

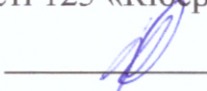
Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

ПОЯСНЮВАЛЬНА ЗАПИСКА

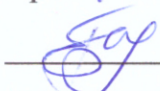
до випускної кваліфікаційної роботи бакалавра
на тему: «Проект програмного рішення для аналізу та
розв'язання прикладних криптографічних задач»

Виконав: студент 4-го курсу групи КБ-22-1

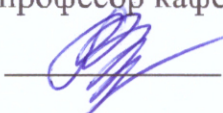
спеціальності 125 «Кібербезпека»

 Тарас ГОРБАТЮК

Керівник: старший викладач кафедри КІ та КБ

 Олександра ПОКОТИЛО

Рецензент: професор кафедри КІ та КБ, д.т.н., доцент

 Володимир ВОРОТНИКОВ

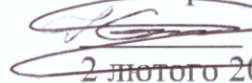
Житомир – 2026

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

ЗАТВЕРДЖУЮ

Завідувач

кафедри комп'ютерної
інженерії та кібербезпеки

 Андрій Єфіменко
2 лютого 2026 р.

ЗАВДАННЯ

на випускню кваліфікаційну роботу бакалавра

Студент: Горбатюк Тарас Миколайович

Тема роботи: «Проект програмного рішення для аналізу та розв'язання прикладних криптографічних задач»

затверджена Наказом університету від 2 лютого 2026 р. №58/с

Термін здачі студентом закінченої роботи 10 червня 2026 р.

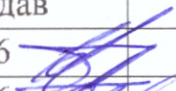
Вихідні дані роботи (зазначається предмет і об'єкт дослідження):

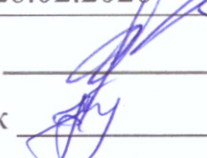
Об'єктом дослідження є процеси аналізу та перетворення інформації в прикладній криптографії. Предметом дослідження є методи, алгоритми та програмні засоби аналізу і перетворення текстових даних.

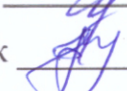
Обладнання та програмне забезпечення:

1. Інтегрована система розробки Visual Studio 2022
2. Операційна система Windows 10

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

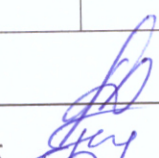
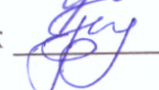
Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Бродський Ю.Б.	02.02.2026 	02.02.2026 
2	Бродський Ю.Б.	16.02.2026 	16.02.2026 
3	Бродський Ю.Б.	26.02.2026 	26.02.2026 

Студент  Тарас ГОРБАТЮК

Керівник  Олександра ПОКОТИЛО

Календарний план

№ з/п	Назви етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка завдання	16.02.2026	Виконано
2	Дослідження аналогів	01.03.2026 – 03.03.2026	Виконано
3	Проектування структури програми	04.03.2026 – 15.03.2026	Виконано
4	Розробка модуля перетворень тексту	16.03.2026 – 15.04.2026	Виконано
5	Розробка графічного інтерфейсу	16.04.2026 – 25.04.2026	Виконано
6	Розробка модулю аналізу тексту	26.04.2026 – 05.05.2026	Виконано
7	Тестування і виправлення недоліків	06.05.2026 – 10.05.2026	Виконано
8	Оформлення пояснювальної записки	11.05.2026 – 25.05.2026	Виконано
9	Отримання відгуку і рецензії	26.05.2026 – 05.06.2026	
10	Захист	10.06.2026	

Студент  Тарас ГОРБАТЮК
 Керівник  Олександра ПОКОТИЛО

АНОТАЦІЯ

У кваліфікаційній роботі розроблено програмне рішення для аналізу та розв'язання прикладних криптографічних задач. Метою роботи є створення зручного інструменту, який поєднує функції перетворення та аналізу текстових даних у єдиному середовищі.

У роботі проведено аналіз основних криптографічних задач, розглянуто методи кодування, шифрування та хешування, а також досліджено підходи до аналізу тексту. Виконано огляд існуючих програмних аналогів та обґрунтовано вибір засобів реалізації.

У результаті розроблено програмний комплекс з графічним інтерфейсом, що реалізує функції перетворення даних та їх аналізу, включаючи статистичні та структурні методи. Проведене тестування підтвердило працездатність та ефективність розробленого рішення.

Пояснювальна записка до випускної кваліфікаційної роботи містить 84 сторінки, 5 таблиць та 59 рисунків. Список використаних джерел містить 24 найменування і займає 3 сторінки.

Ключові слова: КРИПТОГРАФІЯ, КРИПТОАНАЛІЗ, ШИФРУВАННЯ, ХЕШУВАННЯ, АНАЛІЗ ТЕКСТУ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

					ККІК.КР.Б – 125 – 05 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Проект програмного рішення для аналізу та розв'язання прикладних криптографічних задач	Літ.	Арк.	Аркушів
Розроб.		Горбатюк Т.М.		5.06.21				
Керівник		Покотило О.А.		1.08.21			4	84
Реценз.		Воротніков В.В.		8.08.21		ФІКТ, Гр. КБ-22-1		
Н. контр.		Покотило О.А.		8.08.21				
Затверд.		Єфіменко А.А.		5.08.21				

ANNOTATION

In the qualification work, a software solution for analyzing and solving applied cryptographic problems was developed. The goal of the work is to create a convenient tool that combines the functions of transforming and analyzing text data in a single environment.

The work analyzes the main cryptographic problems, considers methods of encoding, encryption and hashing, and also investigates approaches to text analysis. A review of existing software analogues is carried out and the choice of implementation tools is justified.

As a result, a software complex with a graphical interface was developed that implements the functions of transforming data and analyzing them, including statistical and structural methods. The testing confirmed the operability and effectiveness of the developed solution.

The explanatory note to the final qualification work contains 84 pages, 5 tables and 59 figures. The list of sources used contains 24 names and occupies 3 pages.

Keywords: CRYPTOGRAPHY, CRYPTOANALYSIS, ENCRYPTION, HASHING, TEXT ANALYSIS, SOFTWARE.

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				5
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

ВСТУП.....	8
РОЗДІЛ 1 ТЕОРЕТИЧНИЙ АНАЛІЗ ПРИКЛАДНИХ КРИПТОГРАФІЧНИХ ЗАДАЧ.....	10
1.1 Огляд основних прикладних криптографічних задач та визначень.....	10
1.2 Аналіз перетворень інформації	12
1.3 Методи аналізу тексту.....	21
1.4 Аналіз аналогів та вибір способу реалізації теми	24
РОЗДІЛ 2 ПРОЄКТУВАННЯ ПРОГРАМНОГО РІШЕННЯ ДЛЯ АНАЛІЗУ ТА ВИРІШЕННЯ ПРИКЛАДНИХ КРИПТОГРАФІЧНИХ ЗАДАЧ.....	30
2.1 Проектування зовнішнього вигляду програмного рішення.....	30
2.2 Проектування загальних алгоритмів програмного рішення	32
2.3 Розробка алгоритмів перетворення тексту	41
2.4 Розробка алгоритмів аналізу тексту	53
РОЗДІЛ 3 ТЕСТУВАННЯ РОБОТИ ПРОГРАМНОГО РІШЕННЯ.....	61
3.1 Огляд та тестування модуля перетворення тексту	61
3.2 Тестування модуля аналізу тексту.....	74
ВИСНОВКИ.....	80
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	82
ДОДАТКИ.....	85

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				6
Змн.	Арк.	№ докум.	Підпис	Дата		

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

WPF – Windows Presentation Forms (Форми презентацій Windows).

ЕЦП – Електронний цифровий підпис.

ASCII – American Standard Code for Information Interchange (Американський стандартний код для інформаційного обміну).

UTF8 – Unicode Transformation Format (формат перетворення Юнікоду).

RFC – Request for Comments (запит на коментарі).

Hex – Hexadecimal (Шістнадцятковий).

MAC – Media Access Control (Керування доступом до середовища передачі даних).

ECB – Electronic codebook (Електронна книга кодів).

CBC – Cipher block chaining (Ланцюгування шифроблоків).

CFB – Cipher feedback (Зворотний зв'язок по шифротексту).

XOR – exclusive «OR» (Виключне «АБО»).

DES – Data Encryption Standard (Стандарт шифрування даних).

3DES – Triple Data Encryption Standard (Стандарт шифрування даних потрійний).

AES – Advanced Encryption Standard (Просунутий стандарт шифрування).

RC2 – Ron's Code 2 (Код Рона 2).

MD5 - Message-Digest 5 (Дайджест повідомлень 5).

RIPEMD160 - RACE Integrity Primitives Evaluation Message Digest (RACE Оцінка примітивів цілісності Дайджест повідомлень).

RACE - Research and development in Advanced Communications technologies in Europe (Дослідження та розробки в галузі передових комунікаційних технологій у Європі).

SHA – Secure Hash Algorithm (Безпечний алгоритм хешування).

URL – Uniform Resource Locator (Уніфікований локатор ресурсів).

CTF – Capture the Flag (Захоплення прапора).

XAML - Extensible Application Markup Language (розширювана мова розмітки застосунків).

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				7
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми. У сучасному цифровому середовищі питання захисту інформації набуває особливої важливості. З розвитком інформаційних технологій, мережесервісів та систем обробки даних зростає кількість загроз, пов'язаних із несанкціонованим доступом, модифікацією або підробкою інформації. У зв'язку з цим значну роль відіграє криптографія як основний інструмент забезпечення інформаційної безпеки. Разом із тим, ефективне розв'язання прикладних криптографічних задач потребує не лише використання алгоритмів перетворення даних, а й засобів їх аналізу. Існуючі програмні рішення або орієнтовані на окремі задачі, або мають складний інтерфейс, що ускладнює їх використання, особливо у навчальних цілях та при виконанні практичних завдань. Це обумовлює необхідність створення зручного програмного інструменту, який поєднує можливості перетворення та аналізу даних у єдиному середовищі.

Метою кваліфікаційної роботи є розробка програмного рішення для аналізу та розв'язання прикладних криптографічних задач, яке забезпечує зручний інтерфейс користувача та інтегрує функції перетворення і аналізу текстових даних.

Визначена мета обумовлює наступні завдання:

- провести теоретичний аналіз прикладних криптографічних задач та методів їх розв'язання;
- дослідити основні способи перетворення інформації, зокрема кодування, шифрування та хешування;
- проаналізувати методи аналізу тексту, що використовуються у криптоаналізі;
- виконати огляд існуючих програмних аналогів та визначити їх переваги і недоліки;
- сформулювати вимоги до програмного рішення;
- спроектувати структуру та інтерфейс програмного комплексу;
- реалізувати програмне рішення з використанням сучасних технологій;
- провести тестування розробленого програмного продукту та оцінити його ефективність.

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				8
Змн.	Арк.	№ докум.	Підпис	Дата		

Об’єктом дослідження є процеси аналізу та перетворення інформації у прикладних криптографічних задачах.

Предметом дослідження є методи, алгоритми та програмні засоби аналізу і перетворення текстових даних у контексті прикладної криптографії.

У процесі виконання кваліфікаційної роботи було розроблено проєкт програмного рішення, що поєднує функціонал перетворення тексту та його аналізу, реалізоване з використанням мови програмування C# та технології WPF, і призначене для розв’язання практичних криптографічних задач.

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				9
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1 ТЕОРЕТИЧНИЙ АНАЛІЗ ПРИКЛАДНИХ КРИПТОГРАФІЧНИХ ЗАДАЧ

1.1 Огляд основних прикладних криптографічних задач та визначень

Криптологія - наука, яка вивчає методи побудови та аналізу систем захисту інформаційних ресурсів, заснованих на математичних перетвореннях даних з використанням секретних параметрів [1]. Вона поділяється на:

- криптографію - наука про принципи, засоби та методи перетворення даних з метою приховування їх змісту, запобігання несанкціонованого використання або підробки;

- криптоаналіз - наука про методи та способи розкриття зашифрованих повідомлень, а також про тактику та стратегію їх застосування.

Криптографія є однією з ключових галузей сучасної інформаційної безпеки. Вона вирішує проблему забезпечення захисту даних від несанкціонованого доступу, модифікації та підробки. Базується на математичних методах перетворення інформації у форму, незрозумілу для сторонніх осіб, з можливістю подальшого відновлення лише за наявності відповідного ключа або знання алгоритму. У сучасному цифровому середовищі криптографія є невід'ємною складовою функціонування інформаційних систем, мережевих сервісів та програмного забезпечення.

Основними задачами криптографії є забезпечення:

- конфіденційності;
- цілісності;
- автентичності.

Конфіденційність – це захист від несанкціонованого ознайомлення зі змістом повідомлення при збереженні її доступності і цілісності. Це досягається шляхом шифрування даних, що робить їх недоступними для сторонніх осіб. Для цього використовуються симетричні і асиметричні алгоритми шифрування.

Цілісність гарантує неможливість модифікації неавторизованим користувачем, що гарантує, що дані не були змінені під час передачі або зберігання. Реалізується за допомогою хеш-функцій.

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				10
Змн.	Арк.	№ докум.	Підпис	Дата		

Автентичність дозволяє перевірити джерело повідомлення, щоб підтвердити особу чи пристрій, що беруть участь у розмові. Вона досягається використанням електронного цифрового підпису та протоколами ідентифікації.

В табл. 1.1 показано зведену інформацію про основні задачі криптографії та використовувані алгоритми для їх реалізації.

Таблиця 1.1

Основні задачі криптографії

Задача	Забезпечує	Досягається
Конфіденційність	Нечитабельність для третіх осіб	Симетричним і асиметричним шифруванням
Цілісність	Неможливість модифікації	Хеш-функціями
Автентичність	Перевірку джерела	ЕЦП, протоколи ідентифікації

Прикладні криптографічні задачі охоплюють широкий спектр практичних проблем, пов'язаних із захистом інформації. До них належать шифрування текстових даних, генерація та перевірка хешів, аналіз криптостійкості алгоритмів, а також виявлення закономірностей у зашифрованих або відкритих текстах.

Шифрування складається із зашифрування та розшифрування.

Зашифрування – процес перетворення відкритого тексту до виду, незрозумілого несанкціонованому користувачеві [2].

Розшифрування (дешифрування) – процес перетворення шифрованого повідомлення до початкової інформації (відкритого тексту) за допомогою певних правил шифру та відомого ключа.

Криптографічний ключ (ключ) – таємний стан деяких параметрів алгоритму криптографічного перетворення, що забезпечує вибір одного перетворення із сукупності можливих для даного алгоритму.

Симетричний шифр означає шифрування, де для шифрування та розшифрування використовують один і той самий ключ, що забезпечує високу швидкість обробки даних, але потребує безпечного обміну ключами. Принцип симетричного шифрування показаний на рис. 1.1.

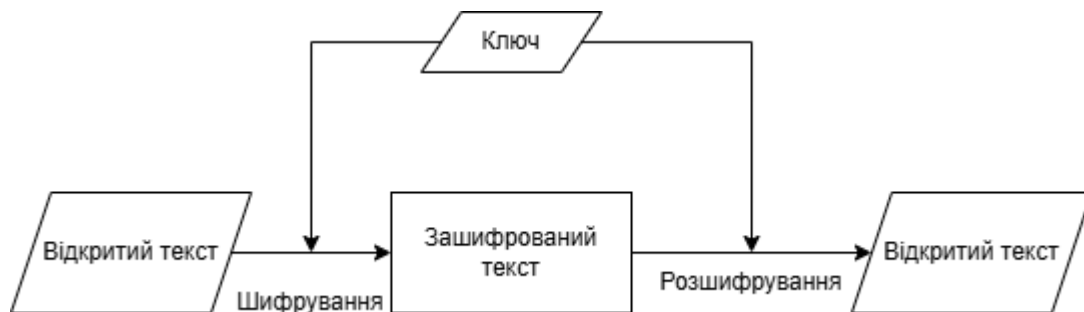


Рисунок 1.1 – Симетричне шифрування

Асиметричний шифр використовує два ключі – один для шифрування, інший для дешифрування, що дозволяє організувати безпечний обмін інформацією навіть у відкритих мережах. Принцип асиметричного шифрування показаний на рис. 1.2.

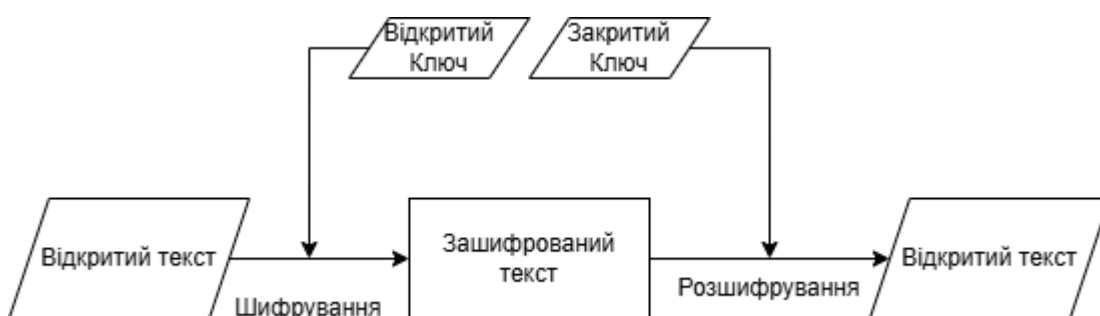


Рисунок 1.2 – Асиметричне шифрування

Хеш функції перетворюють текст довільної довжини у фіксовану за розміром послідовність біт, що називають хешем. При найменшій зміні оригінального тексту хеш змінюється повністю, що дозволяє застосовувати його для перевірки цілісності даних. Також результат хеш функції є незворотнім, тобто відтворити початкове повідомлення неможливо, маючи результат функції.

Таким чином, прикладні криптографічні задачі спрямовані на практичне забезпечення безпеки інформації шляхом використання різноманітних методів її перетворення та аналізу. Їх дослідження та реалізація є важливою складовою розробки сучасних програмних рішень, орієнтованих на обробку та захист даних.

1.2 Аналіз перетворень інформації

Основні перетворення інформації:

- кодування;
- шифрування;

– хешування.

Кодування – це перетворення інформації на зручний для передавання чи зберігання формат за допомогою системи символів.

Система символів – таблиця, яка задає певному символу його числовий номер або код, наприклад «A = 1», «B = 2» і т.д. [3]

Інформація на комп'ютері зберігається у вигляді послідовності 0 та 1, проте щоб цю інформацію можна було зручно прочитати людині, використовують системи кодування, які перетворюють 0 та 1 у читабельний рядок через вибрану систему символів.

Для відображення природньої мови найчастіше використовують кодування ASCII та UTF8.

ASCII (American Standard Code for Information Interchange) – система кодів, яка визначає для символів їх числову відповідність у межах від 0 до 127. Її було розроблено в 1960-х Американським Національним Інститутом стандартів. Ця таблиця кодів включає англійські літери, числа, знаки пунктуації та контрольні знаки. Для зберігання символу ASCII вистачає 7 біт, проте для забезпечення відповідності обробки комп'ютером даних на один символ виділяють 8 біт. Перевагою цього кодування є підтримка більшістю сучасних і старих систем, проте наявність тільки англійських символів дає обмеження на використання.

UTF-8 (Universal Character Set) – система кодів, яка визначає кодування для символів більшості писемних систем світу.[4] Цю систему було визначено в стандарті RFC-3629. Вона також повністю підтримує дані, закодовані системою ASCII. Для зберігання кожного символу визнається код, який може мати довжину від 1 до 4 октетів. Це дозволяє витрачати мінімально простір для зберігання символів та дає значний діапазон для підтримуваної кількості символів. На сьогоднішній день більшість веб-сторінок працюють на кодуванні UTF-8.

Проте дані в комп'ютерному сховищі можуть бути не тільки природньою мовою, і переглянути їх звичайним алфавітом не можливим. Тому для перегляду бінарних даних використовують різні числові системи: вісімкову, десяткову, шістнадцяткову.

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		13

Вісімкова (Octal) система використовує знаки від 0 до 7 для відображення даних. Кожне число відображає три біти, і сьогодні ця система використовується для відображення прав доступу для файлів у Unix-системах, наприклад:

- повний доступ (читання, запис, виконання - 111) відповідає 7;
- тільки перегляд (читання – 100) відповідає 4.

Десяткова (Decimal) система використовує знаки від 0 до 9, і ця система найбільш зрозуміла людині. Десяткове число може відображати будь-яку кількість біт, проте максимальне число росте в два рази з кожним новим бітом для читання. Так як в байті 8 біт, то для його відображення потрібні числа від 0 до 255.

Шістнадцяткова (Hexadecimal або Hex) система використовує символи від 0 до 9 та літери A-F. Система має 16 символів, тому символ відображає вміст 4 біт. Ця система зазвичай використовується для відображення бінарних даних, так як 2 шістнадцяткові символи читають 1 байт, що є зручно для розуміння. Також цю систему використовують у:

- кодуванні кольорів, де #FFFFFF позначає білий колір;
- відображенні MAC-адрес, наприклад 00:09:8C:00:69:63;
- кодування символів у веб-адресах, наприклад %20 кодує пробіл.

Інше застосування кодування – перетворення даних для передачі їх у певному середовищі. Найдавнішим таким кодуванням є азбука Морзе – вона перетворює букви та числа в послідовності коротких і довгих сигналів. Це кодування є давнім і примітивним, проте його досі широко використовують в радіозв'язку та військові, так як воно передає дані в середовищі, де сигнал може бути у станах «Є/Немає», наприклад світло від ліхтаря. Кодування Морзе показано на рис. 1.3. [5]

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		14

MORSE CODE ALPHABET		
A . _	M _ _	Y _ . _ _
B _ _ . .	N _ .	Z _ _ _ .
C _ . _ .	O _ _ _	1 . _ _ _ _
D _ _ .	P . _ _ .	2 . . _ _ _
E .	Q _ _ . _	3 . . . _ _
F . . _ .	R . _ .	4
G _ _ .	S . . .	5
H	T _	6 _
I . .	U . . _	7 _ _ . . .
J . _ _ _ _	V . . . _	8 _ _ _ . .
K _ . _	W . _ _	9 _ _ _ . .
L . _ . .	X _ . . _	0 _ _ _ _

Рисунок 1.3 – Кодування Морзе

Одним із прикладом сучасного кодування для передачі в іншому середовищі є Base64 – це спосіб перетворення бінарних даних в послідовність друкованих символів. Його було описано в стандарті RFC-4648 [6]. Це кодування потрібно для передачі даних в середовищах, які дозволяють тільки друковані символи. Base64 використовує 64 символи: a-z, A-Z, 0-9 та два резервних для різних середовищ. Кожен символ кодування відображає 6 біт даних.

Варто звернути увагу, що кодування не змінює вміст даних, а тільки змінює спосіб відображення. Одна і та ж бінарна послідовність відображається різними літерами і має різну довжину при різних кодуваннях. Приклад відображення даних різними кодуваннями показаний на табл. 1.2.

Таблиця 1.2

Вигляд 24 біт при різному кодуванні

Binary	0	1	1	0	0	0	0	1	0	0	1	1	0	0	0	1	0	0	1	1	1	1	1	1
Octal	3			0			2			3			0			4			7			7		
Hex	6				1				3				1				3				F			
Base64	Y						T						E						/					
Decimal	97								49								63							
ASCII	a								1								?							

В той час як кодування змінює відображення одних і тих же даних, шифрування змінює їх з метою приховування вмісту від осіб, яким не відомий ключ.

Перші алгоритми шифрування працювали на мовний алфавіт. Вони заміняли букву за певним правилом. Одним із найдавнішим прикладом цього є шифр Цезаря [7], який вважається, був винайдений ще Гаєм Юлієм Цезарем. Це є шифром зсуву, оскільки алфавіт шифротексту походить від алфавіту відкритого тексту шляхом зсуву кожної літери на певну кількість позицій. Наприклад, якщо ми використовуємо зсув на 10, то отримаємо таку пару алфавітів шифротексту та відкритого тексту, яка показана на рис. 1.6.

Відкритий текст	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Шифротекст	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J

Рисунок 1.4 – Приклад шифру Цезаря зі зсувом на 10 позицій

По цьому прикладу повідомлення CAESAR зашифрується як MKOCKB.

Покращеним популярним алфавітним шифром є шифр Віженера [8], який був винайдений у 16 столітті французом Блезем де Віженер. Замість зміщення на однакову кількість позицій всіх літер, шифр використовує слово як ключ, де кожна літера відображає кількість позицій для зсуву. Приклад шифрування Віженера показаний на рис. 1.5.

V	I	G	E	N	E	R	Відкритий текст
21	8	6	4	13	4	17	Значення літер відкритого тексту
S	E	C	R	E	T	S	Ключ
18	4	2	17	4	19	18	Значення літер ключа
13	12	8	21	17	23	9	Результат додавання по модулю 26
N	M	I	V	R	X	J	Шифроване повідомлення

Рисунок 1.5 – Приклад шифрування Віженера

В комп'ютерній техніці дані зберігаються в бінарному вигляді, тому алфавітне шифрування не працює. Для таких даних використовують алгоритми для шифрування цифрових даних. Ці алгоритми мають певні властивості:

- блок: частина повідомлення певної довжини в бітах;
- ключ: таємний стан параметрів певної довжини в бітах;

- вектор ініціалізації: випадкова послідовність бітів для ініціалізації алгоритму шифрування при певних режимах.
- кількість раундів: кількість однакових операцій перетворень блоку;
- режими роботи: визначає як застосовувати раунди.

