ці показники вказують на загалом добре покриття основних структур коду, особливо гілок (if/else), класів та операторів. Водночас, відносно нижчі значення для методів та складності вказують на наявність класів з мертвим кодом або відсутніми тестами.

У ході аналізу було також сформовано детальну статистику покриття для окремих класів:

1. Класи з повним (100%) покриттям:

— Сервіси: ColumnarService, TripleDesService, ECDSAService, CaesarService.

— Контролери: RSAController, ECDSAController, HashController, AesController, CaesarController, VigenereController, ColumnarController.

— DTO та допоміжні об'єкти: RSAResultExplanation,

ECDSAResultExplanation, TextWithKeyRequest, TripleDesRequest, VigenereResponse, CaesarResult.

2. Класи з частковим покриттям (60–95%) представленні в Таблиці 2.

Таблиця *2 — Частково покриті класи*

Назва класу	Покриття
VigenereService	97 %
RSAService	93 %
MD5Controller	93 %
ColumnarResponse	62 %
CryptoTrainingAppApplication	64 %
ExceptionHandlerController	50 %

3. Класи з низьким або відсутнім покриттям представленні в Таблиці 3.

Таблиця *3— Класи з мінімальним або нульовим покриттям*

Назва класу	Покриття
CryptoController	30 %
PageController	4 %
TextResponseWithExplanation	0 %

Результати покриття коду, отримані за допомогою JaCoCo, представлені на рисунках 28 та 29: рисунок 28 демонструє покриття загального пакету crypto-training-app, а рисунок 29 — покриття підпакету com.cryptoapp.cryptotrainingapp.

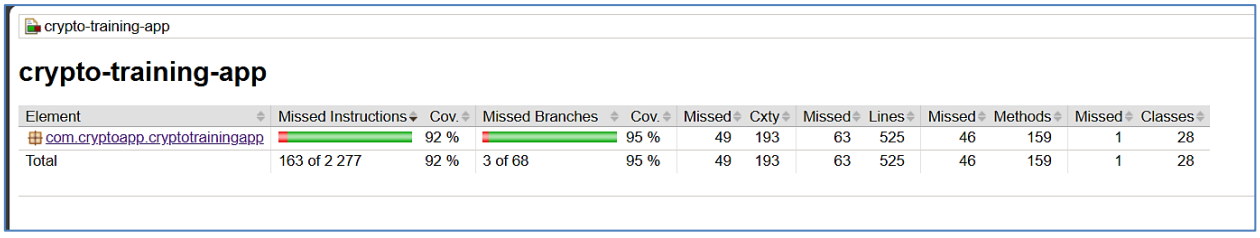


Рисунок *28 — Загальний пакет: crypto-training-app*

crypto-training-app > com.cryptoapp.cryptotrainingapp

com.cryptoapp.cryptotrainingapp

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
PageController		4 %		n/a	32	33	32	33	32	33	0	1
ColumnarResponse		62 %		n/a	4	9	8	18	4	9	0	1
TextResponseWithExplanation		0 %		n/a	3	3	6	6	3	3	1	1
TripleDesController		66 %		n/a	0	3	4	11	0	3	0	1
ExceptionHandlerController		50 %		n/a	2	5	3	6	2	5	0	1
RSAService		93 %		100 %	2	9	2	49	2	7	0	1
MD5Controller		93 %		90 %	1	7	2	25	0	2	0	1
CryptoController		30 %		n/a	1	2	2	3	1	2	0	1
VigenereService		97 %		87 %	1	7	1	39	0	3	0	1
CryptoTrainingAppApplication		64 %		n/a	1	3	2	5	1	3	0	1
AesUtil		95 %		n/a	1	3	1	14	1	3	0	1
ColumnarService		100 %		96 %	1	22	0	51	0	7	0	1
TripleDesService		100 %		100 %	0	6	0	48	0	5	0	1
ECDSAService		100 %		100 %	0	8	0	39	0	7	0	1
CaesarService		100 %		100 %	0	6	0	32	0	3	0	1
HashController		100 %		100 %	0	5	0	20	0	2	0	1
RSAController		100 %		n/a	0	4	0	20	0	4	0	1
ECDSAController		100 %		n/a	0	4	0	13	0	4	0	1
RSAResultExplanation		100 %		n/a	0	15	0	22	0	15	0	1
AesController		100 %		n/a	0	3	0	11	0	3	0	1
CaesarController		100 %		n/a	0	3	0	9	0	3	0	1
ECDSAResultExplanation		100 %		n/a	0	9	0	13	0	9	0	1
VigenereController		100 %		n/a	0	3	0	3	0	3	0	1
CaesarResult		100 %		n/a	0	5	0	10	0	5	0	1
ColumnarController		100 %		n/a	0	3	0	5	0	3	0	1
TextWithKeyRequest		100 %		n/a	0	5	0	7	0	5	0	1
TripleDesRequest		100 %		n/a	0	5	0	7	0	5	0	1
VigenereResponse		100 %		n/a	0	3	0	6	0	3	0	1
Total	163 of 2 277	92 %	3 of 68	95 %	49	193	63	525	46	159	1	28

