Призначення модуля безпеки даних полягає в забезпечені цілісності, конфіденційності та захищеності даних користувачів від несанкціонованого доступу до даних. Модуль надає можливість забезпечити безпечний канал для оплати послуг програми користувачами.

Загальна структура модуля безпеки даних зображена на рис. 1 та складається з семи модулів, на які вона поділяється за функціональними ознаками:

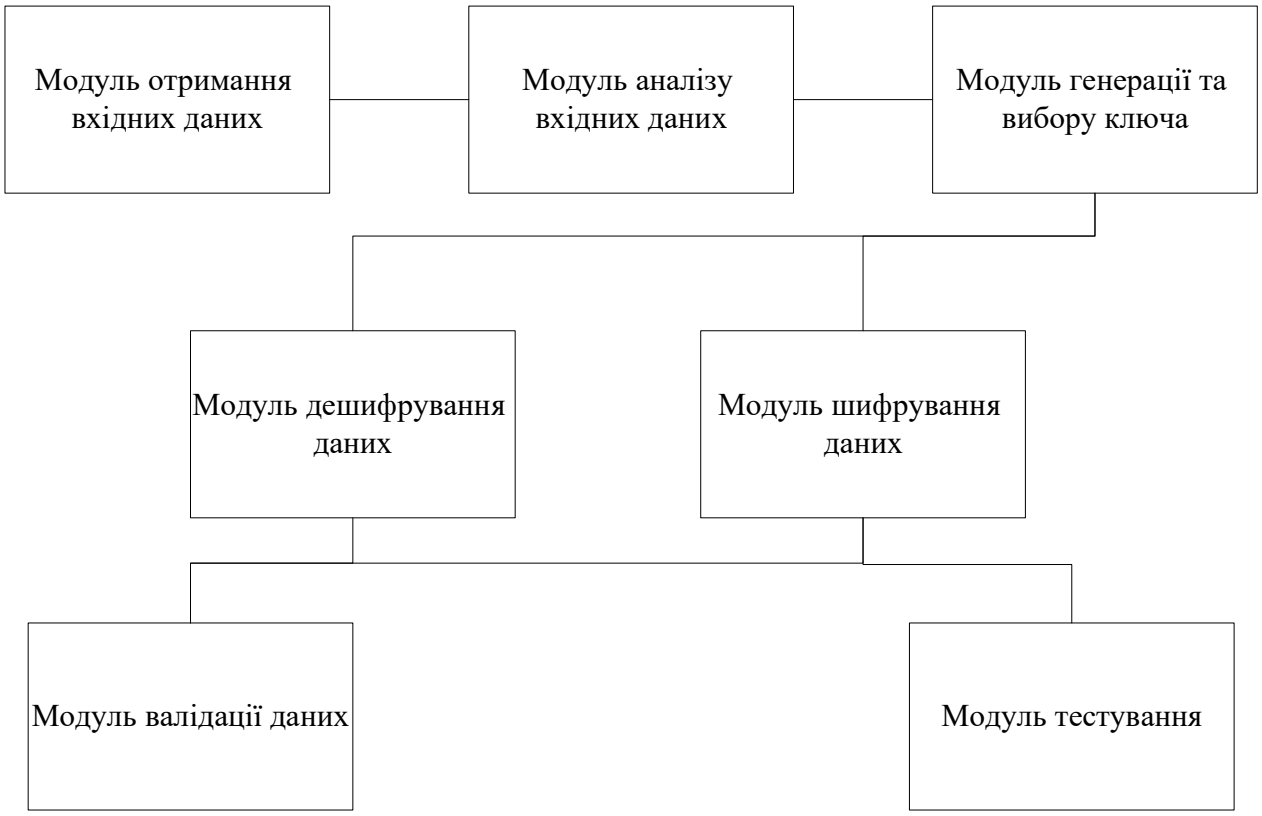

Рис. 1. Структурна схема модуля безпеки даних

1. Модуль отримання вхідних даних – зчитує вхідні дані які ввів користувач, також є можливість зчитування даних з файлу.

2. Модуль аналізу вхідних даних – аналіз даних які передали у програму, вибір оптимального алгоритму шифрування для кожного типу даних.