Основними режимами роботи є: ECB, CBC, CFB.

ECB (Electronic codebook) – при цьому режимі повідомлення розділяється на блоки, і кожен блок шифрується окремо. Цей режим не є рекомендованим для шифрування, оскільки блоки можуть мати ідентичний вигляд при ідентичних даних у відкритому повідомленні. Алгоритм роботи ECB показаний на рис. 1.6.



Рисунок 1.6 – Режим ECB

CBC (Cipher block chaining) – у цьому режимі кожен блок відкритого тексту робить операцію XOR з результатом попереднього шифрування. Перший блок при цьому робить операцію XOR з вектором ініціалізації. Таким чином, кожен блок шифротексту залежить від усіх блоків відкритого тексту, оброблених до цього моменту, і зміна одного біта несе каскадний ефект на увесь наступний шифротекст. Алгоритм роботи CBC показаний на рис. 1.7.

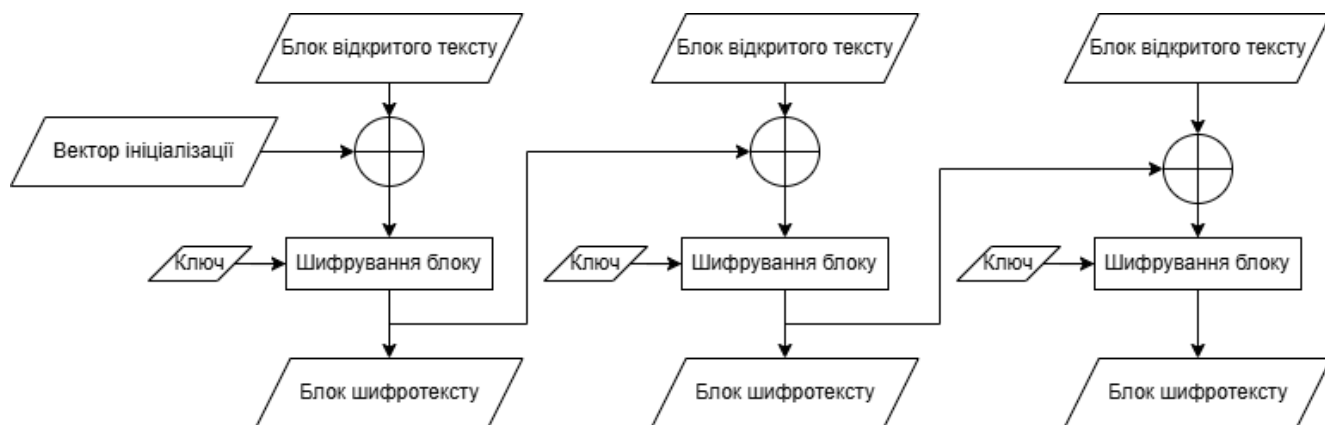


Рисунок 1.7 – Режим CBC

CFB (Cipher feedback) – також з попереднім режимом перетворює шифрування на потік з каскадним ефектом при зміні відкритого тексту. Тільки тут, результат раунду виконує операцію XOR з відкритий текст цього блоку, і після цього переходить до наступного раунду. Перший раунд виконується над вектором ініціалізації. Алгоритм роботи CFB показаний на рис. 1.8.

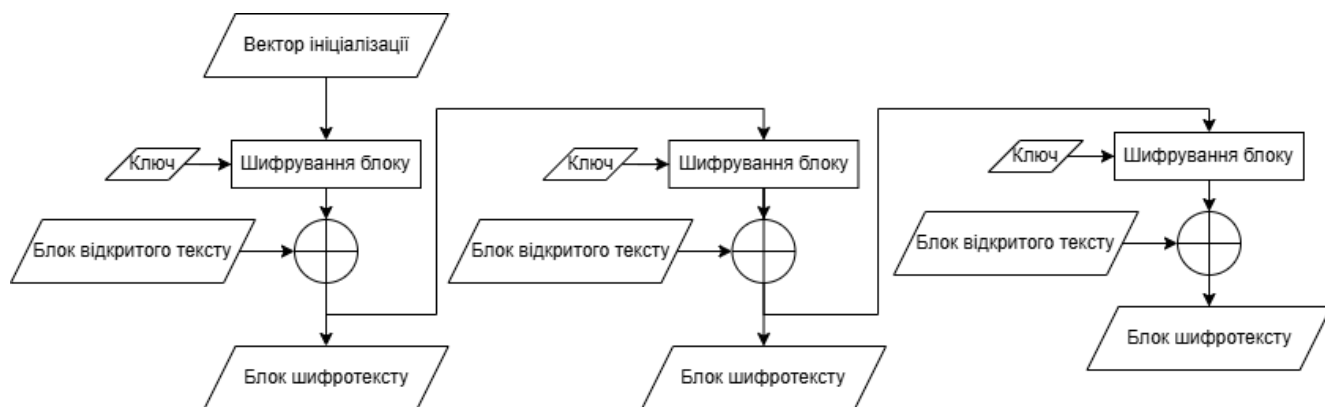


Рисунок 1.8 – Режим CFB

Одним з перших поширеним блоковим шифром став DES (Data Encryption Standard) [9]. Цей шифр було розроблено 1977 року як Федеральний стандарт обробки інформації в США. Цей алгоритм має розмір блоку 64 біт, розмір ключа 64 біт, виконує 16 раундів операцій. Проте 8 біт ключа використовуються для інших цілей під час роботи алгоритму, тому фактична довжина ключа тільки 56 біт. Раунд виконується мережею Фейстеля.

Для покращення алгоритму DES у 1981 році було розроблено шифр 3DES (Triple DES). Відмінність була у тому, що кожен блок проходив раунд шифрування DES тричі. Розміри ключа стали можливими в 128 та 192 біт.

Через втрату криптостійкості попередніх алгоритмів було розроблено шифр AES (Advanced Encryption Standard) [10]. Опубліковано його було в 2001 році Національним інститутом стандартів і технологій США. Цей алгоритм має розмір блоку 128 біт. Розмір ключа має декілька варіантів: 128, 192, 256 біт. Від розміру ключа залежить кількість раундів: 10, 12, 14. Раунд виконується мережею заміни-перестановок.

Іншим блоковим алгоритмом шифрування є RC2 (Ron's Code 2), який було офіційно опубліковано в 1998 [11]. Шифр має розмір блоку 64 біт, проте ключ може

мати розмір від 1 до 128 біт. Для блоку виконується 18 раундів, які складаються з 16 раундів типу «Mixing» та 2 раундів типу «Mashing».

Усі оглянуті алгоритми є симетричними. Це означає, що для шифрування та розшифрування використовується один і той же ключ. Загальне порівняння оглянутих блокових шифрів можна побачити на табл. 1.3.

Таблиця 1.3

Порівняння симетричних блокових шифрів

Назва	Розмір блоку	Розмір ключа	Кількість раундів	Алгоритм раунда
DES	64	64	16	Фейстеля
3DES	64	128	48	Фейстеля
		192		
AES	128	128	10	Заміна-перестановка
		192	12	
		256	14	
RC2	64	1-128	16+2	Mixing + Mashing

На відміну від кодування і шифрування, хешування – це одностороннє перетворення даних, тобто початкові дані неможливо відновити з результату хеш функції [12]. Ці функції є детерміновані, що означає незмінність результату при однакових вхідних даних. Іншою властивістю хешування є однакова довжина результату в бітах для певної хеш функції.

Одним з перших розповсюджених алгоритмів хешування є MD5 (Message-Digest 5) [13]. Його було опубліковано 1992 року. Вхідне повідомлення розділяється на блоки довжиною 448 біт та доповнюється 64 бітами, що зберігають довжину повідомлення. Якщо повідомлення має меншу довжину, то воно доповнюється додатковими бітами, щоб задовільнити необхідних розмір блоку. Результат хешу має розмір 128 біт. Алгоритм виконує 4 раунди по 16 операцій над блоком.

Іншим алгоритмом хешування є RIPEMD160 (RACE Integrity Primitives Evaluation Message Digest 160) [14]. Це 160-бітна хеш функція, розроблена для заміни попередньо описаного хеш алгоритму. Алгоритм був розроблений під керівництвом Європейського проєкту Дослідження та розробки в галузі передових комунікаційних технологій.

Іншими прикладом популярних алгоритмів шифрування є сімейство SHA (Secure Hash Algorithm), розроблених Агентством національної безпеки США.

Першою версією є SHA-1, яку було опубліковано в 1995 році [15]. Результат хешу має розмір 160 біт, розмір блоку 512 біт. Алгоритм виконує 80 раундів над блоками.

Друга версія SHA-2 була опублікована в 2001 році [16]. Вона включає декілька алгоритмів: SHA-256, SHA-384 та SHA 512. Ці алгоритми відповідно мають довжини хешу 256, 384 та 512 бітів. SHA-256 має розмір блоку 512 біт та виконує 64 раундів над ним. SHA-384 та SHA-512 мають розмір блоку 1024 біт та виконують 80 раундів. Порівняння розглянутих хеш функцій показано в таблиці 1.4.

Таблиця 1.4

Порівняння хеш-функцій

Назва	Розмір хешу	Розмір блоку	Кількість раундів	Дата публікації
MD5	128	512	64	1992
RIPEMD160	160	512	80	1996
SHA-1	160	512	80	1995
SHA-256	256	512	64	2001
SHA-384	384	1024	80	2001
SHA-512	512	1024	80	2001

Розглянуті алгоритми кодування, шифрування та хешування є основними видами перетворення інформації, що забезпечують її представлення, захист і контроль цілісності. Проте для ефективного застосування цих методів також важливе

розуміння властивостей самої інформації. Саме тому наступним етапом є розгляд методів аналізу інформації.

1.3 Методи аналізу тексту

Аналіз тексту дозволяє зрозуміти її структуру, статистичну характеристику, потенційні закономірності, що має важливе значення у прикладних криптографічних задачах.

Основні методи аналізу тексту:

- статистичний;
- структурний;
- лінгвістичний.

Статистичний аналіз охоплює багато метрик, але основними є кількість літер, частотний аналіз та кількісна міра інформації. Кількість літер показує довжину повідомлення, та необхідна для розрахунку наступних метрик.

Частотний аналіз – метод аналізу тексту, який оснований на нерівномірності повторюваності букв при використанні природньої мови. Він корисний для аналізу шифротекстів, зашифрованих алфавітними шифрами заміни, так як вони передбачають, що кожній літері відкритого тексту відповідає літера шифротексту, і через це частотний розподіл букв зберігається. Щоб користуватись цим аналізом, необхідна еталонна статистика для потрібного алфавіту. Для українського алфавіту характерний розподіл, показаний на рисунку 1.9. [17]

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				21
Змн.	Арк.	№ докум.	Підпис	Дата		

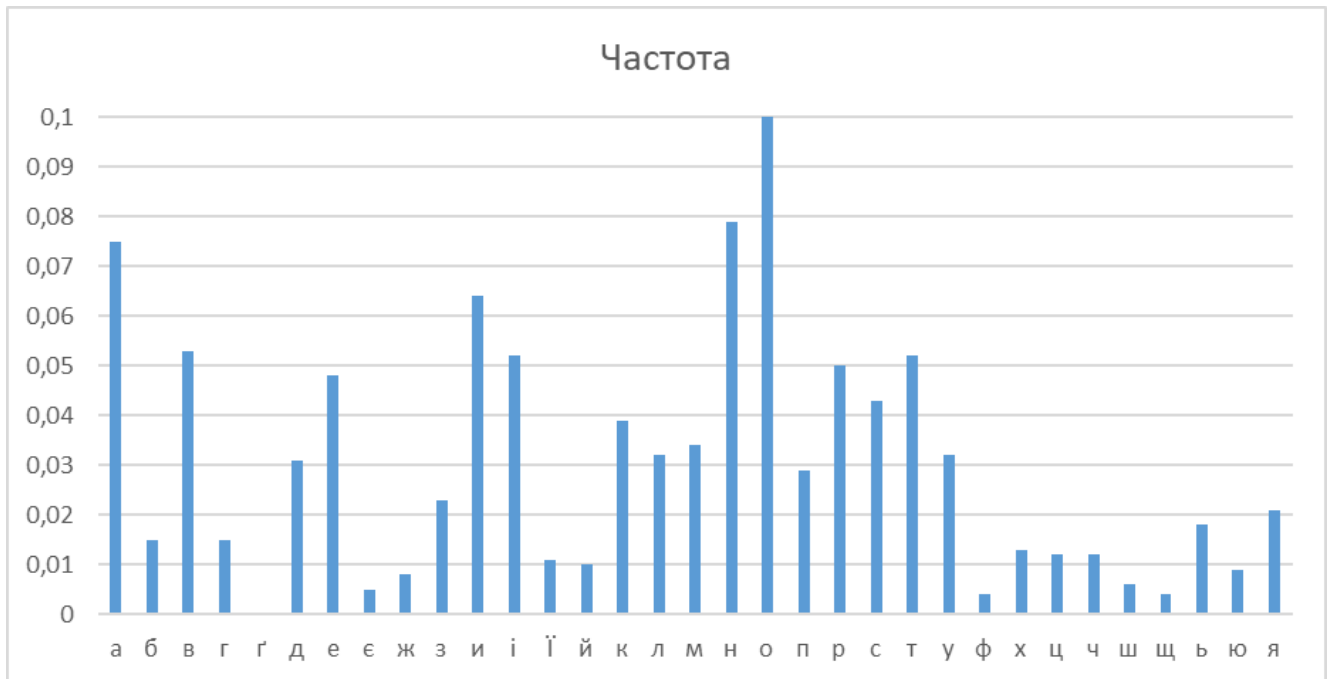


Рисунок 1.9 - Середньостатистичні частоти букв в українській мові

Для аналізу систем, в яких дані можуть бути іншими від природньої мови, існує характеристика кількісної міри інформації [18]. Ця міра визначає обсяг даних, що зменшує невизначеність знань про систему. Тобто, визначає обсяг даних, які створюються джерелом даних. Для обрахунку ентропії використовується формула Шенона:

$$H(x) = - \sum_{i=1}^N p_i \log p_i,$$

де p_i – це ймовірність появи символу;

N – розмір алфавіту.

Ентропія Шенона допомагає визначити, чи дані структуровані або неструктуровані. Алгоритми шифрування створюють неструктуровані та випадкові на вигляд дані. Результат обрахунку ентропії означає наступне:

- 0 означає відсутність випадковості, тобто всі значення повідомлення однакові;
- 3.5 – 5 зазвичай є межами природньої мови;
- 7.5 і більше мають добре зашифровані повідомлення достатньої довжини.

Структурний аналіз передбачає аналіз довжин слів, повторів та шаблонів. Довжина слова визначає, чи може текст бути природньою мовою, так як їх характерні певні довжини слів. Якщо після шифрування текст зберігає довжини слів, можливо визначити:

- визначити можливу мову;
- знаходити типові короткі слова;
- підібрати відповідники для дешифрування.

Якщо ж у повідомленні відсутні пробіли, тоді можливе визначення алгоритму шифрування чи хешування за довжиною даних.

Повтори у даних визначають, чи є текст випадковим. У разі їх наявності можливе визначення повторюваних шаблонів, що відкриває можливості для частотного аналізу чи простих перестановок. Більший обсяг даних дає кращі результати аналізу повторів, та дає більше можливостей для виявлення шаблонних фрагментів. Також цей метод використовується для визначення довжини ключа у поліалфавітних шифрах, за допомогою визначення відстані між повторами та пошуку спільного дільника цього числа.

Лінгвістичний аналіз визначає використовувані алфавіти у повідомленні. У випадку природньої мови це очевидне визначення мови, але у повідомленні з іншими символами це визначає його кодування та можливий шифр. При визначенні кодування з'являється можливість легко перетворити повідомлення до читабельного тексту. Наприклад, при використанні наступних словників в повідомленні визначається характеристика:

- символи 0-7 показують вісімкове кодування;
- символи 0-9,A-F показують шістнадцяткове кодування;
- символи a-z,A-Z,0-9 без пробілів можуть означати кодування Base64;
- інші спеціалізовані словники для: URL адрес, шлях до файлів, електронних пошт і т.д.

Таким чином, аналіз тексту є невід'ємною складовою розв'язання прикладних криптографічних задач, оскільки дозволяє досліджувати як відкриті, так і за-

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				23
Змн.	Арк.	№ докум.	Підпис	Дата		

шифровані дані з метою виявлення їхніх статистичних, структурних і лінгвістичних характеристик. Поєднання методів аналізу з інструментами перетворення даних створює основу для ефективного дослідження та практичного розв’язання криптографічних задач у межах розроблюваного програмного рішення.

1.4 Аналіз аналогів та вибір способу реалізації теми

У процесі розробки програмного рішення для аналізу та розв’язання прикладних криптографічних задач доцільно розглянути існуючі інструменти, які реалізують подібний функціонал. Для цього були вибрані інструменти CyberChef, dCode та CrypTool2.

Першим є CyberChef [19] - це веб-орієнтований інструмент, який позиціонується як «універсальний набір» для обробки даних. Він дозволяє виконувати широкий спектр операцій, зокрема кодування, декодування, шифрування, хешування, компресію та аналіз даних.

Ключовою особливістю цього інструменту є концепція «рецептів», яка дозволяє комбінувати кілька операцій у послідовний ланцюг обробки даних. Це забезпечує високу гнучкість і можливість автоматизації складних перетворень без написання коду. Вигляд CyberChef показаний на рис 1.10.

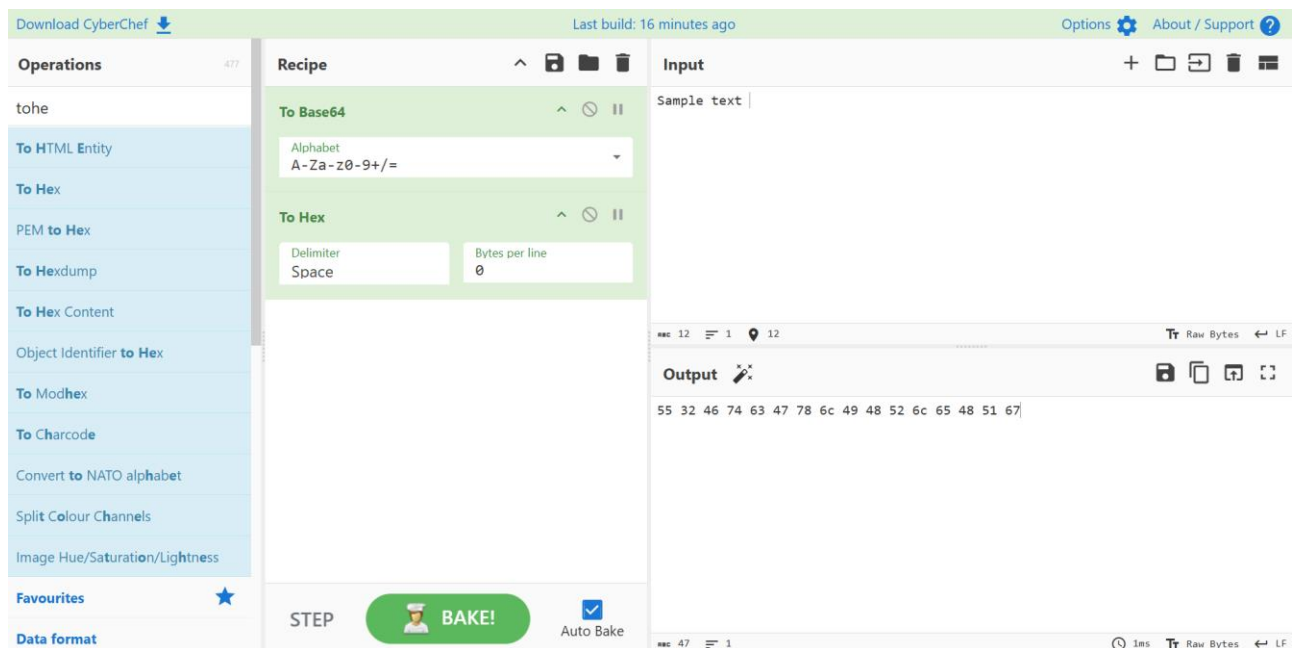


Рисунок 1.10 – Вигляд інструменту CyberChef

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				24
Змн.	Арк.	№ докум.	Підпис	Дата		

CyberChef підтримує сотні операцій, включаючи криптографічні алгоритми, перетворення форматів, аналіз тексту та мережевих структур. Завдяки цьому він широко застосовується у сфері кібербезпеки, зокрема для аналізу невідомих даних та CTF-завдань.

Можна виділити наступні переваги:

- високу універсальність;
- інтуїтивний інтерфейс;
- можливість комбінування операцій.

Проте недоліками є:

- відсутність глибокого криптоаналізу;
- орієнтація більше на перетворення, ніж на аналіз.

Другим інструментом є dCode [20] – це веб-платформа, яка спеціалізується на розв’язанні криптографічних, логічних і математичних задач. Вона містить понад 900 інструментів для роботи з кодами, шифрами та алгоритмами. Основною особливістю dCode є його орієнтація саме на криптоаналіз і розв’язання задач.

Сервіс підтримує автоматичне визначення типу шифру, здатний розпізнавати сотні алгоритмів, що значно спрощує роботу з невідомими повідомленнями. Вигляд платформи показаний на рис. 1.11.



Рисунок 1.11 – Онлайн-платформа dCode

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		25

Переваги:

- потужні інструменти криптоаналізу;
- велика база алгоритмів;
- автоматичне визначення шифру;
- орієнтація на практичні задачі.

Недоліки:

- кожен інструмент на окремій сторінці;
- відсутність єдиного середовища обробки;
- закритий вихідний код;
- обмежені можливості комбінування операцій.

Останнім вибраним засобом є CrypTool2 [21] – це програмне забезпечення, орієнтоване на навчання криптографії та криптоаналізу. Воно реалізує широкий спектр алгоритмів та дозволяє візуалізувати їхню роботу.

Програма використовує концепцію візуального програмування, де користувач створює схеми обробки даних, поєднуючи окремі компоненти. Це дозволяє будувати складні криптографічні процеси та аналізувати їх на різних рівнях. Вигляд CrypTool2 показано на рис. 1.12.

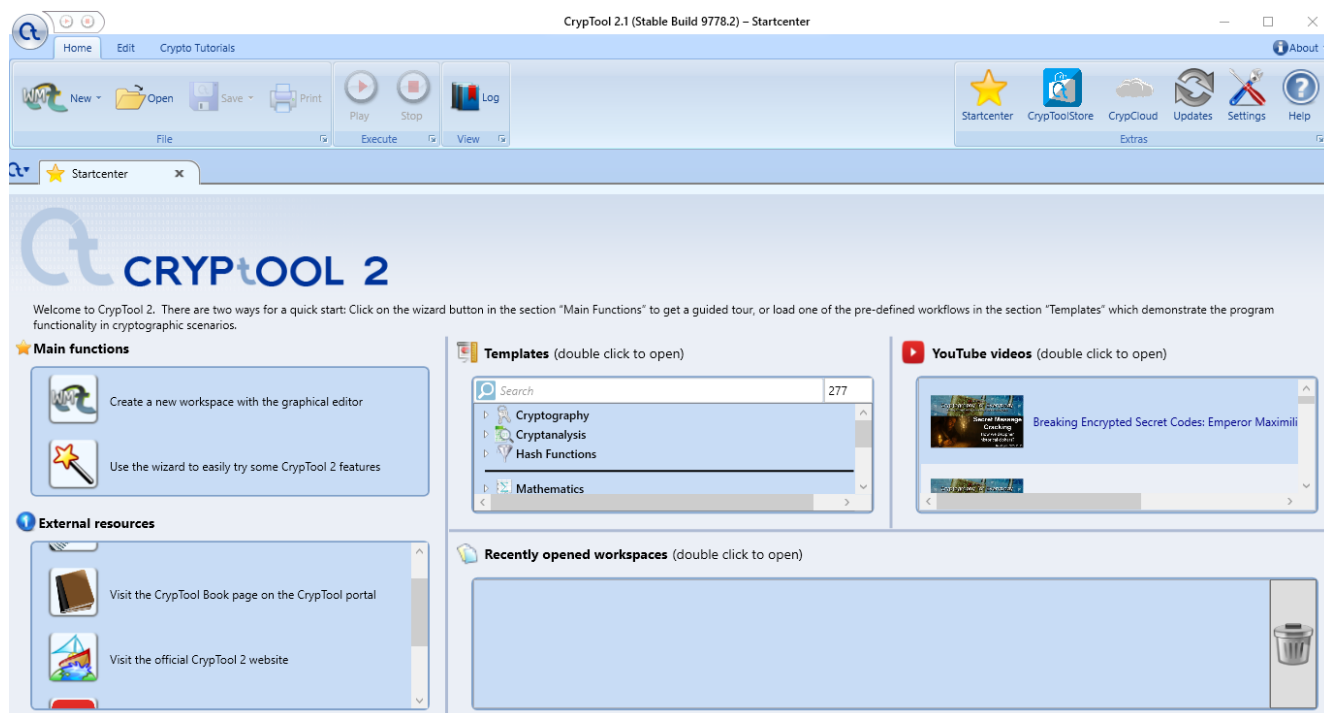


Рисунок 1.12 – Програмне забезпечення CrypTool2

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				26
Змн.	Арк.	№ докум.	Підпис	Дата		

CrypTool2 містить понад 200 компонентів і шаблонів, що охоплюють як класичні, так і сучасні алгоритми криптографії. Також він підтримує різні методи криптоаналізу, включаючи частотний аналіз, визначення довжини ключа та навіть атаки на сучасні алгоритми.