Рисунок *29 — Підпакет com.cryptoapp.cryptotrainingapp*

Проєкт демонструє високий рівень охоплення тестуванням бізнес-логіки, криптографічних сервісів та контролерів. Особлива увага приділяється модульним тестам для класів сервісів шифрування та обробки запитів. Основними недоліками є низьке охоплення допоміжних контролерів, які безпосередньо не впливають на шифрування, але все ж залишаються частиною структури проєкту.

3.7 Розробка рекомендацій щодо використання ПЗ у навчальному процесі

Розроблене програмне забезпечення "Crypto Training App" призначене для навчання основам сучасної та класичної криптографії. Виходячи з функціональних можливостей системи, результатів її тестування, а також вимог до навчальних засобів захисту інформації, сформульовано наступні рекомендації щодо використання веб-застосунку в навчальному процесі:

Формат використання:

- Програмне забезпечення доцільно використовувати під час лабораторних та практичних занять з дисциплін “інформаційна безпека”, “Криптографічні методи захисту інформації”, “Безпека веб-застосунків”.
- Застосунок може виступати як демонстраційний інструмент для викладача, а також для самостійного виконання завдань студентами.

Можливості використання:

- Модульність системи дозволяє користувачеві поступово ознайомитися з різними класами криптографічних алгоритмів (класичні, симетричні, асиметричні, хеш-функції).
- Візуалізація кроків шифрування/дешифрування забезпечує глибше розуміння внутрішньої логіки роботи алгоритмів.
- інтерактивний формат дозволяє експериментувати з різними параметрами (ключі, тексти) та одразу бачити результат.

Педагогічні переваги:

- Висока наочність матеріалу сприяє кращому засвоєнню абстрактних понять криптографії.
- Можливість автоматичної перевірки результатів (під час тестування) дозволяє учням контролювати власний прогрес.

Методичні рекомендації:

- Вчителям рекомендується готувати сценарії уроків на основі використання конкретних модулів (наприклад: урок №1 — Шифр Цезаря, урок №2 — AES).
- Доцільно включати завдання для вивчення властивостей алгоритмів: аналіз чутливості до зміни ключа, симетрія, ефективність.

Технічні умови використання:

- Платформа не потребує зберігання даних на сервері, що дозволяє зберігати конфіденційність студентських експериментів.
- Доступна кросплатформність завдяки веб-інтерфейсу.

Перспективи розширення:

- Застосунок можна інтегрувати в систему дистанційного навчання (сДО), доповнити модулями перевірки знань або адаптувати для мобільних платформ.
- Існує можливість впровадження зворотного зв’язку – реєстрації успіхів, оцінювання тощо.

Таким чином, “Crypto Training App” не лише виконує функцію навчального симулятора, але й може стати основою для побудови повноцінного освітнього середовища для вивчення криптографії. Його впровадження в навчальний процес сприятиме підвищенню якості викладання інформаційної безпеки, розвитку навичок аналітичного мислення та практичного застосування знань у сфері захисту інформації.