Переваги:

- широкий функціонал;
- підтримка сучасної криптографії;
- глибокі можливості аналізу;
- візуалізація алгоритмів;
- відкритий код.

Недоліки:

- складність освоєння;
- перевантажений інтерфейс;
- необхідність встановлення;
- орієнтація більше на навчання, ніж на швидкі задачі.

Розглянуті інструменти мають різну спрямованість:

- CyberChef – універсальний інструмент для перетворення та обробки даних;
- dCode – спеціалізований сервіс для криптоаналізу та розв’язання задач;
- CrypTool2 – комплексна система для навчання та глибокого дослідження криптографії.

CyberChef забезпечує максимальну зручність і швидкість роботи, але поступається у глибині аналізу. dCode, навпаки, має потужні інструменти для дешифрування, але не забезпечує єдиного середовища роботи. CrypTool2 є найбільш функціональним рішенням, проте його складність обмежує використання для швидких практичних задач.

Це свідчить про відсутність універсального інструменту, який би поєднував простоту використання, інтегроване середовище обробки та достатній рівень аналітичних можливостей. У висновку аналізу аналогів виділено наступні вимоги, які вирішено реалізувати в програмному рішенні:

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		27

- зручний інтерфейс користувача;
- концепція «рецепту», так як вона найбільш інтуїтивна для створення ланцюжків розв’язку прикладних криптографічних задач;
- модуль перетворення тексту з можливістю кодування, шифрування та хешування інформації;
- модуль аналізу тексту з можливостями статистичного, лінгвістичного та структурного аналізу.

В якості інструментарію реалізації було вибрано об’єктно-орієнтовану мову програмування C# для внутрішньої логіки програми та WPF (Windows Presentation Forms) для зовнішнього інтерфейсу. Середовищем розробки обрано Visual Studio 2022.

Мова програмування C# [22] є сучасною об’єктно-орієнтованою мовою, що входить до платформи .NET і широко використовується для розробки прикладного програмного забезпечення. Вона забезпечує зручні засоби роботи з пам’яттю, розвинену систему типів, підтримку об’єктно-орієнтованих принципів та вбудовані бібліотеки для реалізації складних алгоритмів, що робить її придатною для створення програм, пов’язаних із обробкою та аналізом даних.

Для розробки графічного інтерфейсу було використано технологію Windows Presentation Foundation [23], яка дозволяє створювати сучасні настільні застосунки з гнучким і масштабованим інтерфейсом. WPF підтримує декларативний підхід до побудови інтерфейсу за допомогою XAML, забезпечує розділення логіки та представлення, а також надає широкі можливості для роботи з графікою, стилями та прив’язкою даних.

Середовищем розробки обрано Visual Studio 2022 [24] – інтегровану систему, що надає повний набір інструментів для створення, налагодження та тестування програм. Воно підтримує роботу з мовою C# і технологією WPF, має зручний редактор коду, систему управління проектами, а також засоби відлагодження, що значно підвищує ефективність розробки програмного забезпечення.

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				28
Змн.	Арк.	№ докум.	Підпис	Дата		

Висновки до першого розділу

У першому розділі було розглянуто основні теоретичні аспекти прикладної криптографії, зокрема її основні задачі, методи перетворення інформації, а саме кодування, шифрування, хешування, та підходи до аналізу текстових даних.

Проведений огляд показав, що ефективне розв'язання прикладних криптографічних задач потребує поєднання засобів перетворення та методів аналізу, які дозволяють досліджувати структуру і властивості даних.

Також було проаналізовано існуючі програмні аналоги та обґрунтовано вибір інструментальних засобів розробки, що створює основу для подальшого проектування та реалізації власного програмного рішення.

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		29

РОЗДІЛ 2 ПРОЄКТУВАННЯ ПРОГРАМНОГО РІШЕННЯ ДЛЯ АНАЛІЗУ ТА ВИРІШЕННЯ ПРИКЛАДНИХ КРИПТОГРАФІЧНИХ ЗАДАЧ

2.1 Проєктування зовнішнього вигляду програмного рішення

Програмну рішення для аналізу та вирішення прикладних криптографічних було вирішено розділити на два модулі:

- модуль перетворення тексту;
- модуль аналізу тексту.

Модуль перетворення тексту має мати зручний інтерфейс для виконання завдань по перетворенню тексту. Тому головними елементами мають бути:

- список наявних функцій перетворення;
- список «рецептів»;
- поле вводу інформації;
- поле для результату перетворення;

Розташування елементів на інтерфейсі було вирішено зробити у вигляді «з ліва на право», щоб елементи інтерфейсу можна було зручно переглядати у порядку, схожому з читанням тексту. Розплановане розташування елементів на інтерфейсі модуля перетворення тексту показано на рис. 2.1.

Список операцій	Рецепт	Поле для вхідного тексту
Операція 1	Виконувана операція 1	
Операція 1	---	
Операція 3	Виконувана операція N	Поле для результату перетворення

Операція N		

Рисунок 2.1 – Проєктування інтерфейсу для модуля перетворення тексту

«Список операцій» буде містити перелік всіх реалізованих функцій перетворення тексту. Перелік функцій буде розділено на підкатегорії, а саме: «Формат даних», «Шифрування», «Хешування», «Інше». Підкатегорії можна буде згортати та розгортати, щоб переглядати тільки ті підкатегорії, які цікавлять користувача. Також важливий функціонал пошуку потрібної функції за її назвою. Пошук необхідний для швидкого визначення, чи є присутня реалізація функції, яка цікавить користувача, або необхідно швидко знайти потрібну функцію, і ручний перегляд кожної підкатегорії може бути довгим. Для кожної функції можна будуть можливості переглянути її короткий опис, та додати її до «Рецепту».

«Рецепт» це послідовний перелік операцій, які було додано зі «Списку операцій». Програма при обрахуванні результату перетворення буде послідовно проходити кожну додану функцію, та виконувати відповідну операцію над введеним текстом.

Поля для вхідного тексту та результату відповідно, слугують для вводу тексту, над яким буде виконуватись обрахунок операцій, та для виведення результату. Серед іншого функціоналу полів важливе елементарне форматування тексту, взаємодія з файловою системою та буфером обміну, виведення простої статистики тексту.

Модуль аналізу тексту має мати зручний інтерфейс для виконання прикладних операцій аналізу. Для раніше згаданих видів аналізу необхідні наступні елементи інтерфейсу:

- поле вводу тексту;
- поле з аналітичними даними;
- поле для виводу результату частотного аналізу;

«Вхідні дані» приймають текст, який буде аналізуватись. Також важлива реалізація простого форматування тексту, взаємодії з файловою системою та буфером обміну для зручного користування.

«Аналіз» виводить результати обрахованих характеристик тексту. В цьому полі можливе налаштування виконуваних операцій аналізу. Це поле має містити обрахування ентропії, виявлених алфавітів та повторів.

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				31
Змн.	Арк.	№ докум.	Підпис	Дата		

Результат частотного аналізу зручніше всього сприймати у вигляді гістограм, де по осі X вказано літери, а по осі Y їх частота. Для виведення такої гістограми необхідний значний простір, тому елемент «Частотний аналіз» займає значну частину екрану. Також для частотного аналізу важливе порівняння з еталонами розподілу частот літер у мовах, тому в інтерфейсі має бути місце для виведення еталонної гістограми потрібної мови.

Маючи на увазі всі ці вимоги, інтерфейс для модуля аналізу тексту було спроектовано, як показано на рис. 2.2.

Поле для вхідного тексту	Аналіз		
	Ентропія	Алфавіти	Повтори
Результат частотного аналізу	Еталонний розподіл частот		

Рисунок 2.2 - Проектування інтерфейсу для модуля аналізу тексту

Визначені два модулі є окремими, перемикання між ними буде відбуватись по типу перемикання між вкладками. Для цього в мові XAML існує елемент <TabControl>, в якому ці модулі можна розподілити під різні <TabItem>.

2.2 Проектування загальних алгоритмів програмного рішення

Модуль перетворення тексту має містити велику кількість реалізованих операцій різних типів. Це створює значний виклик для проектування програмного рішення без повторюваного коду, щоб нові операції можна було легко інтегрувати в програму. Використовувані інструменти, а саме C# та XAML, дозволяють

створювати власні елементи інтерфейсу з потрібними властивостями та зовнішнім виглядом, що значно полегшує створення такого алгоритму.

Для роботи такого алгоритму було вирішено зробити реалізацію, яка складається з таких створених елементів інтерфейсу:

- «OperationBar» для «Списку операцій»;
- «RecipeUniversal» для «Рецепту»;
- інші елементи «Recipe_X» для унікальних випадків в «Рецепті».

«OperationBar» зберігає в собі назву операції та її опис для зовнішнього вигляду на собі, код операції для передання його у функцію додання операції в «Рецепт». Лістинг коду файлу XAML з основними параметрами, які описуються, наступний:

```
<DockPanel Background="#FFF6F6FF">
    <Button
    Command="{Binding ElementName=parent, Path=ClickCommand}"
    CommandParameter="{Binding ElementName=parent, Path=CommandParameter}"
    />
    <Label
    Content="{Binding Path=NameOperationLabel, ElementName=parent}"/>
</DockPanel>
```

Зовнішній вигляд «OperationBar» показаний на рис. 2.3.



Рисунок 2.3 – Вигляд «OperationBar»

Лістинг коду C# з основними параметрами, які описуються, наступний:

```
public string NameOperationLabel {
    get { return (string)GetValue(NameLabelProperty); }
    set { SetValue(NameLabelProperty, value); } }
public ICommand ClickCommand {
    get { return (ICommand)GetValue(ClickCommandProperty); }
    set { SetValue(ClickCommandProperty, value); } }
public object CommandParameter {
    get { return GetValue(CommandParameterProperty); }
    set { SetValue(CommandParameterProperty, value); } }
public static readonly DependencyProperty NameLabelProperty =
DependencyProperty.Register("NameOperationLabel", typeof(string),
typeof(OperationBar));
```

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		33

```

public static readonly DependencyProperty ClickCommandProperty =
DependencyProperty.Register("ClickCommand", typeof(ICommand),
typeof(OperationBar), new PropertyMetadata(null));
public static readonly DependencyProperty CommandParameterProperty
= DependencyProperty.Register("CommandParameter", typeof(object),
typeof(OperationBar), new PropertyMetadata(null));

```

За допомогою конструкцій типу «Binding» у файлі XAML та «DependencyProperty» в кодї C# цей елемент може мати властивості, якими можна зручно керувати ззовні. Значення цих властивостей наступне:

- «ClickCommand» дозволяє задати назву функції, яка буде викликатись;
- «CommandParameter» зберігає код операції;
- «NameOperationLabel» дозволяє зручно задати назву команди ззовні.

Краще це зрозуміти можна, якщо подивитись на приклад додавання елементу «OperationBar» в головному вікні, що наведено в наступному лістингу:

```

<myint:OperationBar
NameOperationLabel="To Hex"
ClickCommand="{Binding AddOperationCommand}"
CommandParameter="ToHex"/>

```

Зовнішній вигляд доданого елементу «OperationBar» із заданими параметрами показаний на рис. 2.4.



Рисунок 2.4 – «OperationBar» із встановленим функціоналом

В цьому прикладі було створено елемент списку операцій, який зберігає значення для функції перетворення даних до шістнадцяткового вигляду. Проте користувач може спостерігати тільки один параметр – «NameOperationLabel». Подібно до цього параметру ще задається параметр опису операції, який показується при наведенні мишкою на картинку «i». Проте цей параметр не було описано вище, так як він не важливий для пояснення алгоритму роботи. Інші два параметри «ClickCommand» та «CommandParameter» слугують для виконання подальшої логіки роботи алгоритму.

В якості заданого параметру для параметру «ClickCommand» було задано значення «AddOperationCommand» - це назва функції програмного коду, яка буде викликатись при натисненні на кнопку «стрілка вправо». «CommandParameter» в

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		34

цьому прикладі зберігає значення «ToHex» - його буде передано у функцію програмного коду «AddOperationToReceiptList» як параметр.

Наступним розглянемо алгоритм роботи функції програмного коду, яка була вище згадана, повна версія якої показана в Додатку А, а основна частина лістингу функції наступна:

```
private void AddOperationToReceiptList(object parameter){
    bool delimiter = true,
        numPicker = true;

    Recipe_Universal recipeItem = new Recipe_Universal();

    switch (parameter)
    {
        case "ToHex":
            recipeItem = new Recipe_Universal("To Hex", "toHex",
                delimiter, numPicker, "Байтів на рядок");
            break;

        //...
    }

    RecipeList.Items.Add(recipeItem);
}
```

Функція «AddOperationToReceiptList» створює елемент «Recipe_Universal» і залежності від переданого параметру наповнює його потрібними даними. Центральною частиною функції є команда «switch». Саме вона в залежності від отриманого параметру переходить до потрібного випадку виконання та створює «Recipe_Universal» з потрібним наповненням. У випадку натиснення «стрілки вправо» на прикладі «To Hex» вище, команда «switch» створила б новий «Recipe_Universal», задала заголовок, код «рецепту», вказала присутність вибору «роздільника», можливість вибору числа та заголовок вибору числа. Після цього робота команди «switch» закінчується, і в кінці функції створений елемент додається до списку «Рецептів». Зовнішній вигляд списку рецептів після виконання функцій в цьому прикладі показаний на рис. 2.5.

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		35

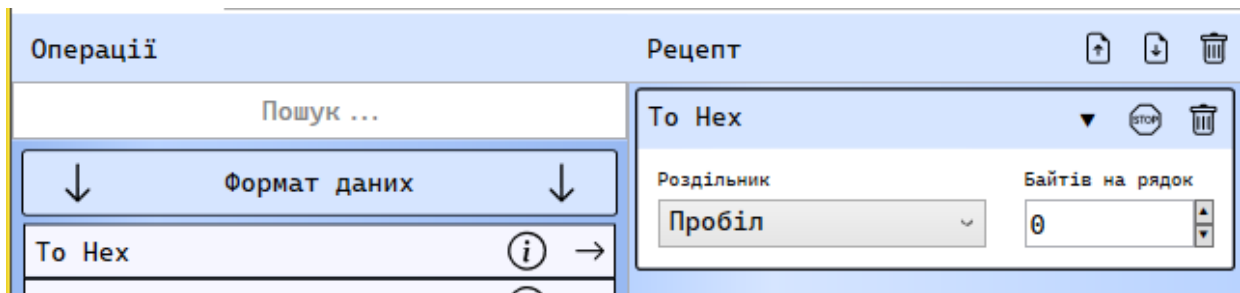


Рисунок 2.5 – Доданий елемент «To Hex»

Мета створеного елемента інтерфейса «Recipe_Universal» у розміщені необхідного функціоналу для більшості елементів рецепту, які можуть мати різні, проте більш-менш однакові потреби – в одному універсальному елементі. Цей елемент інтерфейсу забезпечує такі можливості для елементів рецепту:

- вибір роздільника;
- вибір числа;
- задання заголовку для вибору числа.

Ці можливості задаються при створенні елемента в функції «AddOperationToReceiptList». Тим самим «Recipe_Universal» підходить для операцій, яким потрібне вказання роздільника та задання числового параметра для будь-якої мети. В прикладі «To Hex» цим числовим параметром є кількість байтів на рядок. Опція задання можливостей також означає, що елементу «Рецепту» не обов'язково мати потребу в цих можливостях, і тому елементи «Рецепту», яким ці можливості не потрібні, також користуються цим елементом інтерфейсу. Такий алгоритм роботи програми необхідний для зменшення повторюваності коду завдяки гнучкості елементів інтерфейсу. Це дозволяє помістити створення більшості елементів «Рецепту» в одну функції під один «switch», змінюючи при цьому тільки параметри конструкторів.

Лістинг основного XAML коду елемента «Recipe_Universal» наступний:

```
<DockPanel>
  <local:RecipeBase x:Name="header"/>
  <DockPanel>
    <DockPanel x:Name="containerDelimiter">
      <Label Content="Роздільник"/>
      <ComboBox x:Name="comboBoxDelimiter">
        <ComboBoxItem>Пробіл</ComboBoxItem>
        <ComboBoxItem>Без роздільника</ComboBoxItem>
        <ComboBoxItem>Кома</ComboBoxItem>
      </ComboBox>
    </DockPanel>
  </DockPanel>
```

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		36

```

        <ComboBoxItem>Крапка з комою</ComboBoxItem>
        <ComboBoxItem>Вертикальна лінія</ComboBoxItem>
    </ComboBox>
</DockPanel>
<DockPanel x:Name="containerNumber">
    <Label Content="Байтів на рядок" x:Name="labelNumber"/>
    <local2:NumberPicker x:Name="numberPicker"/>
</DockPanel>
</DockPanel>
</DockPanel>

```

Лістинг основного C# коду елементу «Recipe_Universal» наступний:

```

public partial class Recipe_Universal : UserControl
{
    public string OperationType { get; }

    public Recipe_Universal(string operationName, string
operationType, bool delimiter = false, bool numberPicker = false,
string numberPickLabel = "")
    {
        InitializeComponent();
        header.NameRecipeLabel = operationName;
        OperationType = operationType;
        if (!delimiter) containerDelimiter.Visibility = Visibility.Collapsed;
        if (!numberPicker) containerNumber.Visibility = Visibility.Collapsed;
        labelNumber.Content = numberPickLabel;
    }

    public string GetDelimiter() { return comboBoxDelimiter.Text; }
    public int GetNumber() { return numberPicker.NumValue; }
    //...
}

```

В лістингу вище показаний конструктор класу, який визначає властивості створюваного елементу в залежності від переданих параметрів. Передані параметри визначають заголовок елементу, код «рецепту», та чи буде в інтерфейсі видно елементи роздільника та вибору числа. «Код рецепту», що зберігається у змінній «OperationType» буде важливий для алгоритму обрахунку результату «Рецепту».

Хоч «Recipe_Universal» забезпечує інтерфейсну та функціональну частину для більшості елементів «Рецепту», деякі інші елементи мають зовсім інші потреби. Прикладом елементу інтерфейсу для іншої групи елементів «Рецепту» є «Recipe_Crypto». Він використовується як елемент «Рецепту» для симетричних алгоритмів шифрування. Ці елементи потребують наступних умов функціонування:

– ключ;

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		37

- вектор ініціалізації;
- режим роботи.

Ці потреби значно відрізняються від потреб більшості інших елементів, тому елементи «Рецепту» для функцій симетричного шифрування використовують цей елемент. Приклад доданого елемента «Recipe_Crypto» показаний на рис. 2.6.

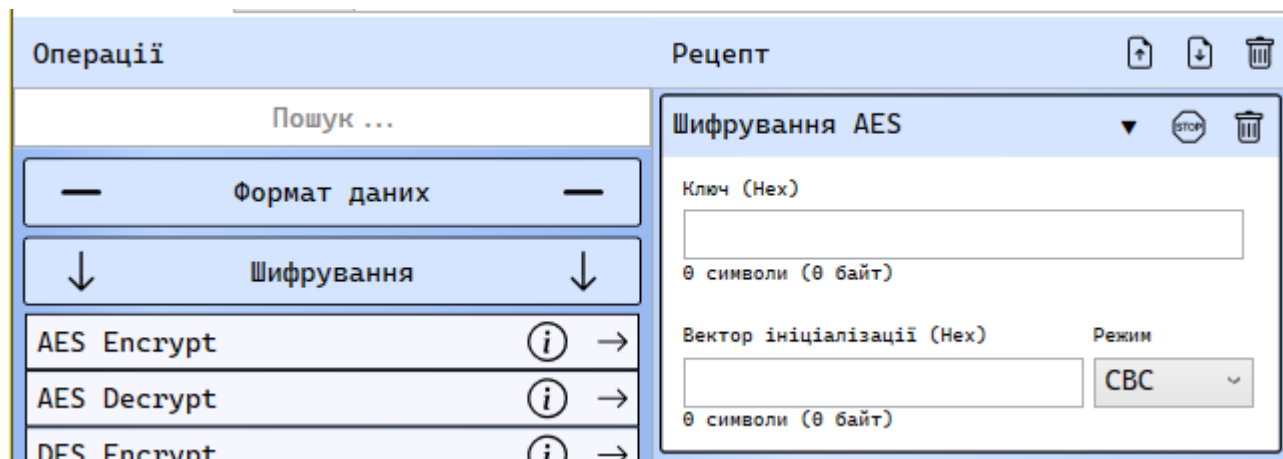


Рисунок 2.6 – Доданий елемент «Шифрування AES» типу «Recipe_Crypto»

Використання окремого елемента для елементів «Рецепту» про шифрування дозволило об'єднати необхідний функціонал для шифрування в одному елементі інтерфейсу. Зокрема, текстові поля всередині цього елемента мають перевірки на правильність вводу даних для «Ключа» та «Вектора ініціалізації». Для програми, введені значення мають бути у шістнадцятковому форматі, проте користувач може ввести будь-що. Тому для цього всередині «Recipe_Crypto» реалізовано перевірку вводу даних, що реалізовано наступним чином.

Лістинг XAML коду текстового поля у «Recipe_Crypto»:

```
<TextBox x:Name="textBoxKey"
    PreviewTextInput="textBoxKey_PreviewTextInput"
    DataObject.Pasting="textBox_Pasting" />
<TextBox x:Name="textBoxIV"
    PreviewTextInput="textBoxKey_PreviewTextInput"
    DataObject.Pasting="textBox_Pasting" />
```

Ці текстові поля при зміні тексту чи вставляння тексту з буфера обміну викликають функції перевірки введення даних, що показано на наступному лістингу:

```
public partial class Recipe_Crypto : UserControl
{
    //...
    private string allowedChars = "0123456789abcdefABCDEF";
```

```

private void textBoxKey_PreviewTextInput(object sender,
TextCompositionEventArgs e)
{
    e.Handled = ValidInput(e.Text);
}

private void textBox_Pasting(object sender,
DataObjectPastingEventArgs e)
{
    string text = (string)e.DataObject.GetData(typeof(String));
    if (ValidInput(text)) e.CancelCommand();
}

private bool ValidInput(string input)
{
    bool handle = true;
    for (int i = 0; i < input.Length; i++)
        if (allowedChars.Contains(input[i]))
        {
            handle = false;
            break;
        }

    return handle;
}
}

```

Функції перевірки тексту при його зміні та перевірки при його вставлянні з буфера обміну перевіряють вміст тексту за допомогою функції «ValidInput». Ця функція дивиться, чи введений текст має в собі дозволені символи, які зберігаються в змінній «allowedChars». При негативному результаті зміни у вмісті текстових полів не відбуваються.

Окрім «Recipe_Crypto», було реалізовано ще ряд елементів інтерфейсу «Рецепту», короткий опис яких приведено в табл. 2.1.

Таблиця 2.1

Короткий опис інших елементів інтерфейсу «Рецепту»

Назва	Для яких операцій	Унікальний вміст
1	2	3
Recipe_Random	Випадкове число, випадковий Nех рядок	Задання меж числа від N до M, задання довжини Nех рядка

1	2	3
Recipe_Vigenere	Шифрування і дешифрування алгоритмом Віженера	Текстовий ключ, вибір алфавіту
Recipe_XOR	XOR операція	Ключ, вибір режиму XOR
Recipe_MorseDecode	Декодування Морзе	Вибір алфавіту результату

Вибір винесення цих операцій в окремі елементи інтерфейсу обґрунтований балансом повторюваності коду та виокремлення функціоналу в окремі елементи інтерфейсу. Зовнішній вигляд елементів із табл. 2.1 показаний на рис. 2.7.

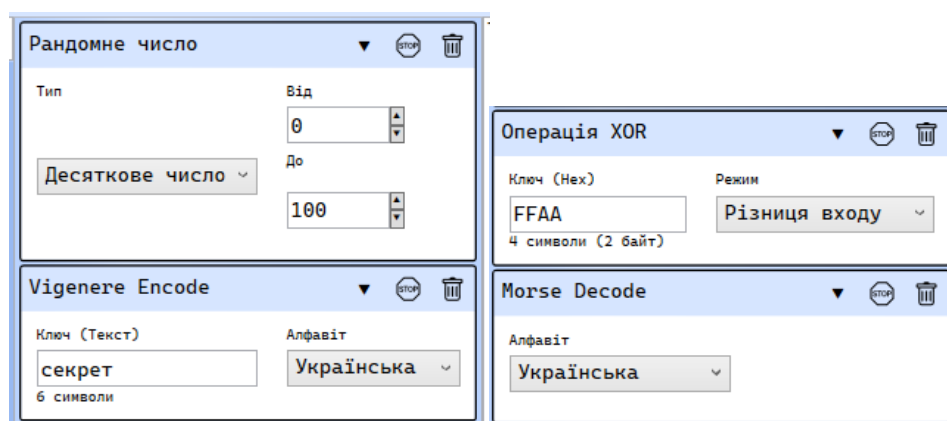


Рисунок 2.7 – Інші реалізовані елементи «Рецепту»

Отже, у користувача є можливість додавання великої кількості різних елементів інтерфейсу до «Рецепту». Наступним кроком є послідовне обрахування операцій над уведеним текстом. Повний лістинг функції обрахування результату наведено в Додатку Б, частина лістингу наступна:

```
private void Button_CalculateRecipe(object sender, RoutedEventArgs e)
{
    string text = TextBoxInput.Text;
    foreach (var item in RecipeList.Items){
        switch (item.ToString()){
            case "Recipe_Universal":
                var itemU = (Recipe_Universal)item;
                switch (itemU.OperationType){
                    case "toHex":
                        text = Calculate_ToHex(text, itemU.GetDelimiter(), itemU.GetNumber());
                        break;
                    case "fromHex":
                        text = Calculate_FromHex(text, itemU.GetDelimiter()); break; //...
                } break;
        }
    }
}
```

```

        case "Recipe_Crypto":
            var itemC = (Recipe_Crypto)item;
            switch (itemC.OperationType){
                case "encaes":
text = Calculate_EncryptAES(text, itemC.Key, itemC.IV, itemC.Mode);
                //... }}}
        TextBoxResult.Text = text;
    }

```

Кожен із реалізованих елементів інтерфейсу «Рецепту» має в собі властивість «OperationType». Вона зберігає «код операції», що використовується для визначення операції, яку треба виконати над текстом. На початку роботи функції дані із текстового поля «TextBoxInput» записуються у змінну, яка буде передаватись в функції операцій. Далі йде основний цикл «foreach», який проходиться по всім елементам списку «Рецепту». Всередині циклу перший «switch» дивиться на тип даних елементу списку «Рецепту», і в залежності від цього перекидує на відповідні частини коду. У випадку «Recipe_Universal», програма далі явно приводить «невідомий» елемент до типу даних «Recipe_Universal». Потім у елементі наступний «switch» переглядає «Код операції», та в залежності від вибору, викликає необхідну функцію обробки тексту. Після проходження всього циклу результат обробки тексту записується у текстове поле результату «TextBoxResult».

Функція «Button_CalculateRecipe» є ключовою в роботі програми. Саме вона створює зв'язок між наповненням списку «Рецепта» і процесом обрахунку результату операцій. Елементи «Рецепту» не зберігають в собі алгоритми перетворення тексту, вони тільки мають «код операції» та необхідні значення для обрахунку, які функція «Button_CalculateRecipe» використовує для виклику функцій обрахунку операцій та передання цих значень.

2.3 Розробка алгоритмів перетворення тексту

Усі функції перетворення тексту мають назву «Calculate_<Назва операції>». Як і «Список операцій» на інтерфейсі, ці поділяються на 4 категорії:

- перетворення кодування;
- шифрування;
- хешування;

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		41

- інші операції.

До першої категорії належать функції, які змінюють представлення тексту на інші алфавіти, залишаючи збережену інформацію. Реалізованими функціями є:

- до шістнадцяткового вигляду і назад;
- до десяткового вигляду і назад;
- до бінарного вигляду і назад;
- з вісімкового вигляду і назад;
- до Base64 і назад.

Для перетворення тексту до шістнадцяткового вигляду в елементі «Рецепту» задається «роздільник» та «кількість байт на рядок». Ці дані передаються у функцію, лістинг якої наступний:

```
private string Calculate_ToHex(string text, char d, int l)
{
    string res = "";
    if (l < 0) l = 0;
    byte[] b = textToByte(text);

    for (int i = 0, li = 1; i < b.Length; i++, li++)
    {
        res += b[i].ToString("X2");
        if (d != 'X' && i != b.Length - 1) res += d;
        if (li == l)
        {
            res += '\n';
            li = 0;
        }
    }
    return res;
}
```

Результат функції формується у змінній «res», яка оголошується на початку. Далі йде перевірка переданої кількості байт на рядок у змінній «l», так як значення має бути тільки позитивним. Вхідні текстові дані, які передані у змінній «text», перетворюються на масив типу «byte», так як перетворення кодування легше робити над таким типом даних. В циклі функції оголошується два ітератора: один для перерахунку текстових даних, інший для підрахунку поточної доданої кількості байт в рядку. При кожному циклі до змінної результату додається перетворений поточ-

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		42

ний байт інформації до шістнадцяткового вигляду. Наступна логіка всередині циклу необхідна для додання роздільника та переходу на новий рядок, якщо така властивість результату була задана при виклику функції.

Обернено до функціоналу перетворення на шістнадцятковий вигляд, реалізовано перетворення шістнадцяткового вигляду до тексту. Проте таке перетворення вимагає набагато більше перевірок та умов, так як неправильні данні викликають помилку в роботі програми. Лістинг функції перетворення шістнадцяткових даних на текст наступний:

```
private string Calculate_FromHex(string text, char d){
    string[] substrings = { };

    if (d == 'X'){
        for (int i = 0; i < text.Length / 2; i++){
            string hexCh = "";
            hexCh += text[i * 2]; hexCh += text[i * 2 + 1];
            substrings = substrings.Append(hexCh).ToArray();
        }
    }
    else {
        if (!text.Contains(d))
            return $"Помилка: текст не має роздільника '{d}'";
        substrings = text.Split(d);
    }

    byte[] b = new byte[substrings.Length];

    try
        for (int i = 0; i < substrings.Length; i++)
            b[i] = byte.Parse(substrings[i], NumberStyles.HexNumber);
    catch (System.OverflowException)
        return "Текст не є формату Hex";

    return byteToText(b);
}
```

Рядок текстових даних у форматі шістнадцяткових символів може мати роздільник, а може не мати. Саме це перевіряє перший «if», так як відсутність роздільника вимагає інший алгоритм перетворення формату. У випадку відсутності роздільника необхідно проходити кожну пару символів, та виокремлювати її у окремі слова. При наявності роздільника є можливість використання вбудованої функції

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				43
Змн.	Арк.	№ докум.	Підпис	Дата		

мови С# «Split()», яка повертає список слів, які розділяються із рядка з використанням переданого як параметр роздільника. Далі, маючи готовий список виокремлених слів, вони перетворюються на тип даних «byte». В кінці роботи функції результат повертається через функцію, яка перетворює список байтів на текстовий рядок.

Іншими реалізованими перетвореннями кодування, а саме десяткового, бінарного, вісімкового мають відповідно схожу реалізацію з використанням інших методів для перетворення одного виду тексту на інший. Основна ідея алгоритму перетворення кодування – це перетворення тексту до типу «byte», з якого вже легко отримувати необхідний вигляд даних.

Другою групою реалізованих операцій є функції симетричного шифрування і дешифрування. Вони в своєму функціоналі схожі, змінюється лише тип шифрування. Тому для них було створено універсальну функцію шифрування, яка отримує як параметр необхідний тип шифрування та іншу параметри, які необхідні для операції. Лістинг цієї функції наступний:

```
private string EncryptorSymmertic(SymmetricAlgorithm algorithm, byte[]
key, byte[] iv, CipherMode mode, string text)
{
    string result = string.Empty;
    try{
        using (algorithm){
            algorithm.Key = key; algorithm.IV = iv; algorithm.Mode = mode;
            using (var encryptor =
algorithm.CreateEncryptor(algorithm.Key, algorithm.IV))
                using (var ms = new MemoryStream()){
                    using (var cs =
                        new CryptoStream(ms, encryptor, CryptoStreamMode.Write))
                        using (var sw = new StreamWriter(cs)){
                            sw.Write(text);
                        }
                    result = byteToHashResult(ms.ToArray());
                }
            }
        }
    }
    catch (CryptographicException e)
        result = "Помилка шифрування: " + e.Message;
    return result;}
```

В якості параметрів у функцію передається алгоритм шифрування, ключ, вектор ініціалізації, режим шифрування та сам текст, який буде шифруватись. Процес шифрування обгорнутий у конструкцію «try-catch» на випадок непередбачуваної

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				44
Змн.	Арк.	№ докум.	Підпис	Дата		

помилки в процесі шифрування. Далі програма починає використовувати переданий алгоритм, задаючи йому параметру ключа, вектора ініціалізації та режиму роботи. Можливі режими роботи, передбачені програмою: CBC, ECB, CFB. Далі створюються необхідні об'єкти для процесу шифрування:

- шифрувальник, який приймає ключ та вектор ініціалізації;
- загальний потік інформації MemoryStream;
- потік криптографічної інформації, який приймає загальний потік інформації, шифрувальник та режим потоку шифрування, що дорівнює «Запису» в цьому випадку;
- потік запису, який записує інформацію.

В результаті дані шифруються, та записуються у змінній «result» у шістнадцятковому форматі, яка потім повертається як результат виконання функції.

Універсальна функція розшифрування «DecryptorSymmertic» має схожу конструкцію, проте відрізняється у створюваних об'єктах та деяких параметрах:

- замість шифрувальника створює дешифрувальник;
- потік криптографічної інформації ставиться в режим «Читання»;
- замість потоку запису створюється потік читання.

Лістинг частини універсальної функції дешифрування «DecryptorSymmertic», які відрізняються від функції шифрування, наступний:

```
private string DecryptorSymmertic(SymmetricAlgorithm algorithm, byte[]
key, byte[] iv, CipherMode mode, string text)
{
//...
using (var decryptor =
    algorithm.CreateDecryptor(algorithm.Key, algorithm.IV))
    using (var ms = new MemoryStream(stringHexToByte(text))) {
        using (var cs =
new CryptoStream(ms, decryptor, CryptoStreamMode.Read))
            using (var sr = new StreamReader(cs))
            {
                result = sr.ReadToEnd();
            }
        }
    }
}
```

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				45
Змн.	Арк.	№ докум.	Підпис	Дата		

Використовують ці універсальні функції реалізовані алгоритми шифрування AES, DES, 3DES та RC2. Приклад використання універсальної функції шифрування для виконання алгоритму DES має наступний лістинг:

```
private string Calculate_EncryptDES(string text, string key, string
IV, CipherMode mode)
{
    byte[] key_b = { };
    byte[] iv_b = { };

    if (key.Length == 8 * 2) key_b = stringHexToByte(key);
    else return "Помилка: ключ має мати довжину 8 байт";

    if (IV.Length == 8 * 2) iv_b = stringHexToByte(IV);
    else return "Помилка: вектор ініціалізації має мати довжину 8
байт";

    return EncryptorSymmertic(DES.Create(), key_b, iv_b, mode, text);
}
```

Функція «Calculate_EncryptDES» має на меті перевірити передані параметри на придатність до використання алгоритмом DES, перетворити їх на необхідний тип даних, та виконати універсальну шифрувальну функцію, який передано алгоритм DES за допомогою коду «DES.Create()». У випадку алгоритма DES, ключ і вектор ініціалізації мають мати довжину у 8 байт, або 64 біт, що у перевіряє ця функція.

Функція розшифрування DES має аналогічну структуру та перевірку на правильність уведених даних, має лістинг:

```
private string Calculate_DecryptDES(string text, string key, string
IV, CipherMode mode)
{
    //...

    return DecryptorSymmertic(DES.Create(), key_b, iv_b, mode, text);
}
```

Інші реалізовані алгоритми шифрування, а саме AES, 3DES, RC2, зроблені схожим чином, але мають свої власні перевірки правильності введених даних. Даних підхід зменшує повторюваність коду, та створює можливості для легкого розширення функціоналу програми на випадок реалізації інших алгоритмів симетрич-

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		46

ного шифрування. Така реалізація стає можливою через особливості об'єктно орієнтованого програмування можливо, а саме що об'єкти симетричного шифрування всі походять від батьківського класу «SymmetricAlgorithm».

Схожий підхід було застосовано для реалізації третьої групи операцій – хешування. Реалізовані функції, а саме MD5, RIPEMD160, SHA1, SHA256, SHA384, SHA512, походять від батьківського класу «HashAlgorithm», через що було створено універсальну функцію, яка виконувала операцію хешування в залежності від переданого алгоритму, і її лістинг наступний:

```
private string Calculate_Hash(string text, HashAlgorithm algorithm)
{
    byte[] byteHash = algorithm.ComputeHash(textToByte(text));
    return byteToHashResult(byteHash);
}
```

Для виконання операції хешування особливих перевірок на правильність вводу даних не потрібно, тому реалізація є доволі простою. Використання цієї універсальної функції шифрування відбувається наступним чином:

```
private string Calculate_HashMD5(string text)
    => Calculate_Hash(text, MD5.Create());

private string Calculate_HashRIPEMD160(string text)
    => Calculate_Hash(text, RIPEMD160.Create());
```

Приклад вище наводить дві функції, яку обраховують хеші MD5 та RIPEMD160. Мета цих функцій – створення об'єкту, який зберігає в собі алгоритми хешування та виклик універсальної хеш функції, при цьому зберігаючи правило проектування цієї програми щодо іменування функцій обрахунку операцій.

Останньою групою функцій перетворення є категорія «Інше». Кожна реалізована тут функція вимагала особливого підходу, та розробку власного алгоритму або навіть елементу інтерфейсу. Ця група функцій включає:

- генерація випадкового числа/рядка;
- алгоритм Цезаря для алфавіту/ASCII/UTF8;
- кодування і розкодування Морзе;
- шифрування і розшифрування Віженера;
- операція XOR;

Генерація випадкового числа реалізована наступним чином:

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		47

```
private string Calculate_RandomNumber(int min, int max){
    Random r = new Random();
    if (min > max)
    {
        int temp = min;
        min = max;
        max = temp;
    }
    return r.Next(min, max).ToString();
}
```

Функція генерування випадкового рядка створює рядок заданої довжини у шістнадцятковому форматі. Така функція необхідна для генерації ключів для алгоритмів симетричного шифрування. Лістинг наступний:

```
private string Calculate_RandomHex(int length)
{
    Random r = new Random();
    string res = "";
    for (int i = 0; i < length; i++)
        res += r.Next(16).ToString("X");
    return res;
}
```

Для алгоритму Цезаря було створено три реалізації. Перша реалізація є стандартною, та переносить символ вздовж алфавіту. Вона замінює символи українського та англійського алфавіту та зберігає відповідні великі та малі літери.

Лістинг функції для обрахунку алфавітного Цезаря наступний:

```
private string Calculate_ROT13(string text, int step)
{
    string res = "";
    string alphabetua = "абвггдеежзиіійклмнопрстуфхцчшщьюя";
    string alphabetUA = "АБВГГДЕЄЖЗИІІЙКЛМНОПРСТУФХЦЧШЩЬЮЯ";
    string alphabeteng = "abcdefghijklmnopqrstuvwxyz";
    string alphabetENG = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    for (int i = 0; i < text.Length; i++)
    {
        if (alphabetua.Contains(text[i]))
            res += rotateByStep(alphabetua, text[i], step);
        else if (alphabetUA.Contains(text[i]))
            res += rotateByStep(alphabetUA, text[i], step);
        else if (alphabeteng.Contains(text[i]))
            res += rotateByStep(alphabeteng, text[i], step);
        else if (alphabetENG.Contains(text[i]))
            res += rotateByStep(alphabetENG, text[i], step);
        else
            res += text[i];
    }
    return res;
}
```

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		48

}

Функція отримує параметрами текст та крок для зміщення. Далі вона оголошує алфавіти для двох мов у великому та малому регістрі. Всередині циклу, яких проходить по тексту, визначається належність до визначених алфавітів поточної літери, і у випадку належності здійснюється заміна на визначену кількість кроків за допомогою функції «rotateByStep». У випадку неналежності до оголошених алфавітів функція зберігає поточний символ у незміненому вигляді.

Функція «Calculate_ROT13» використовує функцію «rotateByStep» для визначення нового символу із заданим кроком, і її лістинг наступний:

```
private char rotateByStep(string alphabet, char oldCh, int step)
{
    char c;
    int pos = GetPos(alphabet, oldCh);
    if (pos + step >= 0)
        c = alphabet[(pos + step) % alphabet.Length];
    else
        c = alphabet[alphabet.Length + ((pos + step) % alphabet.Length) - 1];
    return c;
}
```

Функція «rotateByStep» приймає параметрами алфавіт, поточну літеру та крок зміщення. Спочатку вона визначає позицію поточної літери та записує її в оголошену змінну «pos». Далі, в залежності від того, чи результат додавання поточної позиції і кроку позитивним чи негативним числом, застосовує різні формули для обрахунку позиції. Це необхідно, так як в мові програмування оператор «%», який відповідає операції знаходження остачі від ділення, зберігає знак мінусу при застосуванні до від'ємного числа. Таким чином, ця функція може зробити перестановку на будь-яке позитивне чи негативне число позицій.

Для виконання шифрування Цезаря вздовж алфавіту ASCII використовується функція з наступним лістингом:

```
private string Calculate_ROT47(string text, int step)
{
    byte[] b = Encoding.ASCII.GetBytes(text);
    byte bstep = (byte)step;
    for (int i = 0; i < b.Length; i++)
        b[i] = (byte)(b[i] + bstep);
    return new string(Encoding.ASCII.GetChars(b));
}
```

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				49
Змн.	Арк.	№ докум.	Підпис	Дата		

В кодуванні ASCII число від 0 до 255 відповідає певній літері, і саме вздовж цього кодування здійснюється переміщення поточного символу. З схожим принципом працює функція шифрування Цезаря для кодування UTF8 з наступним лістингом:

```
private string Calculate_ROT8000(string text, int step)
{
    char[] chars =
Encoding.UTF8.GetChars(Encoding.UTF8.GetBytes(text));
    for (int i = 0; i < chars.Length; i++)
        chars[i] = (char)(chars[i] + step);
    return new string(chars);
}
```

Наступним реалізованою операцією є кодування Морзе. Функція перетворює українські та англійські тексти на послідовність крапок і тире, при цьому пропускаючи інші символи при кодуванні. Лістинг функції кодування Морзе:

```
private string Calculate_MorseEncode(string text)
{
    string result = string.Empty;
    var MorseEncodeMap = GetMorseEncodingDictionary();

    for (int i = 0; i < text.Length; i++)
        if (char.IsLetter(text[i])){
            char ch = char.ToLower(text[i]);
            if (MorseEncodeMap.ContainsKey(ch)){
                result += MorseEncodeMap[ch];
                result += ' ';
            }
        }
    return result;
}
```

На початку роботи функції в змінну «MorseEncodeMap» отримується словник, який зберігає відповідність літер українського і англійського алфавіту кодуванням Морзе у вигляді «a = .-». Далі функція проходить по всьому тексту, та записує результат перетворення літер отриманого тексту у змінну «result», при цьому пропускаючи символи, які не кодуються.

Наступною реалізованою операцією є шифрування Віженера. Для шифрування і дешифрування українського і англійського алфавіту було створено універсальну функцію, яка може шифрувати та дешифрувати по Віженеру будь-який алфавіт, який передано як параметр. Лістинг цієї функції:

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				50
Змн.	Арк.	№ докум.	Підпис	Дата		

```
private string VigenereEncode(string text, string key, string
alphabet, int side){
    if (key.Length == 0) return text;
    string result = "";
    text = text.ToLower();
    for (int i = 0, j = 0; i < text.Length; i++){
        char c = text[i];
        if (alphabet.Contains(text[i])){
            int newPos = GetPos(alphabet, text[i]) +
                side * GetPos(alphabet, key[j++ % key.Length]);
            c = alphabet[(alphabet.Length + newPos) % alphabet.Length];
        }
        result += c;
    }
    return result;
}
```

Функція «VigenereEncode» приймає параметрами текст, ключ, алфавіт мови шифрування та напрямок шифрування. У циклі, який проходиться переданим тестом, перевіряється відповідність поточної літери до алфавіту шифрування, і у позитивному випадку здійснюється перестановка літери на певну кількість позицій, що залежить від ключа.

Завдяки тому, що ця функція є універсальною, її використання є простим, та залежить від переданих параметрів. Наприклад, лістинг функції шифрування Віженером українським алфавітом:

```
private string Calculate_VigenereEncodeUA(string text, string key)
{
    string alphabet = "абвгґдеежзиіїйклмнопрстуфхцщщя";
    return VigenereEncode(text, key, alphabet, 1);
}
```

Функція «Calculate_VigenereEncodeUA» приймає текст та ключ, і викликає функцію «VigenereEncode», передаючи в неї додатково оголошений алфавіт та напрямок шифрування. Аналогічно було створено функції дешифрування українським алфавітом, та шифрування і дешифрування англійським алфавітом. Таке проєктування дозволяє зберігати правило проєктування цієї програми щодо іменування функцій обрахунку операцій.

Останньою реалізованою операцією є XOR. Програма може виконувати шифрування цим методом, використовуючи три режими. Проте на початку, виклика-

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		51

ється основна функція обрахунку XOR «Calculate_XOR», яка дивиться на переданий режим роботи, та викликає необхідну функцію з потрібним алгоритмом, результат яких передає назад відповідь. Лістинг цієї функції наступний:

```
private string Calculate_XOR(string text, string key, int mode)
{
    if (key.Length == 0) return text;
    string result = "";
    byte[] key_b = stringHexToByte(key);
    switch (mode) {
        case 0: result = XORnormal(text, key_b); break;
        case 1: result = XORinputDiff(text, key_b); break;
        case 2: result = XORoutputDiff(text, key_b); break; }
    return result;
}
```

Ця функція перетворює переданий ключ у текстовому вигляді до типу даних «byte». Далі вона в залежності від переданого числа «mode» викликає наступні режими шифрування XOR:

- XOR звичайний;
- XOR різниця входу;
- XOR різниця виходу.