ВИСНОВКИ

1. Проаналізовано сучасні криптографічні алгоритми, включаючи класичні, симетричні, асиметричні методи шифрування, а також алгоритми хешування та цифрового підпису.
2. Досліджено можливості платформи Java для реалізації криптографічних алгоритмів. Архітектура криптографії Java (JCA), розширення Java Cryptography Extension (JCE) та бібліотека Bouncy Castle розглянуті як основні інструменти для побудови криптографічних систем на Java.
3. Було оглянуто фреймворку Spring, включаючи його ключові модулі: Spring Boot, Spring MVC, Spring Security, Spring Data. Було вивчено механізми побудови REST API, обробки запитів, налаштування компонентів та управління життєвим циклом біна.
4. Розроблено та впроваджено веб-застосунок «Crypto Training App» з підтримкою навчальних, практичних та тестових модулів для вивчення криптографічних алгоритмів.
5. Реалізовано повнофункціональні модулі для роботи з такими алгоритмами: Caesar, Vigenere, Column Permutation, AES, Triple DES, RSA, ECDSA, MD5 та SHA-256. Всі алгоритми супроводжуються інтерактивною візуалізацією та поясненням обчислювальних кроків.
6. Було впроваджено систему тестування користувачів з перевіркою знань на основі вивченого матеріалу.
7. Застосунок було протестовано за допомогою модульних тестів та аналізу покриття коду за допомогою JaCoCo. Було отримано високі показники покриття бізнес-логіки , що свідчить про якість реалізації.
8. Розроблено рекомендації щодо використання веб-застосунку в навчальному процесі: як допоміжного інструменту для викладання криптографії у вищих навчальних закладах або як засобу самостійного навчання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Свєрлік В.К., Коломоєць Г.П. Розробка веб-застосунку для навчання криптографічним алгоритмам засобами платформи Java. Збірник наукових праць студентів, аспірантів, докторантів і молодих вчених «Молода наука–2025» / Запорізький національний університет. Запоріжжя. ЗНУ. 2025. Т.5. С. 211–213.
2. ENG-H1-2024-Report-russian-cyber-operations. URL: https://cip.gov.ua/services/cm/api/attachment/download?id=65898 (дата звернення: 06.06.2025).
3. Stallings W. Cryptography and Network Security: Principles and Practice. 5th ed. United States Edition. Boston. Pearson. 2010. 744 с. ISBN 978-0-13-609704-4. URL: https://z-lib.id/book/32837459-cryptography-and-network-security-4 (дата звернення: 06.06.2025).
4. Nita S.L., Mihailescu M.I. Cryptography and Cryptanalysis in Java: Creating and Programming Advanced Algorithms with Java SE 17 LTS and Jakarta EE 10. 1st ed. Springer. 2022. 232 p. ISBN 978-1-4842-8104-8. URL: https://z-lib.id/book/cryptography-and-cryptanalysis-in-java-creating (дата звернення: 06.06.2025).
5. Stinson D.R., Paterson M.B. Cryptography: Theory and Practice. 4th ed. Boca Raton. CRC Press. Taylor & Francis Group. 2018. 599 с. ISBN 9781138197015. URL: https://www.ic.unicamp.br/~rdahab/cursos/mo421-mc889/Welcome_files/Stinson-Paterson_CryptographyTheoryAndPractice-CRC%20Press%20%282019%29.pdf (дата звернення: 06.06.2025).
6. Oppliger R. Cryptography 101: From Theory to Practice. 1st ed. Boston. Artech House. 2021. 650 с. ISBN 978-1-63081-846-3. URL: https://edu.anarcho-copy.org/Against%20Security%20-%20Self%20Security/Cryptography%20101%20From%20Theory%20to%20Practice.pdf (дата звернення: 06.06.2025).
7. Alencar M.S. de. Cryptography and Network Security. River Publishers. 2022.

322 с. (River Publishers in Security and Digital Forensics). ISBN 978-87-7022-407-9. URL: https://z-lib.id/book/33306711-cryptography-and-network-security-7 (дата звернення: 06.06.2025).