Перший режим є звичайним, де текст розділяється на блоки розміром з ключ, та проходять процес шифрування незалежно один від одного. Цей режим схожий на режим ECB алгоритмів симетричного шифрування. Функція шифрування звичайним XOR має лістинг:

```
private string XORnormal(string text, byte[] key){
    string res = "";
    for (int i = 0; i < text.Length; i++){
        res += (char)((byte)text[i] ^ key[i % key.Length]);
    }
    return res;
}
```

У другому режимі «різниця входу» ключем стає відкритий текст попереднього раунда. Це створює лавинний ефект при шифруванні, через що результат наступних блоків залежить від відкритого тексту до цього моменту. Функція шифрування XOR з різницею входу має лістинг:

```
private string XORinputDiff(string text, byte[] key){
    string res = "";
    for (int i = 0; i < text.Length; i++){
        res += (char)((byte)text[i] ^ key[i % key.Length]);
    }
    return res;
}
```

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				52
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        key[i % key.Length] = (byte)text[i];
    }
    return res;
}

```

У другому режимі «різниця виходу» ключем стає зашифрований текст попереднього раунда. Це створює ефект схожий на попередній режим. Функція шифрування XOR з різницею виходу має лістинг:

```

private string XORoutputDiff(string text, byte[] key){
    string res = "";
    for (int i = 0; i < text.Length; i++) {
        byte processed = (byte)((byte)text[i] ^ key[i % key.Length]);
        res += (char)(processed);
        key[i % key.Length] = processed;
    }
    return res;
}

```

Отже, було розглянуто проектування алгоритмів перетворення тексту для першого модуля програми. Для другого модулю далі буде показано процес розробки алгоритмів для аналізу тексту.

2.4 Розробка алгоритмів аналізу тексту

Програмне рішення має виконувати методи аналізу тексту:

- статистичний;
- структурний;
- лінгвістичний.

Статистичний метод включає розмір тексту, ентропію та частотний аналіз. Щоб обраховувати розмір тексту і ентропію, на текстових полях вводу і виводу інформації створюються події зміни тексту, які викликаються кожного разу, як текст зазнає змін. Ці події викликають функції обрахування статистики, яка виводить інформацію на інтерфейс користувача. Програмне рішення має у своєму складі три текстові поля, в кожному з яких окремо обраховується статистика довжини тексту і ентропія. Приклад обрахунку статистики буде показано на текстовому полі із модуля аналізу тексту. Першим кроком є створення виклику функції обрахунку статистики через подію зміни тексту, що показано на лістингу XAML коду:

```
<RichTextBox x:Name="textBoxAnalys"
```

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		53

TextChanged="textBoxAnalys_TextChanged"/>

При будь-якій зміні тексту викликається функція, лістинг якої наступний:

```
private void textBoxAnalys_TextChanged(object sender,
TextChangedEventArgs e){
    string text = GetTextAnalys();
    labelStatInputAnalysLetters.Content = $"Символів: {text.Length}";
    labelStatInputAnalysLines.Content = $"Рядків: {text.Split(new[] {
"\r\n", "\r", "\n" }, StringSplitOptions.None).Length}";

    double entropy = CalculateEntropy(text);
    labelEntropyAnalys.Content = entropy.ToString("F2");
    progressBarEntropyAnalys.Value = entropy;
    progressBarEntropyAnalys.Foreground = GetEntropyColor(entropy);
}
```

Перша частина функції обраховує кількість символів і рядків у введеному тексті, та виводить результат на інтерфейс користувача. Друга частина обраховує ентропію через функцію «CalculateEntropy», та виводить її результат на інтерфейс користувача в якості числа, та елементу «progressBar», який слугує візуальним показником для значення обрахованої ентропії. «progressBar» в залежності від числа ентропії змінює свій колір, який обраховується функцією «GetEntropyColor», та розмір частини елементу, який заповнений кольором.

Лістинг коду XAML для «progressBar», яке показує результат обрахування ентропії на інтерфейсі наступний:

```
<DockPanel>
    <Label Content="Ентропія"/>
    <Grid Margin="5">
        <ProgressBar x:Name="progressBarEntropyAnalys" Minimum="0"
Maximum="10" Orientation="Vertical" />
        <Label x:Name="labelEntropyAnalys" Content="0,00"
Background="White"/>
    </Grid>
</DockPanel>
```

Для обрахунку числового значення ентропії використовується функція з наступним лістингом:

```
private double CalculateEntropy(string input){
    if (string.IsNullOrEmpty(input)) return 0.0;
    var frequencies = input
        .GroupBy(c => c)
        .ToDictionary(g => g.Key, g => g.Count());

    int length = input.Length;
    double entropy = 0.0;
```

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		54

```

foreach (var freq in frequencies.Values){
    double p = (double)freq / length;
    entropy -= p * Math.Log(p, 2);
}
return entropy;
}

```

Функція обраховує кількість появи кожного символу у тексті, та групує їх у «Словник», де кожній літері відповідає її кількість появи у тексті. Далі, на основі довжини тексту і кількості появи кожної літери, обраховується ентропія за формулою Шенона.

Далі числове значення ентропії необхідно перетворити на колір для «progressBar». Колір має бути найбільш зелений для значень 3.5 – 5, і чим далі від цих значень, тим більш червоний. Так як колір складається з трьох каналів кольору – червоний, зелений, синій, і їх значення має бути від 0 до 255, можливо створити формули обрахунку каналів на основі ентропії. В результаті для трьох каналів кольору були створені формули:

$$R = \min\left(255 * \left(\frac{entropy}{2} - 2.25\right)^4, 255\right),$$

де R = значення червоного каналу;

$$G = 255 * \max\left(0, 1 - \left(\frac{entropy}{3.25} - 1.38\right)^4\right),$$

де G = значення зеленого каналу.

Значення синього каналу завжди дорівнює 0.

Графіки формул для каналів кольору при значеннях до 0 до 10 показані на рис. 2.8.

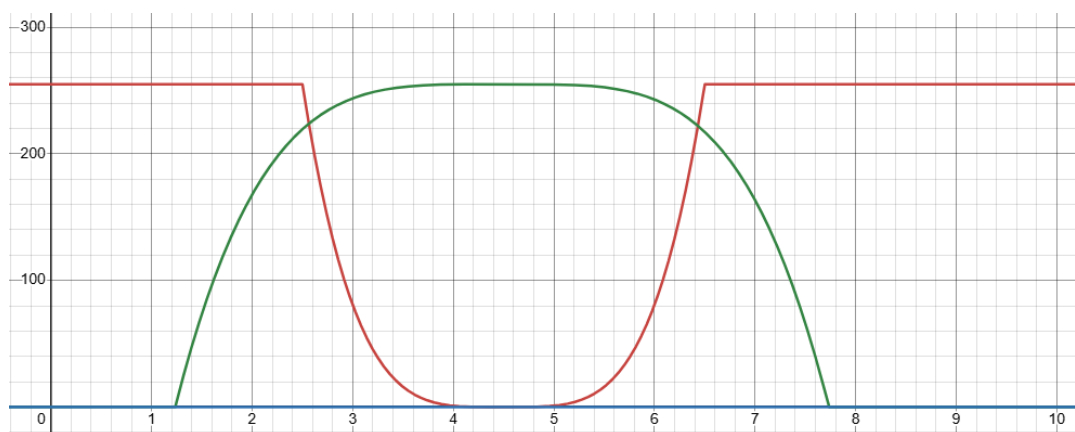


Рисунок 2.8 – Графіки каналів кольору

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				55
Змн.	Арк.	№ докум.	Підпис	Дата		

Використовуючи зазначені формули, при різних значеннях ентропії «progressBar» набуває виглядів, які показано на рис. 2.9, тим самим досягаючи ефект плавного показу зеленішого кольору для значення ентропії природньої мови.

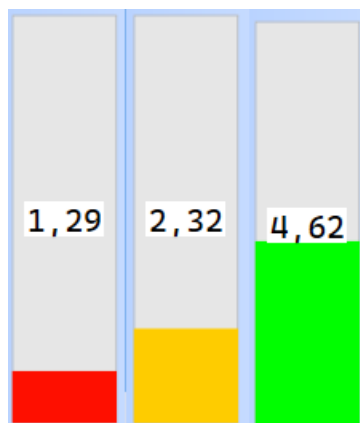


Рисунок 2.9 – Варіанти «progressBar»

Для показу результатів частотного аналізу було створено окремий елемент інтерфейсу «MyHistogram», який приймає «словник» розподілу літер, та на його основі створює графічну статистику у вигляді гістограми. Лістинг основного функціоналу функції отримання даних елементу «MyHistogram» наступний:

```
public void SetData(Dictionary<char, int> values, double width = -1)
{
    if (values.Count == 0) return;
    int max = values.Values.Max();
    int count = values.Values.Sum();
    SetLabelPercentValues(max, count);
    SetLabelCountValues(max);
    foreach (var item in values){
        Label label = new Label();
        label.Content = item.Key;
        ProgressBar progressBar = new ProgressBar();
        progressBar.Maximum = max;
        progressBar.Value = item.Value;
        panelProgress.Children.Add(progressBar);
        panelLabel.Children.Add(label);
    }
}
```

В основному коді програми, елемент «MyHistogram» створюється таким чином, як показано на лістингу частини функції створення статистики:

```
private void buttonAddHistogram_Click(object sender, RoutedEventArgs e){
    string text = GetTextAnalys().ToLower();
    if (text == "") return;
    /**
```

```

var filteredCount = new Dictionary<char, int>();
/**
MyHistogram mh = new MyHistogram();
mh.SetData(filteredCount);
panelForHistogram.Children.Clear();
panelForHistogram.Children.Add(mh);
}

```

Функція «buttonAddHistogram_Click» спрацьовує при натисканні користувачем кнопки «Зробити гістограму», і в залежності від вибраних алфавітів, які треба обраховувати, обраховує частоту появи символів та впорядковує їх за алфавітом. Далі цей результат передається у об'єкт «MyHistogram», який створюється на інтерфейсі користувача. В результаті, створюється об'єкт інтерфейсу, який показано на рис. 2.10.

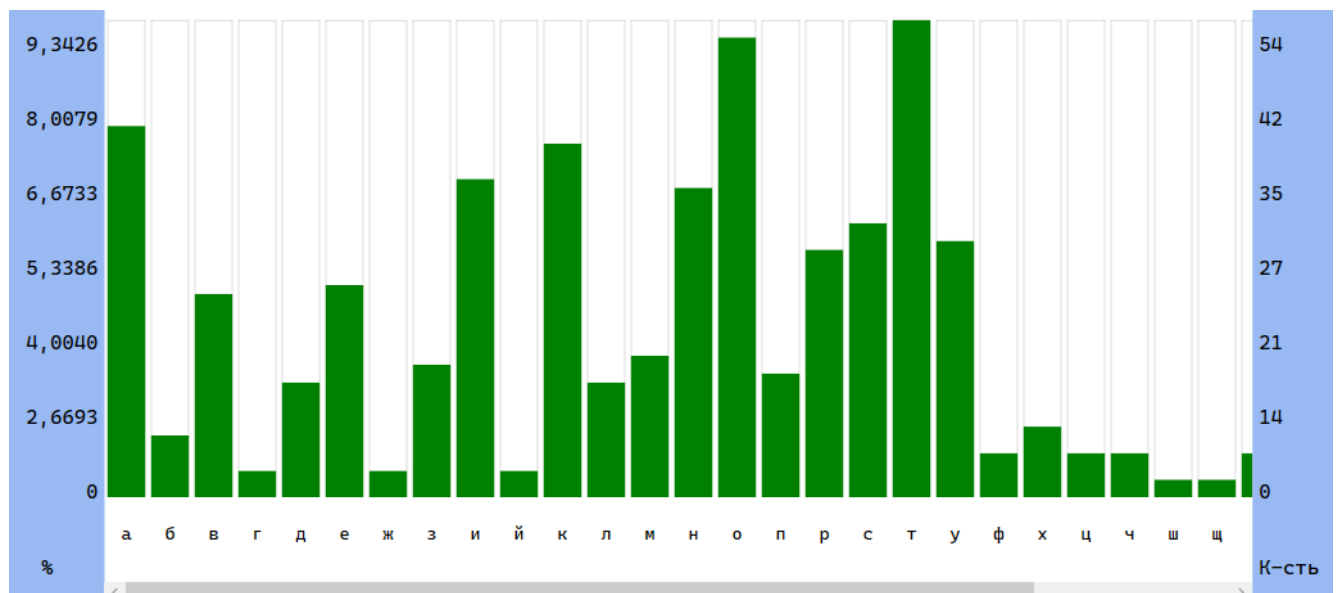


Рисунок 2.10 – Елемент «MyHistogram»

Для реалізації структурного аналізу було створено функціонал обрахунку повторів у введеному тексті. Лістинг функції обрахування повторів наступний:

```

private void ButtonCountRepeats_Click(object sender, RoutedEventArgs e){
var SearchResult = new Dictionary<string, int>();
string text = GetTextAnalys();
int size = numpickRepeat.NumValue;
for (int i = 0; i < text.Length - size; i++){
string searchText = text.Substring(i + size);
string searchStr = text.Substring(i, size);
if (SearchResult.ContainsKey(searchStr)) continue;
string repeat = string.Empty;
int count = 0;
while (searchText.Contains(searchStr)){

```

```

        count = CountSubstr(searchText, searchStr);
        repeat = searchStr;
        searchStr += text[i + searchStr.Length];
    }
    if (count > 0 && !SearchResult.ContainsKey(repeat))
        SearchResult[repeat] = count + 1;
}
/**
}

```

В якості початкових даних функція отримує текст та мінімальну довжину для пошуку повторів. В головному циклі текст розбивається на дві частини: текст для пошуку повтору і підрядок, який шукають. Далі робиться перевірка чи для цього підрядка вже був здійснений пошук, і якщо так – то алгоритм переходить на наступний цикл. Другий цикл «while» збільшує розмір підрядка до тих пір, доки є результати пошуку повторів. В результаті у словнику «SearchResult» записуються повтори та їх кількість в тексті.

В другій частині цієї функції здійснюється виведення результатів на інтерфейс користувача, як показано в наступному лістингу:

```

private void ButtonCountRepeats_Click(object sender, RoutedEventArgs e){
    /**
    Random r = new Random();
    SearchResult = SearchResult.OrderByDescending(
        x => x.Value).ToDictionary(x => x.Key, x => x.Value);
    foreach (var item in SearchResult){
        byte Base = 0xA0;
        byte R = (byte)(Base + r.Next(0xFF - Base));
        byte G = (byte)(Base + r.Next(0xFF - Base));
        byte B = (byte)(Base + r.Next(0xFF - Base));
        var ws =
            new WordStat(item.Key, Color.FromRgb(R, G, B), item.Value);
        listRepeats.Items.Add(ws);
    }
}

```

Спочатку результати пошуку сортуються по числу кількості знайдених повторів. Далі на основі цього список повторів «listRepeats» наповнюються елементами списку, які відображають повторюваний текст, його колір відображення в текстовому полі, довжину повтору та кількість повторів. Зовнішній вигляд цього списку показаний на рисунку 2.11.

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				58
Змн.	Арк.	№ докум.	Підпис	Дата		

Повторення			
Виявлене повторення	Колір	Довжина	К-сть
текст		5	6
екст		4	6
татисти		7	5
атисти		6	5
тисти		5	5

Рисунок 2.11 – Список повторів

Модуль аналізу тексту також має реалізований функціонал лінгвістичного аналізу. Для цього використовується функція з наступним лістингом:

```
private void buttonCalculateAlphabets_Click(object sender,
RoutedEventArgs e){
    string text = GetTextAnalys().ToLower();
    if (text == "") return;
    var counts = text.GroupBy(c => c)
        .ToDictionary(g => g.Key, g => g.Count());
    string alphabetua = "абвгґдєжзиіїйклмнопрстуфхцщья",
        alphabeteng = "abcdefghijklmnopqrstuvwxyz";
    int countUkr = 0, countEng = 0;
    foreach (var item in counts)
        if (alphabetua.Contains(item.Key)) countUkr += item.Value;
        else if (alphabeteng.Contains(item.Key)) countEng += item.Value;
    int totalCount = countUkr + countEng;
    double resUkr = (float)countUkr * 100 / totalCount,
        resEng = (float)countEng * 100 / totalCount;
    labelAlphabetEng.Content = $"{resEng:F2}% Англійський";
    labelAlphabetUkr.Content = $"{resUkr:F2}% Український";
    progressAlphabetEng.Value = resEng;
    progressAlphabetUkr.Value = resUkr;
}
```

Функція підраховує кількість символів кожного алфавіту, і в представленій частині лістингу це показано на прикладі українського і англійського алфавіту. Повна версія також рахує пробіли, спеціальні символи, числа та інші знаки. Результат обрахунку показується на інтерфейсі у вигляді тесту та гістограм, зроблених через «progressBar». Вивід результату в програмі має вигляд, показаний на рис. 2.12.

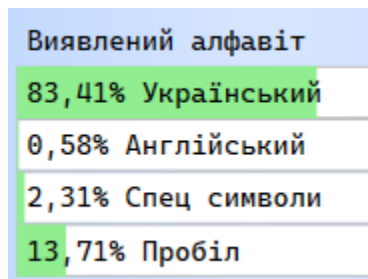


Рисунок 2.13 – Вивід лінгвістичного аналізу

Висновки до другого розділу

В другому розділі було показано процес проектування зовнішнього вигляду програмної реалізації та принцип роботи загального алгоритму програми, що дозволило створити умови для зручної реалізації великої кількості функцій перетворення тексту.

Було створено програмну реалізацію методів перетворення та аналізу тексту, що в результаті дало готове програмне рішення для вирішення поставлених задач, розглянутих в першому розділі.

РОЗДІЛ 3 ТЕСТУВАННЯ РОБОТИ ПРОГРАМНОГО РІШЕННЯ

3.1 Огляд та тестування модуля перетворення тексту

При запуску програмного рішення користувача зустрічає інтерфейс, який показано на рис. 3.1.

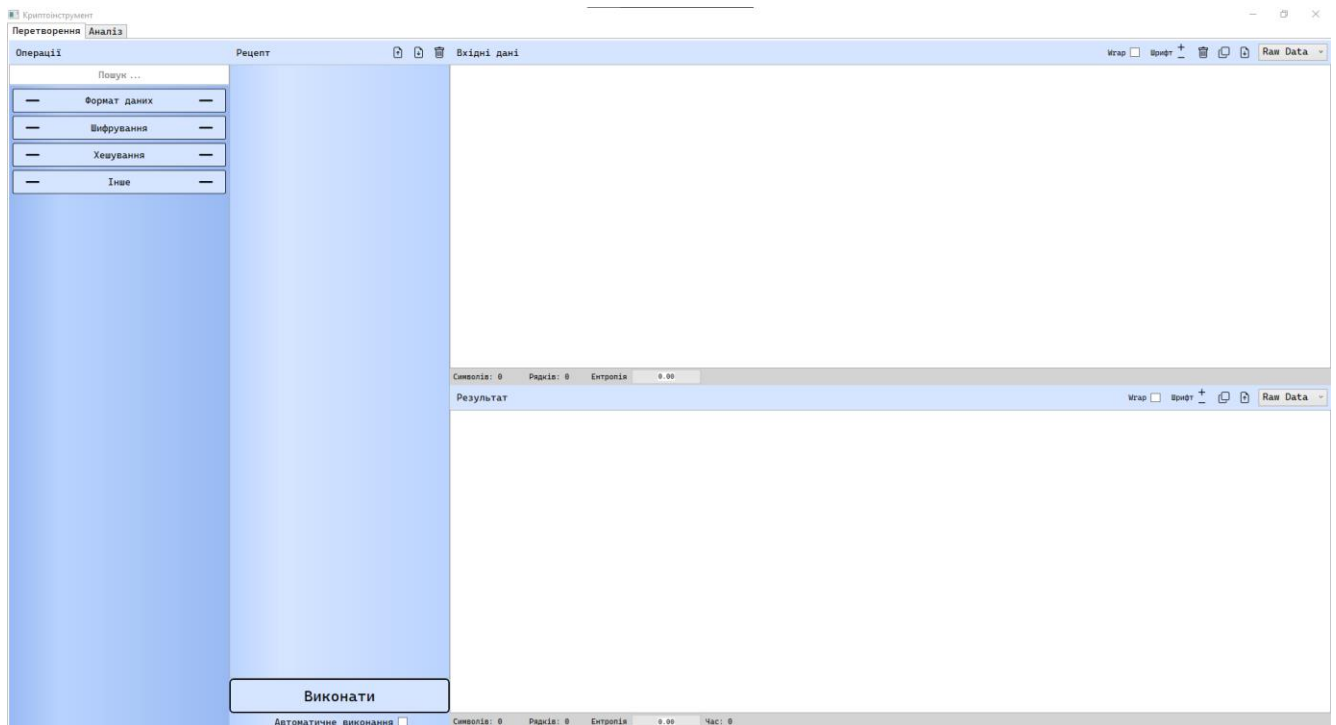


Рисунок 3.1 – Інтерфейс програми

Програма складається з двох модулів, які можна перемикаючи натискаючи кнопки «Перетворення» і «Аналіз» у верхній частині екрану. Наразі в програмі вибраний модуль «Перетворення». Інтерфейс цього модуля складається з 4 частин:

- «Операції» - містить список операцій над текстом;
- «Рецепт» - показує поточний алгоритм обробки тексту;
- «Вхідні дані» - поле для вводу тексту;
- «Результат» - поле для показу результату перетворення.

Елемент «Операції» містить список, який складається з 4 частин:

- «Формат даних»;
- «Шифрування»;
- «Хешування»;
- «Інше».

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		61

Щоб переглянути кожен з цих списків, необхідно натиснути клавішою миші на панель з назвою списку, який цікавить. Результат натиснення на список «Шифрування» показаний на рис. 3.2.

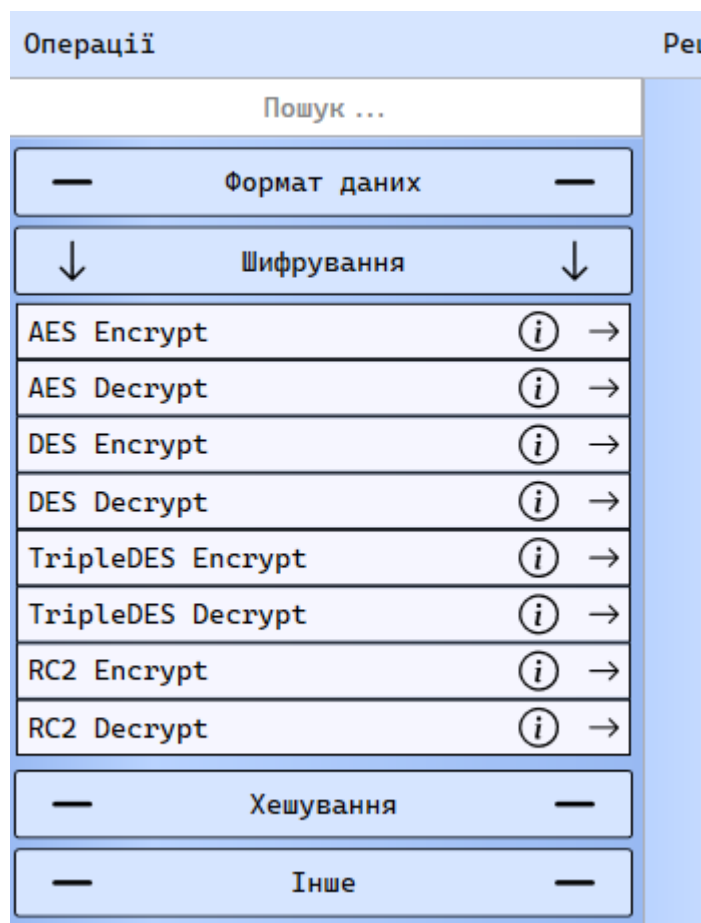


Рисунок 3.2 – Розгорнутий список «Шифрування»

При повторному натисненні на елемент «Шифрування» список згортається. Як видно на рисунку (див. рис. 3.2), в програмі реалізовані операції шифрування і розшифрування AES, DES, 3DES та RC2. Вкладка «Хешування» містить операції, показані на рис. 3.3.



Рисунок 3.3 – Список операцій «Хешування»

Вкладка «Формат даних» містить операції, показані на рис. 3.4.

↓	Формат даних	↓
To Hex	(i) →	
From Hex	(i) →	
To Decimal	(i) →	
From Decimal	(i) →	
To Binary	(i) →	
From Binary	(i) →	
To Octal	(i) →	
From Octal	(i) →	
To Base64	(i) →	
From Base64	(i) →	

Рисунок 3.4 – Список операцій «Формат даних»

Вкладка «Інше» містить операції, показані на рис. 3.5.

↓	Інше	↓
Random	(i) →	
ROT13	(i) →	
ROT47	(i) →	
ROT8000	(i) →	
Morse Encode	(i) →	
Morse Decode	(i) →	
Vigenere Encode	(i) →	
Vigenere Decode	(i) →	
XOR	(i) →	

Рисунок 3.5 – Список операцій «Інше»

Також частина «Операції» має поле «Пошук» для пошуку необхідних операцій за їх назвою. Приклад пошуку операцій «Нех» показаний на рис. 3.6.

Нех

↓	Формат даних	↓
To Hex	(i) →	
From Hex	(i) →	

Рисунок 3.6 – Функція пошуку

Кожен елемент списку «Операцій» може бути доданий до «Рецепту» при натисненні на кнопку «Стрілка вправо». Також для цих елементів передбачена можливість перегляду короткого опису операції, яку вони виконують. Перегляд короткого опису для елемента «To Hex» показано на рис. 3.7.

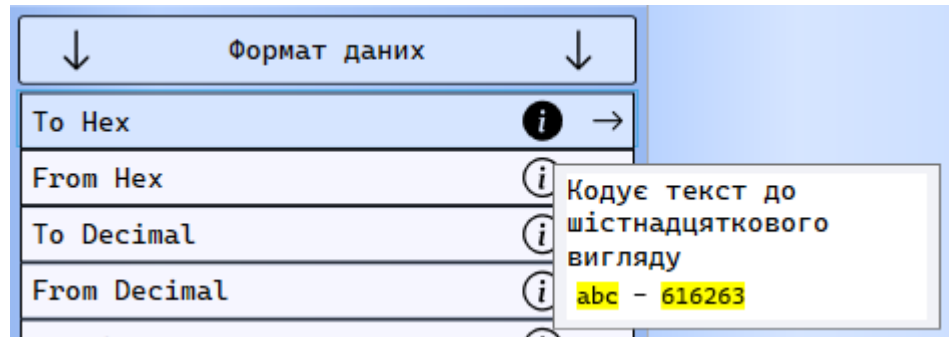


Рисунок 3.7 – Перегляд короткого опису

Наступна частина модуля – це «Рецепт». Він показує поточний алгоритм обробки тексту, який створив користувач. Приклад інтерфейсу «Рецепту» при доданому елементі з кожної категорії операцій показано на рис. 3.8.

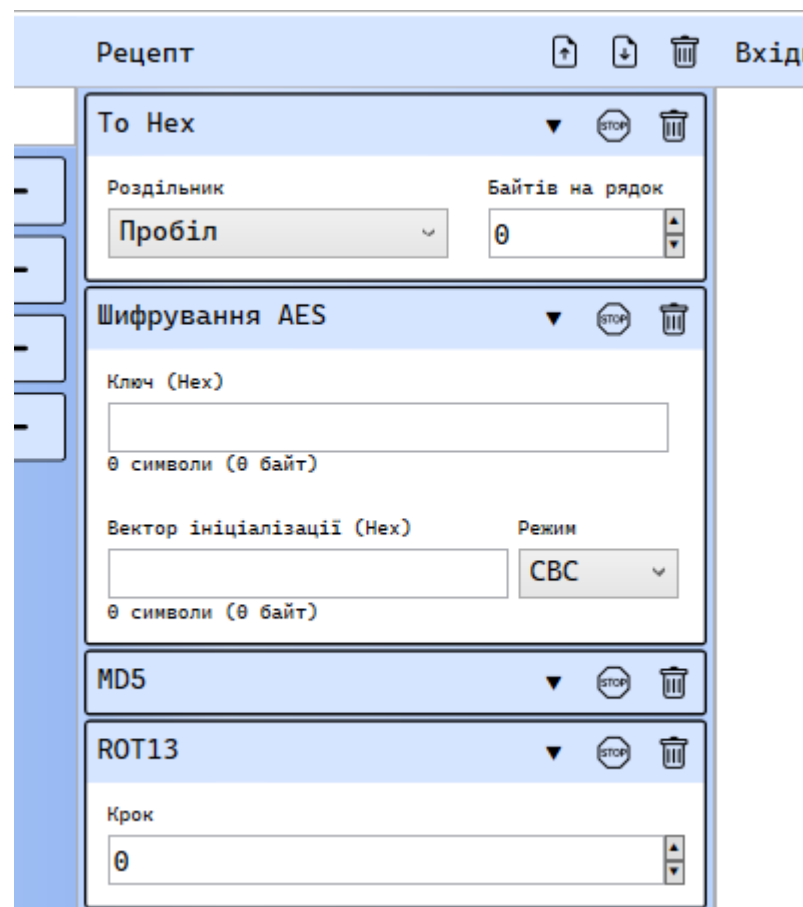


Рисунок 3.8 – Приклад «Рецепту»

Додані операції можна переміщати за допомогою «перетягування», вмикати ігнорування операції за допомогою кнопки «stop», згортати інтерфейс операції за допомогою «стрілки вниз», та видаляти операції при натисненні «сміттєвого баку». При ігноруванні операції з прикладу мають вигляд, як показано на рис. 3.9.

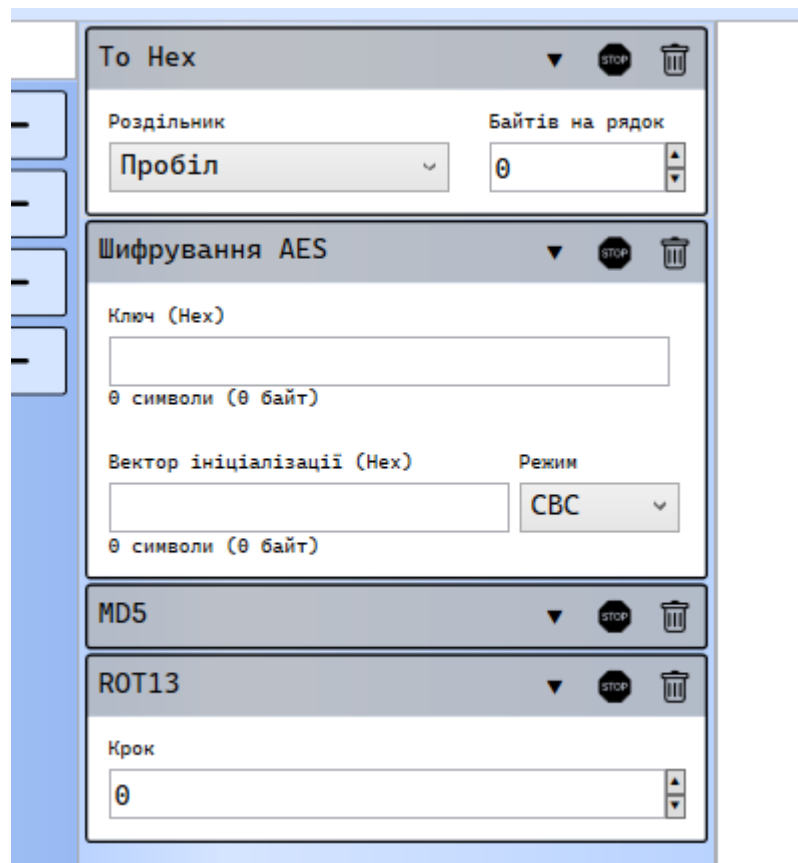


Рисунок 3.9 – Ігнорування операцій

Функція ігнорування потрібна, якщо користувач хоче пропустити операцію без її видалення. Функція згортання ховає підінтерфейс елементу «Рецепту», що дає більше місця на інтерфейсі «Рецепту». У згорнутому вигляді операції мають вигляд, що показаний на рис. 3.10.

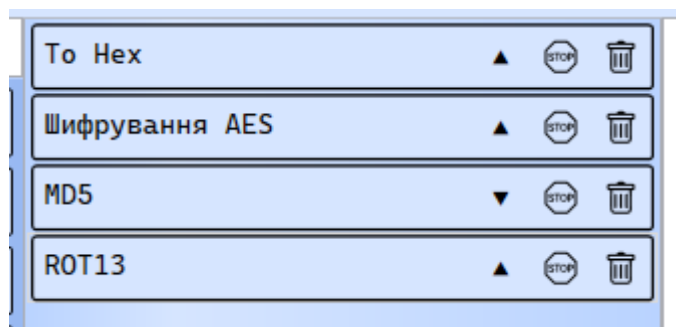


Рисунок 3.10 – Згорнуті операції

Також частина «Рецепт» має функціонал збереження алгоритму до файлу, та завантаження його звідти. У файл, крім операцій та їх порядку, також зберігаються:

- дані всередині операції;
- стан ігнорованості;
- стан згорнутості.

Збереження за завантаження рецепту з файлової системи відбувається кнопками, показаними на рис. 3.11.

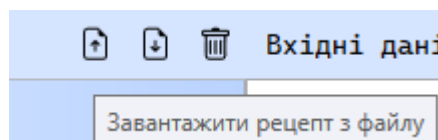


Рисунок 3.11 – Кнопки для збереження «Рецепту»

Кнопка «Кошика» (див. рис. 3.11) очищує поле «Рецепту» від операцій.

Останньою частиною поля «Рецепту» є кнопка «Виконати», яка обраховує перетворення тексту по алгоритму, який відображається списком в «Рецепті». Також поряд є можливість встановлення функції автоматичного виконання алгоритму, натиснувши на прапорець «Автоматичне виконання». Тоді алгоритм перетворення буде виконуватись при кожній зміні в списку «Рецепту». Зовнішній вигляд цього функціоналу показаний на рис. 3.12.

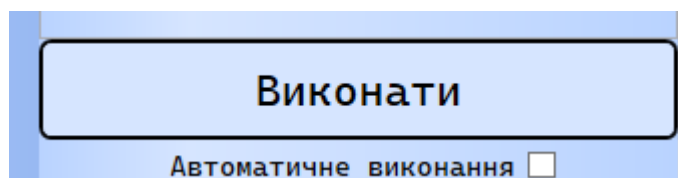


Рисунок 3.12 – Кнопка для обрахунку

Наступною частиною інтерфейсу є «Вхідні дані», які призначені для вводу текстової інформації, зміну зовнішнього вигляду тексту для зручності, та виведення простої інформації про текст. У верхній частині елемента перебувають кнопки для взаємодії з текстовим полем. По порядку їх функціонал:

- «Wrap» - вмикає автоматичний відображення тексту на новому рядку;
- «Шрифт» - змінює розмір шрифту тексту;
- «Корзина» - очищує тестове поле;
- «Копіювання» - вставляє текст із буфера обміну;
- «Завантажити» - відкриває файл та вставляє його вміст в тестове поле;

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		66

- «Кодування» - змінює режим читання вхідних даних між «Raw Data», «ASCII», «UTF8».

Більшу частину елемента займає текстове поле – простір для вводу вхідних даних. В нижній частині обраховується проста статистика:

- кількість символів;
- кількість рядків, що не враховує автоматичний перенос тексту;
- ентропія.

Зовнішній вигляд описаного елемента показаний на рис. 3.13.

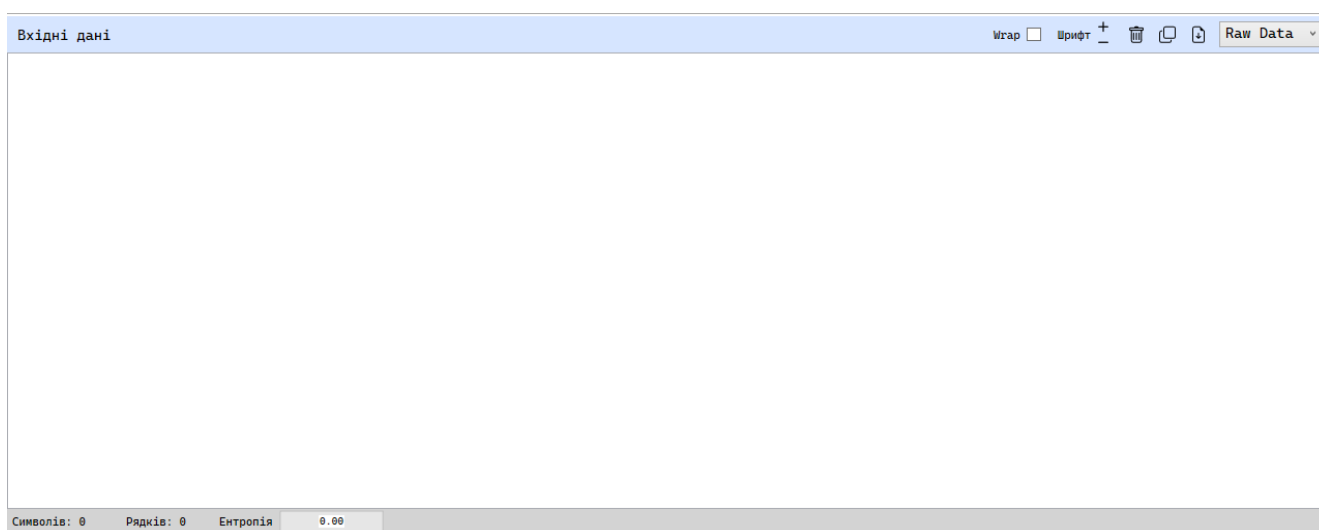


Рисунок 3.13 – Елемент «Вхідні дані»

Останній елемент «Результат» має схожі властивості і функціонал з «Вхідні дані», проте є наступні відмінності:

- замість вставляння з буферу обміну – в нього копіюється вміст текстового поля;
- замість читання файлу – в нього зберігається результат;
- текстове поле редагувати вручну не можливо;
- в рядку статистики також показує час витрачений на обрахування алгоритму.

Зовнішній вигляд частини «Результату» показаний на рис. 3.14.

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		67

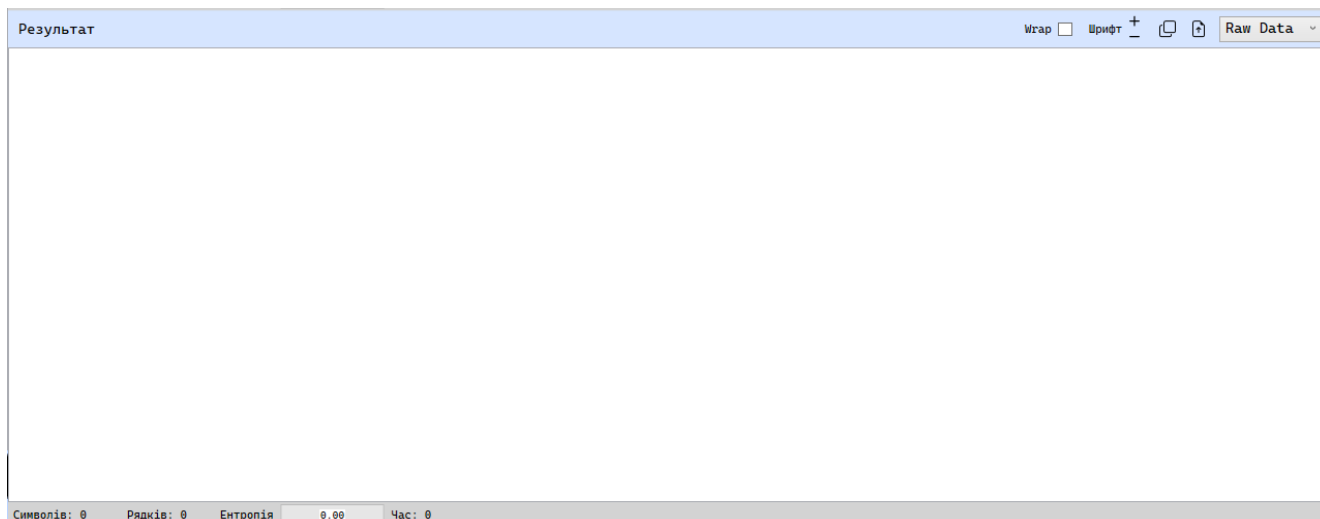


Рисунок 3.14 – Елемент «Результат»

Перше тестування модулю буде проведено на простому прикладі: уведений текст треба переробити у шістнадцятковий формат. Для цього в текстове поле вписуються дані, в поле «Рецепту» додається операція «То Нех», в операції задаються параметри роздільника та кількості байт на рядок. Для кращого вигляду шрифт вхідних і вихідних даних збільшено за допомогою кнопок «Шрифт» на відповідних елементах модулю. Приклад результату виконання показаний на рис. 3.15

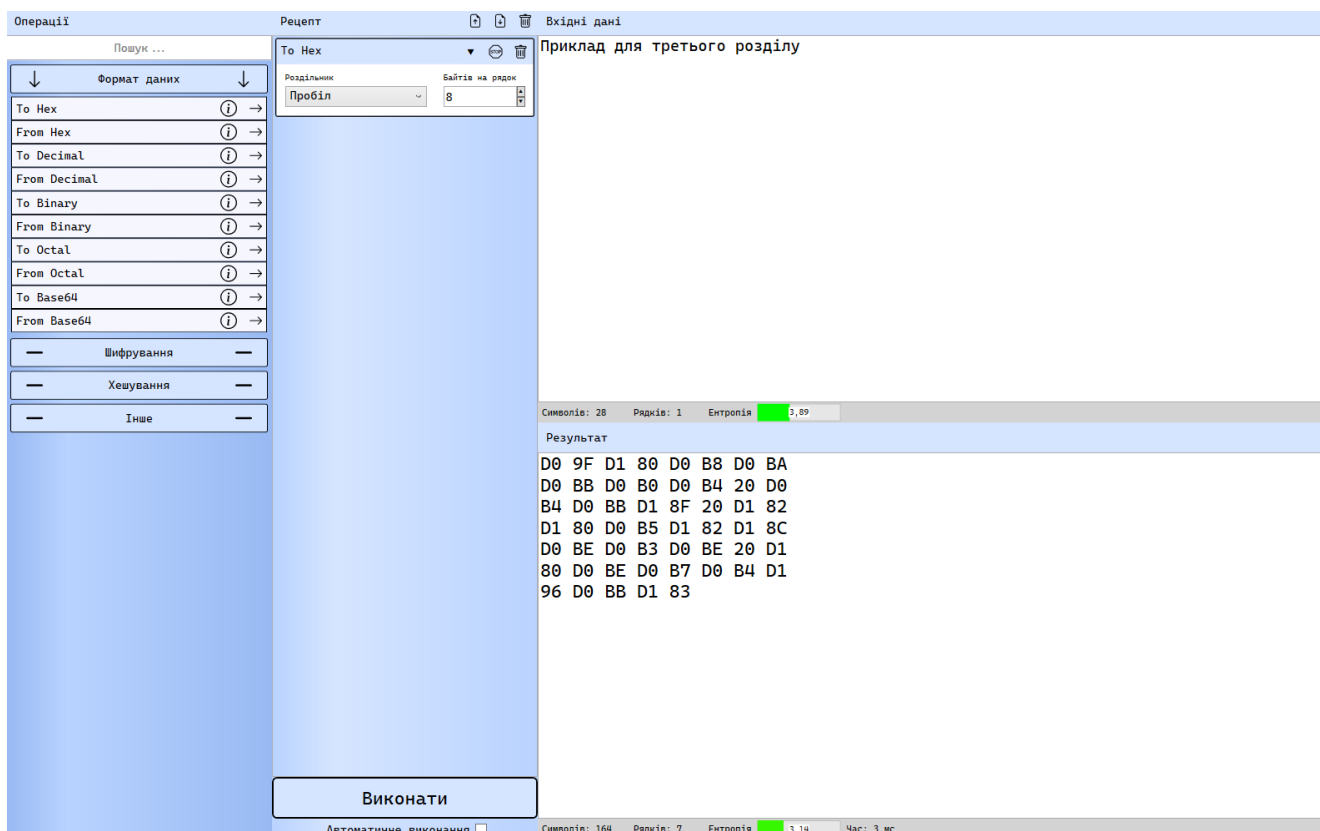


Рис. 3.15 – Приклад простої операції

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		68

Для наступного тестування буде проведено операцію шифрування і розшифрування алгоритмом AES. Для цього в поле «Рецепту» необхідно додати елементи «Шифрування AES» та «Дешифрування AES», та задати їм однакові ключі, вектори ініціалізації та режими роботи. Результат такого алгоритму показаний на рис. 3.16.

Рисунок 3.16 – Шифрування і дешифрування

Поля в операціях шифрування приймають тільки символи, що існують в шістнадцятковому форматі, для уникнення неправильного вводу тексту. Якщо ж довжина введених даних не буде правильною, програма видасть помилку в «Результаті». У випадку AES довжина ключа має бути 16, 24 або 32 байти. На наступному рис. 3.17 показано помилку, яку виводить програма, якщо ці умови не виконати.

Рисунок 3.17 – Виведення помилки

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		69

В продовження попереднього тесту, далі буде переглянуто результат шифрування в середині алгоритму попереднього «Рецепту». В програмній реалізації функції шифрування дають результат у шістнадцятковому вигляді. Тому для перегляду зашифрованого тексту між елементами «AES» додано операції перетворення до шістнадцяткового вигляду і назад. Для перегляду результату всередині алгоритму необхідно натиснути кнопки «stop» на операціях, які користувач хоче ігнорувати. В результаті виходить виконання програми, як показано на рис. 3.18.

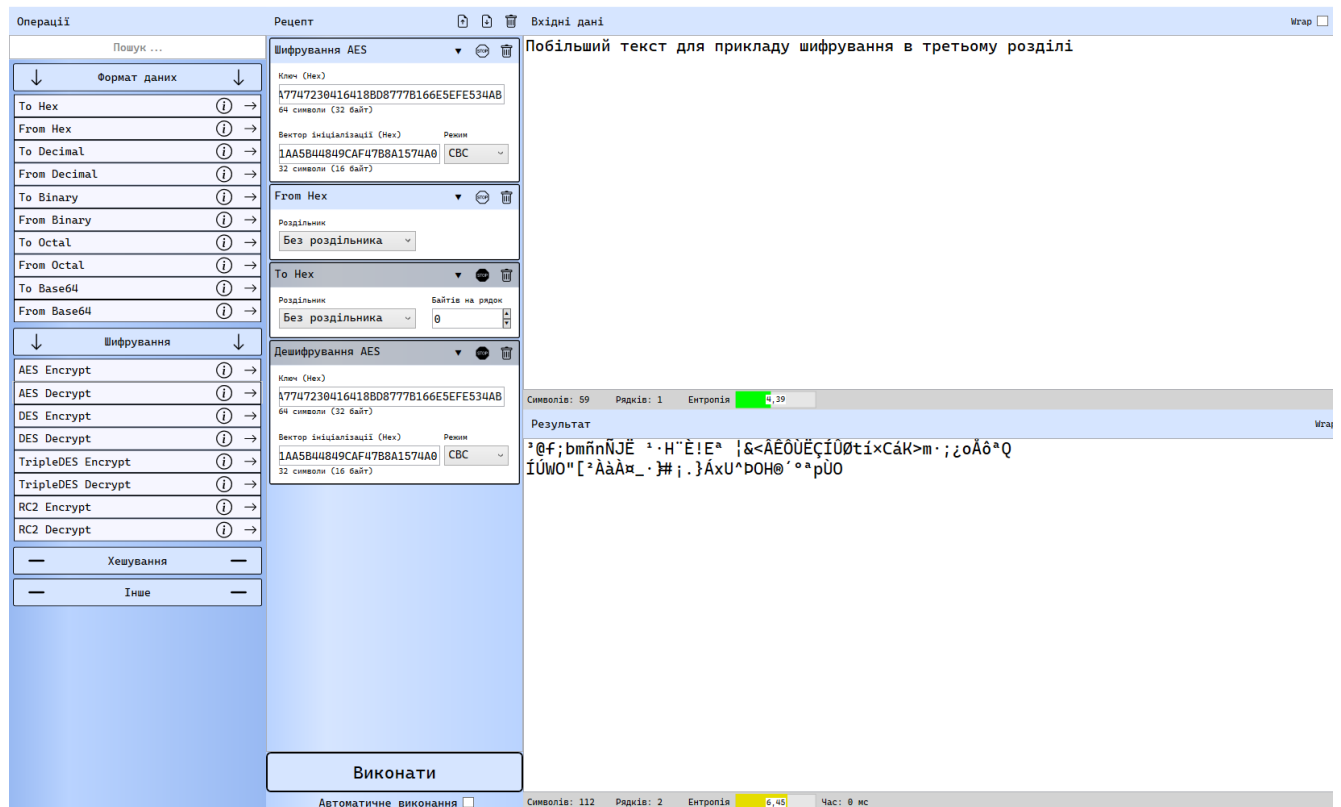


Рисунок 3.18 – Перегляд результату шифрування

Окрім перетвореного тексту, в його властивостях також виросло значення ентропії, що значить більшу випадковість та складність даних. При вимкненні ігнорування другої половини операцій здійснюються процес дешифрування тексту назад до відкрити вигляду, що успішно виконується на рис. 3.19.

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				70
Змн.	Арк.	№ докум.	Підпис	Дата		

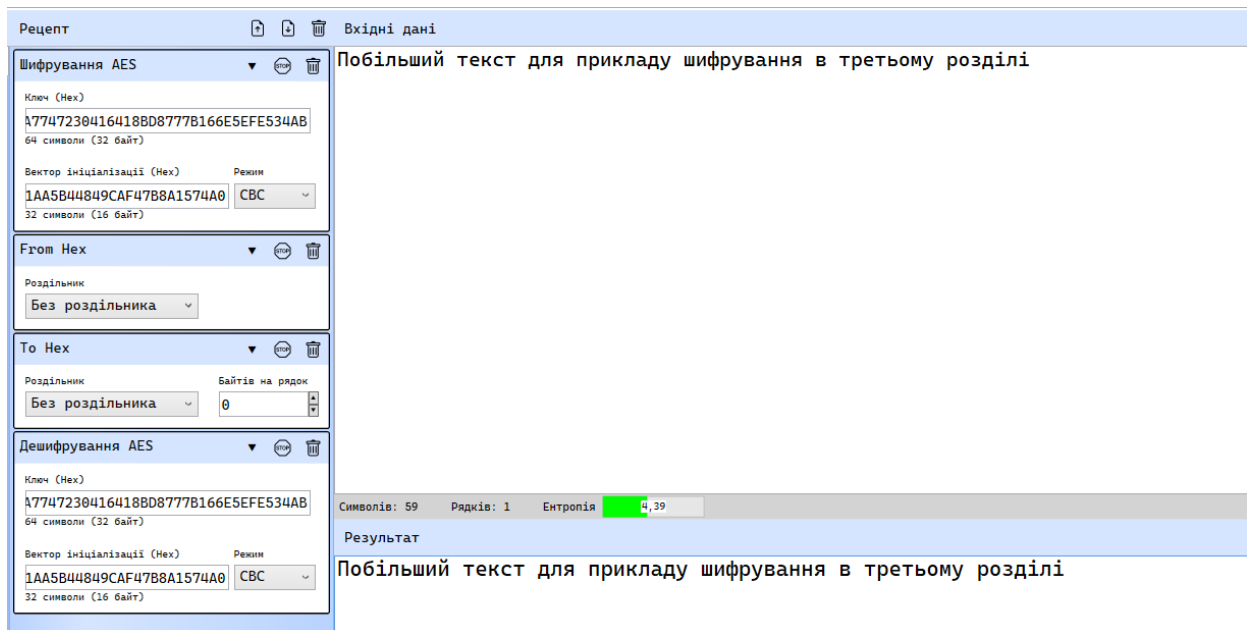


Рисунок 3.19 – Приклад перетворення

В результаті цього тесту було показано, що програма успішно виконує функції шифрування та дешифрування разом з кодуванням тексту, через що текст успішно перетворюється до зашифрованого і розшифрованого вигляду.

Далі цей «Рецепт» можна зберегти до файлу. Для цього необхідно натиснути на кнопку збереження рецепту, вибрати шлях та назву для файлу, та натиснути «Зберегти», як показано на рис. 3.20.

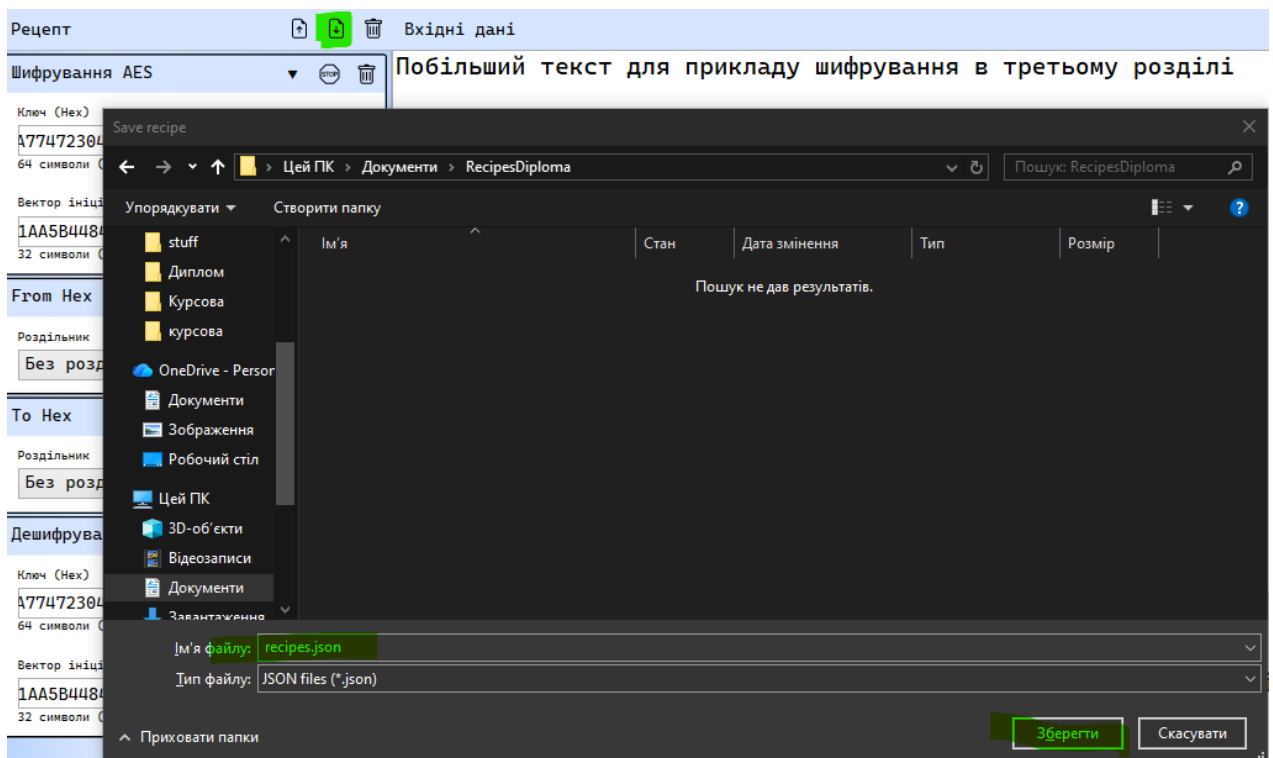


Рисунок 3.20 – Процес збереження «Рецепту»

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				71
Змн.	Арк.	№ докум.	Підпис	Дата		

Далі цей збережений алгоритм користувач може завантажити через кнопку завантаження рецепту. Цей функціонал дозволяє зберігати «Рецепти» без необхідності повтору створення їх кожного разу при необхідності. Вигляд збереженого рецепту у файлі показаний на рис. 3.21.

```

1  {
2    {
3      "RecipeItem": "Crypto",
4      "Parameters": {
5        "IsIgnored": "False",
6        "IsBodyVisible": "Visible",
7        "OperationType": "encaes",
8        "headerName": "\u0042\u0043\u0044\u0044\u0043\u0043\u0043\u0043\u0044F AES",
9        "Key": "5B0FD636AF36BC084F6F5AA35A2708DCFD3C16B98ADC7F40BAF4265B0D924B51",
10       "IV": "926007B17503E4A8032640D710845BED",
11       "SelectedMode": "0"
12     }
13   },
14   {
15     "RecipeItem": "Crypto",
16     "Parameters": {
17       "IsIgnored": "False",
18       "IsBodyVisible": "Visible",
19       "OperationType": "decaes",
20       "headerName": "\u0041\u0043\u0044\u0043\u0044\u0044\u0043\u0043\u0043\u0044F AES",
21       "Key": "5B0FD636AF36BC084F6F5AA35A2708DCFD3C16B98ADC7F40BAF4265B0D924B51",
22       "IV": "926007B17503E4A8032640D710845BED",
23       "SelectedMode": "0"
24     }
25   }
26 }

```

Рисунок 3.21 – Збережений рецепт

Маючи великий вибір операцій, користувач має можливість створення великої кількості варіантів «Рецепту». Для тестування було вибрано деякі з них, які наведено далі.

Приклад операції хешування показаний на рис. 3.22.

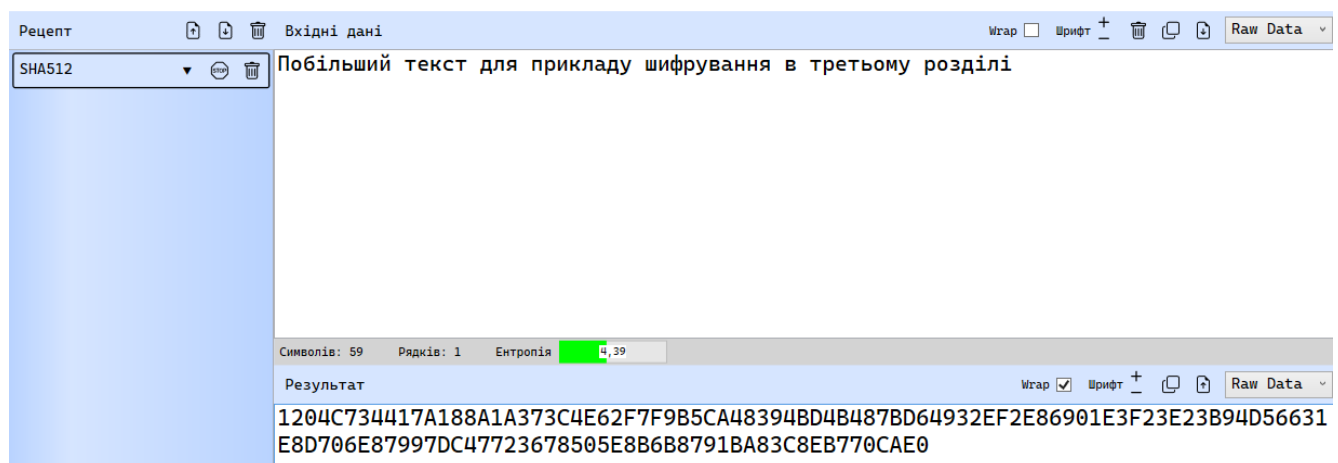


Рисунок 3.22 – Приклад хешування SHA512

Приклад операції кодування Морзе показаний на рис. 3.24.

Рецепт

Морзе Encode

Побільший текст для прикладу шифрування в третьому розділі

Виконати

Символів: 59 Рядків: 1 Ентропія: 4,39

Результат

Час: 1 мс

Рисунок 3.23 – Приклад кодування Морзе

Приклад операції шифрування алгоритмом Віженера показаний на рис. 3.24.

Рецепт

Vigenere Encode

Ключ (Текст)

секрет

6 символів

Алфавіт

Українська

Виконати

Символів: 59 Рядків: 1 Ентропія: 4,56

Результат

Час: 0 мс

Рисунок 3.24 – Приклад шифрування Віженера

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				73
Змн.	Арк.	№ докум.	Підпис	Дата		

3.2 Тестування модуля аналізу тексту

Другим модулем програмного рішення є модуль для аналізу тексту. При переході на нього натисненням на вкладку «Аналіз», користувачу постає інтерфейс, показаний на рис. 3.25.

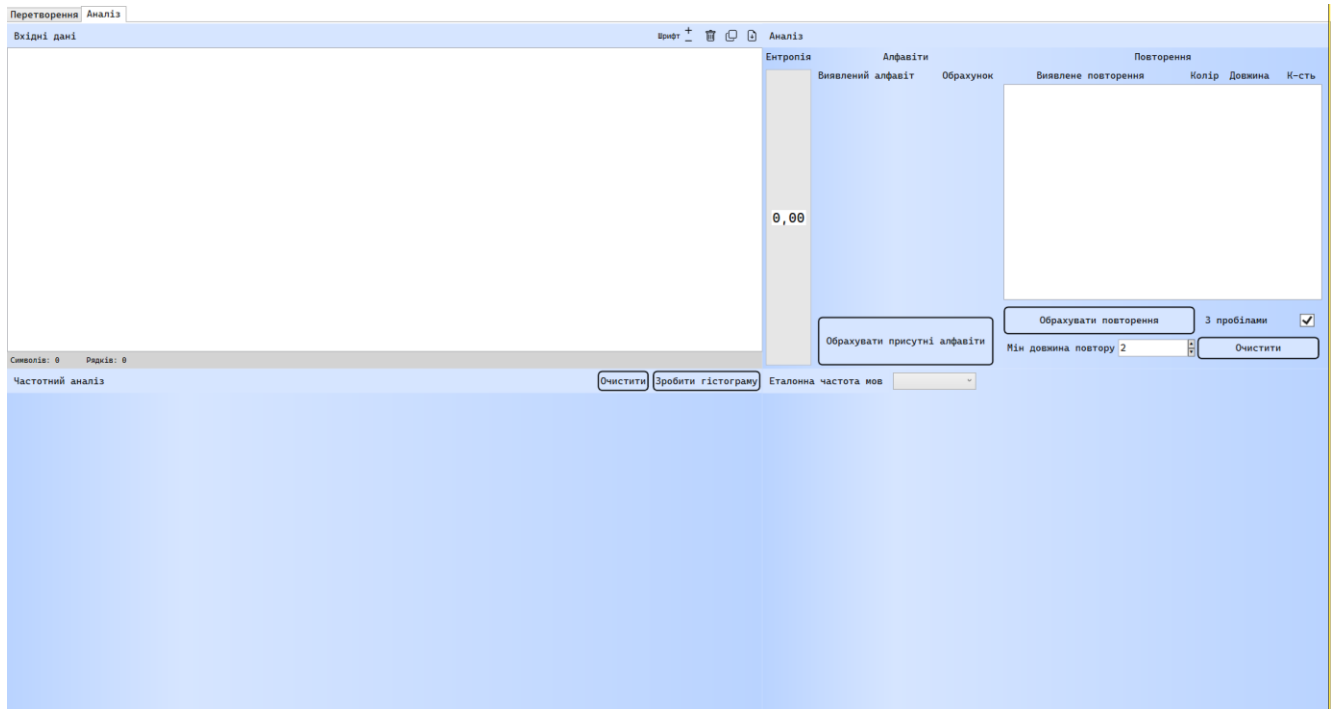


Рисунок 3.25 – Модуль «Аналіз»

Інтерфейс цього модуля складається з 4 частин:

- «Вхідні дані» - поле для вводу тексту;
- «Аналіз» - панель з функціями аналізу тексту;
- «Частотний аналіз» - поле для виводу результату частотного аналізу;
- «Еталонна частота мов» - поле для виведення еталонних частот мов.

Вхідні дані мають функціонал, аналогічний частині «Вхідні дані» модуля «Перетворення». Різницею є відсутність можливості налаштування автоматичного переходу тексту на новий рядок та відсутність вибору кодування, так як для аналізу тексту воно не має впливу. В рядку статистики відсутній показник ентропії, так як його перенесено на панель «Аналіз».

Панель «Аналіз» складається з трьох частин для різних видів аналізу – «Ентропія», «Алфавіти» та «Повторення». «Ентропія» показує значення ентропії через

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		74

число та «progressBar». «Алфавіти» показують виявлені алфавіти у введеному тексті. «Повторення» показують обраховані повтори тексту. Зовнішній вигляд панелі «Аналізу» показаний на рис. 3.26.

Рисунок 3.26 – Панель «Аналіз»

Для перегляду функціоналу необхідно спочатку ввести дані у текстове поле, що зроблено на рис. 3.27.

Рисунок 3.27 – Дані для тестування

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				75
Змн.	Арк.	№ докум.	Підпис	Дата		

Значення «Ентропії» обраховується автоматично. Для обрахунку використуваних алфавітів необхідно натиснути кнопку «Обрахувати присутні алфавіти». В результаті програма покаже результати обрахунку алфавітів у вигляді горизонтальної гістограми, що показано на рис. 3.28.

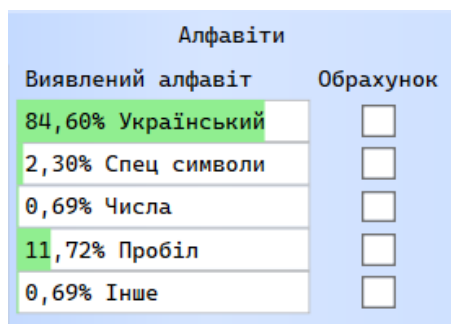


Рисунок 3.28 – Обраховані «Алфавіти»

Частина з обрахуванням «Повторення» має в собі декілька налаштувань пошуку повторів, а саме:

- прапорець врахування пробілу при пошуку;
- мінімальна довжина повтору.

При вимкненні прапорця врахування пробілу та встановленні мінімальної довжини пошуку повторів «4», результат пошуку показаний на рис. 3.29.

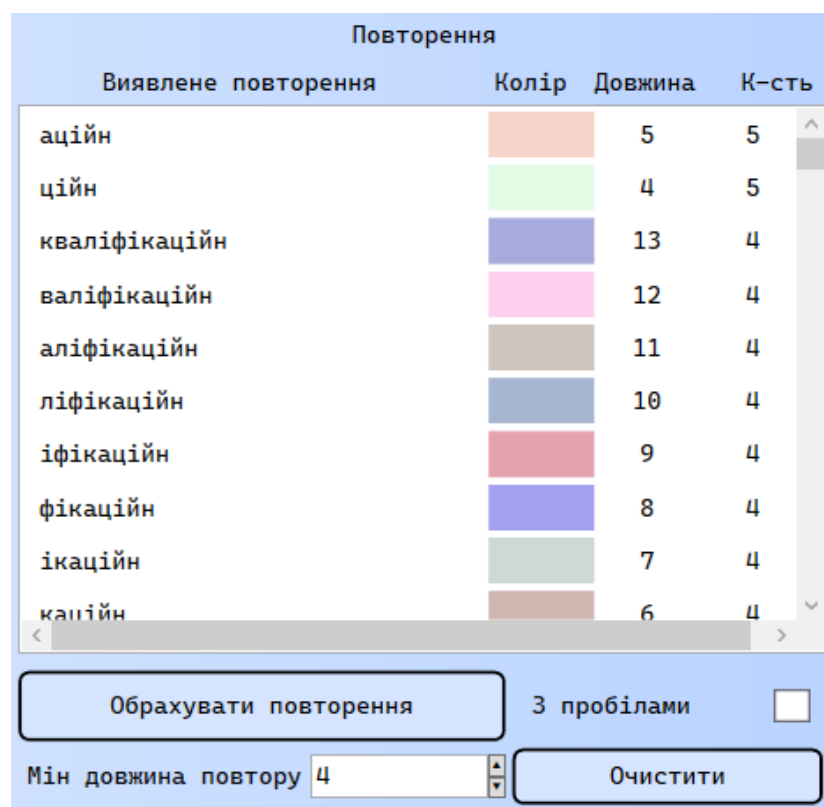


Рисунок 3.29 – Обраховані повтори

В списку повторів виводиться текст повтору, його довжина і кількість. Також є стовбець кольору – він відповідає за колір виділення повтору при його виборі в списку. Приклад такого виділення показаний на рис. 3.30.

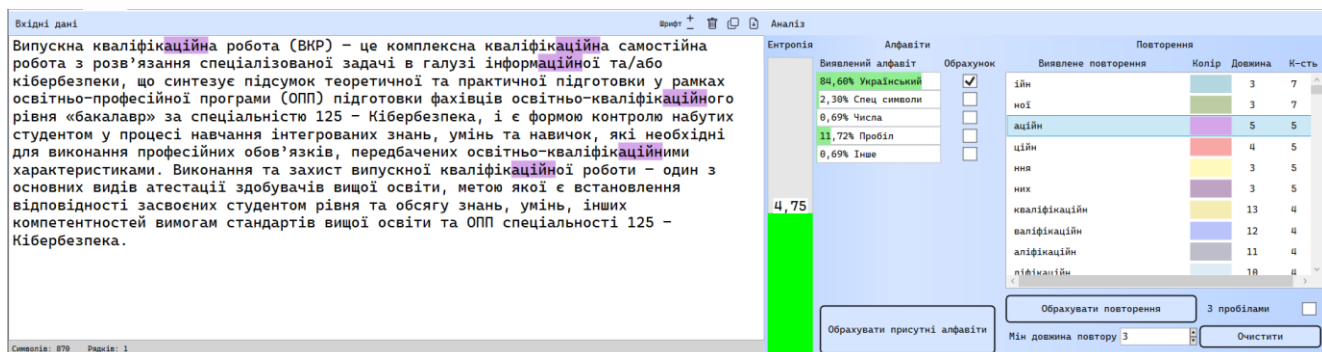


Рисунок 3.30 – Виділення повторів

Для обрахування частотного аналізу користувачу необхідно в панелі «Алфавіти» прапорцями вибрати словники, які будуть враховуватись при обрахуванні гістограми. Для прикладу, в «Алфавітах» було вибрано тільки «Числа» і «Спец символи», як показано на рис. 3.31.

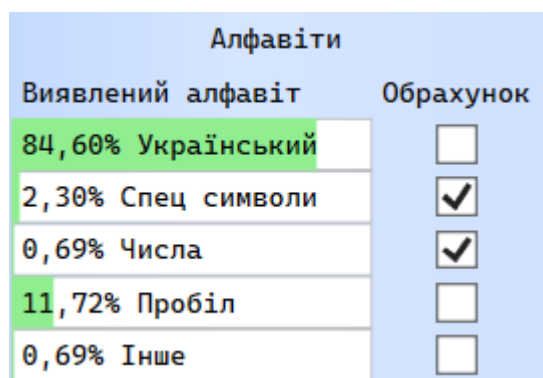


Рисунок 3.31 – Вибір алфавітів

При натисненні на кнопку «Зробити гістограму» на панелі «Частотний аналіз» створився результат, показаний на рис. 3.32.

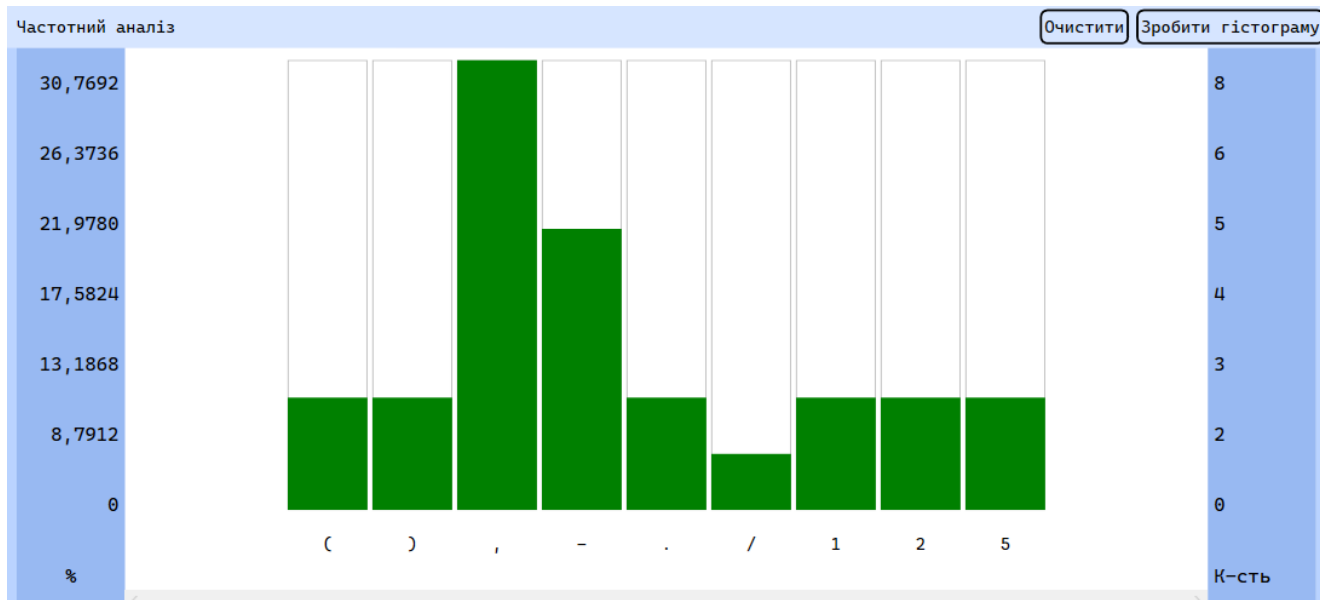


Рисунок 3.32 – Приклад першого частотного аналізу

При аналогічному виборі тільки «Український» алфавіту, при натисненні на кнопку «Зробити гістограму» обраховується частотний аналіз з результатом, показаний на рис. 3.33.

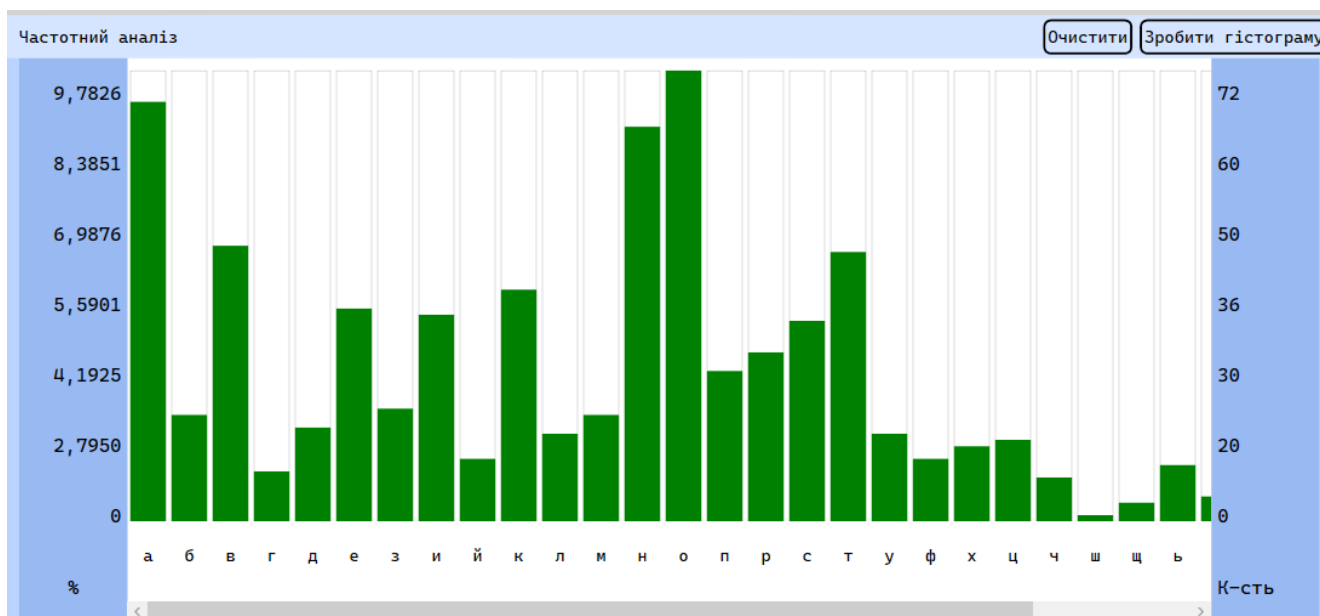


Рисунок 3.33 – Частотний аналіз літер української мови в тексті

На останній частині «Аналізу» можливо вибрати показ еталонної частоти розподілу букв у англійській і українській мові, що буде показано у вигляді подібної гістограми.

В результаті виконання роботи всіх частин модуля «Аналізу», зовнішній вигляд інтерфейсу буде виглядати, як показано на рис. 3.34.

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				78
Змн.	Арк.	№ докум.	Підпис	Дата		

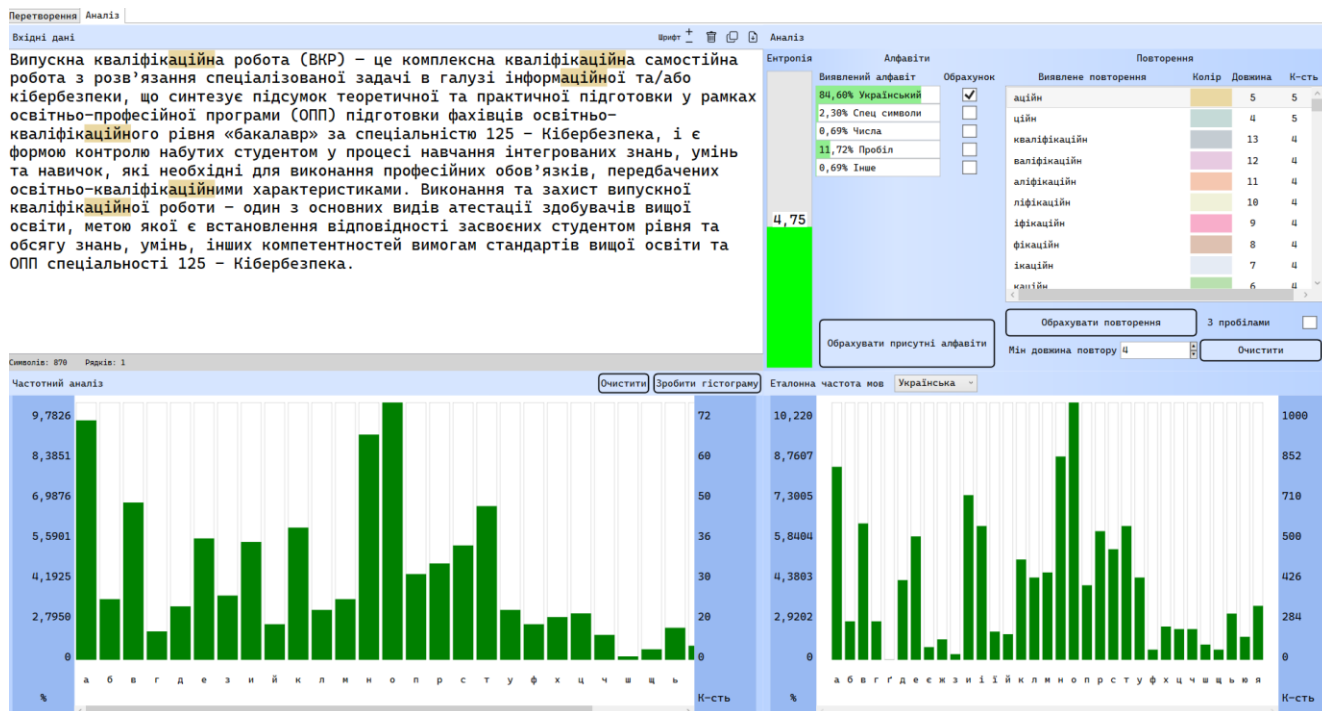


Рисунок 3.34 – Виконання всіх аналізів

Висновок до третього розділу

В результаті третього розділу була представлена робота програмного рішення для вирішення прикладних криптографічних задач.

Було оглянуто інтерфейс першого модуля перетворення текстових даних, представлена робота декількох прикладів алгоритмів перетворення, продемонстрована робота функціоналу взаємодії з файловою системою, перевірена стійкість до введення неправильних даних.

Продемонстрована робота другого модуля, а саме розглянуто реалізовані методу аналізу тексту – його ентропія, використовувані алфавіти, повторення та частотний аналіз.

Створене програмне рішення виконує усі поставлені завдання та реалізовує всі оглянуті в теоретичній частині алгоритми перетворення та аналізу тексту.

ВИСНОВКИ

Основним результатом кваліфікаційної роботи є розробка програмного рішення для аналізу та розв'язання прикладних криптографічних задач, яке забезпечує інтеграцію функцій перетворення та аналізу текстових даних у межах єдиного програмного середовища та відповідає поставленій меті роботи.

У процесі виконання роботи було реалізовано наступні функції програмного комплексу:

- модуль перетворення тексту, що забезпечує виконання операцій кодування, декодування, шифрування та хешування даних;
- механізм формування «рецептів», який дозволяє створювати послідовності операцій обробки даних;
- модуль аналізу тексту, що реалізує статистичний, структурний та лінгвістичний аналіз;
- функції обчислення ентропії, частотного розподілу символів та виявлення повторень в тексті;
- зручний графічний інтерфейс користувача, реалізований із використанням технології WPF.

Реалізовані функції повністю відповідають поставленим завданням кваліфікаційної роботи. Зокрема, виконано теоретичний аналіз предметної області, досліджено методи перетворення та аналізу даних, проведено огляд існуючих програмних аналогів, спроектовано архітектуру програмного рішення та реалізовано його основний функціонал. Ступінь відповідності результатів поставленим завданням є високим, оскільки програмний комплекс охоплює як базові, так і розширені можливості обробки та аналізу текстових даних.

Практична цінність отриманих результатів полягає у можливості використання розробленого програмного рішення для розв'язання прикладних задач у сфері кібербезпеки, зокрема під час аналізу зашифрованих повідомлень, виконання навчальних та практичних завдань, а також підготовки до CTF-змагань. Розроблений програмний комплекс може бути використаний як навчальний інструмент та допоміжний засіб для спеціалістів з інформаційної безпеки.

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				80
Змн.	Арк.	№ докум.	Підпис	Дата		

Подальший розвиток роботи може бути спрямований на розширення функціональних можливостей програмного комплексу, зокрема додавання нових криптографічних алгоритмів, реалізацію автоматичного визначення типу шифру, удосконалення методів криптоаналізу, а також оптимізацію інтерфейсу користувача та підвищення продуктивності програмного забезпечення.

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		81

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Вербіцький О. В. Вступ до криптології. Львів: Видавництво науково-технічної літератури, 1998. ISBN 966-7148-03-3.
2. Щур Н.О. Лекція 1. Основні поняття та задачі криптології. Житомир : Державний університет Житомирська політехніка. URL: https://learn.ztu.edu.ua/pluginfile.php/406460/mod_resource/content/1/%D0%9B%D0%B5%D0%BA%D1%86%D1%96%D1%8F1.pdf.
3. Jennings T. An annotated history of some character codes or ASCII. WPS. 29.10.2004. URL: <https://web.archive.org/web/20081019221633/http://www.wps.com/projects/codes/index.html>.
4. Yergeau F., Alis Technologies. UTF-8, a transformation format of ISO 10646. Datatracker. 01.11.2003. URL: <https://datatracker.ietf.org/doc/html/rfc3629>.
5. How do you learn Morse code, and what is it used for?. BBC. IYKYK. URL: <https://www.bbc.co.uk/bitesize/articles/z4bm9ty#z9gxkhv>.
6. Josefsson S., SJD. The Base16, Base32, and Base64 Data Encodings. Datatracker. 01.10.2006. URL: <https://datatracker.ietf.org/doc/html/rfc4648>.
7. Savarese C., Hart B. The Caesar Cipher. Cryptography. 26.04.2010. URL: <https://www.cs.trincoll.edu/~crypto/historical/caesar.html>.
8. Vigenere Cipher. Dcode. Poly-Alphabetic Cipher. URL: <https://www.dcode.fr/vigenere-cipher>.
9. DATA ENCRYPTION STANDARD (DES) / National Institute of Standards and Technology. Federal Information Processing Standards, 1999. URL: <https://csrc.nist.gov/files/pubs/fips/46-3/final/docs/fips46-3.pdf>.
10. Advanced Encryption Standard (AES) / National Institute of Standards and Technology. Federal Information Processing Standards, 2001. URL: <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf>.
11. Rivest R., MIT Laboratory for Computer Science. RFC 2268: A Description of the RC2(r) Encryption Algorithm. RFC Editor. 01.03.1998. URL: <https://www.rfc-editor.org/info/rfc2268/>.

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				82
Змн.	Арк.	№ докум.	Підпис	Дата		

- 12.Що таке хешування?. *Binance. Криптографія.* 31.01.2023. URL: <https://www.binance.com/uk-UA/academy/articles/what-is-hashing>.
- 13.Rivests R., MIT Laboratory for Computer Science. The MD5 Message-Digest Algorithm. *RFC Editor.* 01.04.1992. URL: <https://www.rfc-editor.org/rfc/rfc1321.html>.
- 14.Bosselaers A., Preneel B. The hash function RIPEMD-160. *Personal page.* URL: <https://homes.esat.kuleuven.be/~bosselae/ripemd160.html>.
- 15.SECURE HASH STANDARD / National Institute of Standards and Technology. Federal Information Processing Standards Publication, 1995. URL: <https://nvlpubs.nist.gov/nistpubs/Legacy/FIPS/fipspub180-1.pdf>.
- 16.National Institute of Standards and Technology. Descriptions of SHA-256, SHA-384, and SHA-512. *NIST.* URL: <https://web.archive.org/web/20130526224224/https://csrc.nist.gov/groups/STM/cavp/documents/shs/sha256-384-512.pdf>.
- 17.Сушко, С., Фомичова, Л., & Барсуков, Є. (2010). ЧАСТОТИ ПОВТОРЮВАНOSTІ БУКВ І БІГРАМ У ВІДКРИТИХ ТЕКСТАХ УКРАЇНСЬКОЮ МОВОЮ. *Ukrainian Information Security Research Journal*, 12(3 (48)). <https://doi.org/10.18372/2410-7840.12.1968>
- 18.Бродський Ю. Б. Лекція 1. Вступ. Інформація та ентропія: кількісна міра. Державний університет Житомирська політехніка. URL: https://learn.ztu.edu.ua/pluginfile.php/424016/mod_resource/content/1/%D0%9B%D0%B5%D0%BA%D1%86%D1%96%D1%8F%201_2025%20%D0%A2%D0%B5%D0%BE%D1%80%20%D1%96%D0%BD%D1%84_%D0%92%D1%81%D1%82%D1%83%D0%BF_%D0%9A%D1%96%D0%BB%D1%8C%D0%BA%20%D0%BC%D1%96%D1%80%D0%B0_.pdf.
19. GCHQ. *CyberChef*. URL: <https://gchq.github.io/CyberChef/>.
- 20.About dCode on dCode.fr. *dCode.fr*. URL: <https://www.dcode.fr/about>.
- 21.About CrypTool 2. *CrypTool*. URL: <https://www.cryptool.org/en/ct2/>.
- 22.Microsoft. C# documentation. *Microsoft. Learn.* URL: <https://learn.microsoft.com/uk-ua/dotnet/csharp/tour-of-csharp/>.

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				83
Змн.	Арк.	№ докум.	Підпис	Дата		

23. Microsoft. Windows Presentation Foundation Overview. *Microsoft. Learn.*
URL: <https://learn.microsoft.com/en-us/dotnet/desktop/wpf/overview/>.
24. Microsoft. What is Visual Studio?. *Microsoft. Learn.* URL:
<https://learn.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=visualstudio>.

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		84

ДОДАТКИ

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				85
Змн.	Арк.	№ докум.	Підпис	Дата		

Функції для створення об'єктів в «Рецепті»