8. Paar C., Pelzl J. Understanding Cryptography: A Textbook for Students and Practitioners. Springer. 2010. URL: https://archive.org/details/understandingcry0000paar/mode/2up (дата звернення: 06.06.2025).
9. Hook D. Beginning Cryptography with Java. Wiley Publishing Inc. 2005. URL: https://download.e-bookshelf.de/download/0000/5851/01/L-G-0000585101-0002361270.pdf (дата звернення: 06.06.2025).
10. Knudsen J.B. Java Cryptography. O'Reilly Media. 1998. URL: https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2010-2011/Java%20Cryptography.pdf (дата звернення: 06.06.2025).
11. Java Cryptography Architecture (JCA) Reference Guide. Oracle. URL: https://docs.oracle.com/en/java/javase/17/security/java-cryptography-architecture-jca-reference-guide.html (дата звернення: 06.06.2025).
12. Stallings W. Cryptography and Network Security: Principles and Practice. 5th ed. International Edition. Boston. Pearson. 2010. 744 с. ISBN 978-0-13-705632-3. URL: https://www.amazon.de/-/en/Cryptography-Network-Security-Principles-International/dp/013705632X (дата звернення: 06.06.2025).
13. About CrypTool 1. URL: https://www.cryptool.org/en/ct1/ (дата звернення: 06.06.2025).
14. Welcome to CrypTool-Online. URL: https://www.cryptool.org/en/cto/ (дата звернення: 06.06.2025).
15. About CrypTool 2. URL: https://www.cryptool.org/en/ct2/ (дата звернення: 06.06.2025).
16. Spring Framework Documentation. URL: https://docs.spring.io/spring-framework/docs/current/reference/html/ (дата звернення: 06.06.2025).
17. Spilca L. Spring Start Here: Learn what you need and learn it well. Manning Publications. 2021. 416 p. ISBN 978-1-61729-869-1. URL: https://z-

lib.id/book/32987936-spring-start-here (дата звернення: 06.06.2025).

18. Walls C. Spring in Action. Sixth Edition. Manning Publications. 2022. 520 p. ISBN 978-1-61729-757-1. URL: https://z-lib.id/book/spring-in-action-sixth-edition-1 (дата звернення: 06.06.2025).
19. Bouncy Castle Documentation. URL: https://www.bouncycastle.org/documentation/documentation-java/ (дата звернення: 06.06.2025).
20. Spring Boot Reference Documentation. URL: https://docs.spring.io/spring-boot/docs/current/reference/html/ (дата звернення: 06.06.2025).
21. JaCoCo Java Code Coverage Library. URL: https://www.jacoco.org/jacoco/ (дата звернення: 06.06.2025).
22. Rivest R. The MD5 Message-Digest Algorithm (RFC 1321). Internet Engineering Task Force. 1992. URL: https://www.ietf.org/rfc/rfc1321.txt (дата звернення: 09.06.2025).
23. National Institute of Standards and Technology. FIPS PUB 180-4: Secure Hash Standard (SHS). August 2015. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf (дата звернення: 09.06.2025).
24. Garcia B. Mastering Software Testing with JUnit 5: Comprehensive Guide to Develop High Quality Java Applications. 1st ed. Packt Publishing. 2017. 350 p. ISBN 978-1787124394. URL: https://www.packtpub.com/product/mastering-software-testing-with-junit-5/9781787124394 (дата звернення: 09.06.2025).

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, Свєрлік Віктор Костянтинович, студент 4 курсу, форми навчання денної, Інженерного навчально-наукового інституту ім. Ю.М. Потебні ЗНУ, спеціальність 121 Інженерія програмного забезпечення освітньої програми "Програмне забезпечення систем", адреса електронної пошти se121-17@stu.zsea.edu.ua, — підтверджую, що написана мною кваліфікаційна робота на тему «**Розробка Веб-застосунку для навчання криптографічним алгоритмам засобами платформи Java**» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

- згоден на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Дата 11.06.2025 _____ Свєрлік Віктор Костянтинович
(підпис) (прізвище та ініціали) (студент)

Дата 11.06.2025 _____ Коломоєць Геннадій Павлович
(підпис) (прізвище та ініціали) (керівник)