```

private void AddOperationToReceiptList(object parameter)
{
    bool delimiter = true,
        noDelimiter = false,
        numPicker = true,
        noNumPicker = false;

    Recipe_Universal recipeItem = new Recipe_Universal();

    switch (parameter)
    {
        case "ToHex":
            recipeItem = new Recipe_Universal("To Hex", "toHex",
            delimiter, numPicker, "Байтів на рядок");
            break;
        case "FromHex":
            recipeItem = new Recipe_Universal("From Hex", "fromHex",
            delimiter);
            break;
        case "ToDecimal":
            recipeItem = new Recipe_Universal("To Decimal",
            "toDecimal", delimiter);
            break;
        case "FromDecimal":
            recipeItem = new Recipe_Universal("From Decimal",
            "fromDecimal", delimiter);
            break;
        case "ToBinary":
            recipeItem = new Recipe_Universal("To Binary", "toBinary",
            delimiter);
            break;
        case "FromBinary":
            recipeItem = new Recipe_Universal("From Binary",
            "fromBinary", delimiter);
            break;
        case "ToOctal":
            recipeItem = new Recipe_Universal("To Octal", "toOctal",
            delimiter);
            break;
        case "FromOctal":
            recipeItem = new Recipe_Universal("From Octal",
            "fromOctal", delimiter);
            break;
        case "ToBase64":
            recipeItem = new Recipe_Universal("To Base64",
            "toBase64");
            break;
        case "FromBase64":
    
```

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				86
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        recipeItem = new Recipe_Universal("From Base64",
"fromBase64");
        break;
    case "MD5":
        recipeItem = new Recipe_Universal("MD5", "md5");
        break;
    case "RIPEMD160":
        recipeItem = new Recipe_Universal("RIPEMD160",
"ripemd160");
        break;
    case "SHA1":
        recipeItem = new Recipe_Universal("SHA1", "sha1");
        break;
    case "SHA256":
        recipeItem = new Recipe_Universal("SHA256", "sha256");
        break;
    case "SHA384":
        recipeItem = new Recipe_Universal("SHA384", "sha384");
        break;
    case "SHA512":
        recipeItem = new Recipe_Universal("SHA512", "sha512");
        break;
    case "ROT13":
        recipeItem = new Recipe_Universal("ROT13", "rot13",
noDelimiter, numPicker, "Крок");
        break;
    case "ROT47":
        recipeItem = new Recipe_Universal("ROT47", "rot47",
noDelimiter, numPicker, "Крок");
        break;
    case "ROT8000":
        recipeItem = new Recipe_Universal("ROT8000", "rot8000",
noDelimiter, numPicker, "Крок");
        break;
    case "morseenc":
        recipeItem = new Recipe_Universal("Morse Encode",
"morseenc");
        break;
    }

    recipeItem.DeleteRequested += (s, e) => {
RecipeList.Items.Remove(recipeItem); };
    recipeItem.DeleteRequested += (s, e) =>
    {
        if (checkboxAutorecipe.IsChecked == true)
            buttonCalculate.RaiseEvent(new
RoutedEventArgs(System.Windows.Controls.Button.ClickEvent));
    };
    RecipeList.Items.Add(recipeItem);

    if (checkboxAutorecipe.IsChecked == true)

```

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				87
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        {
            buttonCalculate.RaiseEvent(new
RoutedEventArgs(System.Windows.Controls.Button.ClickEvent));
        }
    }

private void AddOtherOperationToReceiptList(object parameter)
{
    switch (parameter)
    {
        case "Random":
            var recipeItemRandom = new Recipe_Random();
            recipeItemRandom.DeleteRequested += (s, e) => {
RecipeList.Items.Remove(recipeItemRandom); };
            recipeItemRandom.DeleteRequested += (s, e) =>
            {
                if (checkboxAutorecipe.IsChecked == true)
                    buttonCalculate.RaiseEvent(new
RoutedEventArgs(System.Windows.Controls.Button.ClickEvent));
            };
            RecipeList.Items.Add(recipeItemRandom);
            break;
        case "morsedec":
            var recipeItemMorse = new Recipe_MorseDecode();
            recipeItemMorse.DeleteRequested += (s, e) => {
RecipeList.Items.Remove(recipeItemMorse); };
            recipeItemMorse.DeleteRequested += (s, e) =>
            {
                if (checkboxAutorecipe.IsChecked == true)
                    buttonCalculate.RaiseEvent(new
RoutedEventArgs(System.Windows.Controls.Button.ClickEvent));
            };
            RecipeList.Items.Add(recipeItemMorse);
            break;
        case "vigenc":
            var recipeItemVigEnc = new Recipe_Vigenere("Vigenere
Encode", "vigenc");
            recipeItemVigEnc.DeleteRequested += (s, e) => {
RecipeList.Items.Remove(recipeItemVigEnc); };
            recipeItemVigEnc.DeleteRequested += (s, e) =>
            {
                if (checkboxAutorecipe.IsChecked == true)
                    buttonCalculate.RaiseEvent(new
RoutedEventArgs(System.Windows.Controls.Button.ClickEvent));
            };
            RecipeList.Items.Add(recipeItemVigEnc);
            break;
        case "vigdec":
            var recipeItemVigDec = new Recipe_Vigenere("Vigenere
Decode", "vigdec");
            recipeItemVigDec.DeleteRequested += (s, e) => {
RecipeList.Items.Remove(recipeItemVigDec); };

```

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		88

```

        recipeItemVigDec.DeleteRequested += (s, e) =>
        {
            if (checkboxAutorecipe.IsChecked == true)
                buttonCalculate.RaiseEvent(new
RoutedEventArgs(System.Windows.Controls.Button.ClickEvent));
            };
            RecipeList.Items.Add(recipeItemVigDec);
            break;
        case "xor":
            var recipeItemXor = new Recipe_XOR();
            recipeItemXor.DeleteRequested += (s, e) => {
RecipeList.Items.Remove(recipeItemXor); };
            recipeItemXor.DeleteRequested += (s, e) =>
            {
                if (checkboxAutorecipe.IsChecked == true)
                    buttonCalculate.RaiseEvent(new
RoutedEventArgs(System.Windows.Controls.Button.ClickEvent));
            };
            RecipeList.Items.Add(recipeItemXor);
            break;
        }

        if (checkboxAutorecipe.IsChecked == true)
        {
            buttonCalculate.RaiseEvent(new
RoutedEventArgs(System.Windows.Controls.Button.ClickEvent));
        }
    }

private void AddCryptoOperationToReceiptList(object parameter)
{
    Recipe_Crypto recipeItem = new Recipe_Crypto();
    switch (parameter)
    {
        case "EncAES":
            recipeItem = new Recipe_Crypto("Шифрування AES",
"encaes");
            break;
        case "DecAES":
            recipeItem = new Recipe_Crypto("Дешифрування AES",
"decaes");
            break;
        case "EncDES":
            recipeItem = new Recipe_Crypto("Шифрування DES",
"encdes");
            break;
        case "DecDES":
            recipeItem = new Recipe_Crypto("Дешифрування DES",
"decdes");
            break;
        case "EncTDES":

```

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				89
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        recipeItem = new Recipe_Crypto("Шифрування TripleDES",
"enctdes");
        break;
        case "DecTDES":
            recipeItem = new Recipe_Crypto("Дешифрування TripleDES",
"dectdes");
            break;
        case "EncRC2":
            recipeItem = new Recipe_Crypto("Шифрування RC2",
"encrc2");
            break;
        case "DecRC2":
            recipeItem = new Recipe_Crypto("Дешифрування RC2",
"decrc2");
            break;
    }
    recipeItem.DeleteRequested += (s, e) => {
RecipeList.Items.Remove(recipeItem); };
    recipeItem.DeleteRequested += (s, e) =>
    {
        if (checkboxAutorecipe.IsChecked == true)
            buttonCalculate.RaiseEvent(new
RoutedEventArgs(System.Windows.Controls.Button.ClickEvent));
    };
    RecipeList.Items.Add(recipeItem);

    if (checkboxAutorecipe.IsChecked == true)
    {
        buttonCalculate.RaiseEvent(new
RoutedEventArgs(System.Windows.Controls.Button.ClickEvent));
    }
}

```

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		90

Функція обрахунку алгоритму перетворення

```

private void Button_CalculateRecipe(object sender, RoutedEventArgs e)
{
    string text = TextBoxInput.Text;
    var watch = Stopwatch.StartNew();

    foreach (var item in RecipeList.Items)
    {
        switch (item.ToString())
        {
            case "Diplom.MyInterface.RecipeItems.Recipe_Universal":
                var itemUni = (Recipe_Universal)item;
                if (itemUni.IsIgnored) break;
                switch (itemUni.OperationType)
                {
                    case "toHex":
                        text = Calculate_ToHex(text,
                        Delimeter(itemUni.GetDelimeter()), itemUni.GetNumber());
                        break;
                    case "fromHex":
                        text = Calculate_FromHex(text,
                        Delimeter(itemUni.GetDelimeter()));
                        break;
                    case "toDecimal":
                        text = Calculate_ToDecimal(text,
                        Delimeter(itemUni.GetDelimeter()));
                        break;
                    case "fromDecimal":
                        text = Calculate_FromDecimal(text,
                        Delimeter(itemUni.GetDelimeter()));
                        break;
                    case "toBinary":
                        text = Calculate_ToBinary(text,
                        Delimeter(itemUni.GetDelimeter()));
                        break;
                    case "fromBinary":
                        text = Calculate_FromBinary(text,
                        Delimeter(itemUni.GetDelimeter()));
                        break;
                    case "toOctal":
                        text = Calculate_ToOctal(text,
                        Delimeter(itemUni.GetDelimeter()));
                        break;
                    case "fromOctal":
                        text = Calculate_FromOctal(text,
                        Delimeter(itemUni.GetDelimeter()));
                        break;
                    case "toBase64":
                        text = Calculate_ToBase64(text);
                }
            }
        }
    }

```

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				91
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        break;
    case "fromBase64":
        text = Calculate_FromBase64(text);
        break;
    case "md5":
        text = Calculate_HashMD5(text);
        break;
    case "ripemd160":
        text = Calculate_HashRIPEMD160(text);
        break;
    case "sha1":
        text = Calculate_HashSHA1(text);
        break;
    case "sha256":
        text = Calculate_HashSHA256(text);
        break;
    case "sha384":
        text = Calculate_HashSHA384(text);
        break;
    case "sha512":
        text = Calculate_HashSHA512(text);
        break;
    case "rot13":
        text = Calculate_ROT13(text,
itemUni.GetNumber());
        break;
    case "rot47":
        text = Calculate_ROT47(text,
itemUni.GetNumber());
        break;
    case "rot8000":
        text = Calculate_ROT8000(text,
itemUni.GetNumber());
        break;
    case "morseenc":
        text = Calculate_MorseEncode(text);
        break;
    default:
        break;
    }
    break;
case "Diplom.MyInterface.RecipeItems.Recipe_Random":
    var itemRandom = (Recipe_Random)item;
    if (itemRandom.IsIgnored) break;
    switch (itemRandom.GetComboboxType())
    {
        case 0: //Number
            text =
Calculate_RandomNumber(itemRandom.randMin, itemRandom.randMax);
            break;
        case 1: //Hex

```

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				92
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        text =
Calculate_RandomHex(itemRandom.randLength);
        break;
    }
    break;
case "Diplom.MyInterface.RecipeItems.Recipe_Crypto":
    var itemCrypto = (Recipe_Crypto)item;
    if (itemCrypto.IsIgnored) break;
    switch (itemCrypto.OperationType)
    {
        case "encaes":
            text = Calculate_EncryptAES(text,
itemCrypto.Key, itemCrypto.IV, itemCrypto.Mode);
            break;
        case "decaes":
            text = Calculate_DecryptAES(text,
itemCrypto.Key, itemCrypto.IV, itemCrypto.Mode);
            break;
        case "encdes":
            text = Calculate_EncryptDES(text,
itemCrypto.Key, itemCrypto.IV, itemCrypto.Mode);
            break;
        case "decdes":
            text = Calculate_DecryptDES(text,
itemCrypto.Key, itemCrypto.IV, itemCrypto.Mode);
            break;
        case "enctdes":
            text = Calculate_EncryptTDES(text,
itemCrypto.Key, itemCrypto.IV, itemCrypto.Mode);
            break;
        case "dectdes":
            text = Calculate_DecryptTDES(text,
itemCrypto.Key, itemCrypto.IV, itemCrypto.Mode);
            break;
        case "encrc2":
            text = Calculate_EncryptRC2(text,
itemCrypto.Key, itemCrypto.IV, itemCrypto.Mode);
            break;
        case "decrc2":
            text = Calculate_DecryptRC2(text,
itemCrypto.Key, itemCrypto.IV, itemCrypto.Mode);
            break;
    }
    break;
case "Diplom.MyInterface.RecipeItems.Recipe_MorseDecode":
    var itemMorse = (Recipe_MorseDecode)item;
    if (itemMorse.IsIgnored) break;
    switch (itemMorse.Mode)
    {
        case 0: // ua decpde
            text = Calculate_MorseUaDecode(text);
            break;
    }

```

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		93

```

        case 1: // en decode
            text = Calculate_MorseEngDecode(text);
            break;
        }
        break;
    case "Diplom.MyInterface.RecipeItems.Recipe_Vigenere":
        var itemVig = (Recipe_Vigenere)item;
        if (itemVig.IsIgnored) break;
        switch (itemVig.OperationType)
        {
            case "vigenc":
                if (itemVig.Mode == 0)
                    text = Calculate_VigenereEncodeUA(text,
itemVig.Key);
                else
                    text = Calculate_VigenereEncodeENG(text,
itemVig.Key);

                break;
            case "vigdec":
                if (itemVig.Mode == 0)
                    text = Calculate_VigenereDecodeUA(text,
itemVig.Key);
                else
                    text = Calculate_VigenereDecodeENG(text,
itemVig.Key);

                break;
        }
        break;
    case "Diplom.MyInterface.RecipeItems.Recipe_XOR":
        var itemXor = (Recipe_XOR)item;
        if (itemXor.IsIgnored) break;
        text = Calculate_XOR(text, itemXor.Key, itemXor.Mode);
        break;
    default:
        break;
    }
}

TextBoxResult.Text = text;

watch.Stop();
labelTime.Content = $"Час: {watch.ElapsedMilliseconds} мс";
}

```

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		94

Посилання на репозиторій

<https://gitlab.com/TarasHorbatiuk/diploma-crypto-tool>

		Горбатюк Т.М.			ККІКБ.КР.Б – 125 – 05 – ПЗ	Арк.
		Покотило О.А.				95
Змн.	Арк.	№ докум.	Підпис	Дата		