3. Модуль генерації та вибору ключа – генерує ключ для відповідного алгоритму шифрування. Є можливість генерування ключа різної довжини та збереження у файл ключа, відкриття ключа з файлу.

4. Модуль шифрування даних – шифрує дані оптимізованим алгоритмом які надійшли у програму.

5. Модуль дешифрування даних – дешифрує дані які надійшли у програму.

6. Модуль тестування – заміряє швидкість шифрування та дешифрування даних, при різних розмірах вхідних даних.

7. Валідація даних – модуль визначення відповідності розроблюваного програмного забезпечення.

Функції та призначення модуля безпеки даних

Модуль безпеки даних забезпечує безпеку електронних даних користувачів. Тримання даних в електронному вигляді є дуже зручним, дані особливо великих розмірів не займають багато місця, їх легко копіювати та розповсюджувати. Данні завантажують до мережі інтернет або у хмарні сервіси щоб користувач легко отримав доступ до них з будь якої точки світу. Модулі безпеки використовуються в майже усіх програмах так як кожен бажає щоб його дані залишалися захищеними, конференційними та цілісними, та щоб доступ до них мали лише ті користувачі які мають на це дозвіл. Втрата даних може призвести як до не стабільної роботи програми так і до значних фінансових проблем особливо якщо йде мова про віртуальні сервіси оплати та програми які використовують віртуальну валюту та мають справу з грошовими переводами.

Наприклад, в сервісах де кожен користувач має свій особистий рахунок за допомою якого сплачує певні платні послуги програми, якщо люди не санкціоновано отримають доступ до рахунка вони зможуть викрасти кошти та приватні дані користувача. Також якщо грошові переводи не будуть захищеними то гроші не дійдуть до власників сервісу, в свою чергу постраждає як користувач який не отримує певну послугу та втратив свої конфіденційні данні так і власники сервісу які не в змозі забезпечити надійність, та приватності даних що призведе до не довіри користувачів цьому сервісу та фінансових збитків.

Мій модуль безпеки даних представлений у вигляді DLL бібліотеки та у вигляді візуалізованої програми. DLL бібліотека, надає змогу зручно використовувати програму в інших проектах. Бібліотеку зручно використовувати та підключати до модулів інших програм, що є однією з переваг при виборі модуля безпеки який планують підключити до програми, що значно полегшуй використання модуля безпеки і не потребую значних зусиль для підключення та використання.

Також я представив мій розроблений модуль безпеки у візуальному вигляді. Це зручно демонструє користувачу можливості модуля. Користувач може шифрувати та дешифрувати дані, встановлюючи та використовуючи різні налаштування для модуля безпеки. Є можливість перевірки швидкості шифрування та дешифрування залежно від розміру даних та їх типу. Також можливо завантажувати фали які потрібно шифрувати з файлу або вводити данні в ручну у саму програму це залежить від об’єму самих даних, якщо данні великого розміру то зручніше та швидше буде їх завантажити з файлу.

Для забезпечення конфіденційності даних, їх треба шифрувати. Навіть якщо зловмисник скопіює або отримує доступ до даних, він не зможе скористатися цією інформацією так як вона зашифрована за допомогою модифікованого алгоритму з використанням двох ключів. Це забезпечує конфіденційність даних користувача та дозволить запобігти їх подальше використання.

Ще одна функція модулю безпеки це шифрування об’єктів класу. Об’єкти класу які використовуються в декількох програмах або їх просто зберігають або відправляють по мережі інтернет вони займають багато місця, використовують більше трафіку при їх передачі. Шифрування цих об’єктів за допомогою модифікованого алгоритму по перше забезпече безпеку даних які записані в об’єкт класу, по друге за рахунок перебору полів класу та переводу в байт коди виконується шифрування та стиснення виликих за розміром даних що в свою чергу зменшує розмір самих об’єктів класу и вони займають менше

місця при зберіганні на жорстких носіях, флеш-накопичувач та під час передачі в мережі інтернет.

Центральним алгоритмом, що забезпечує безпеку даних є алгоритм шифрування, що дозволяє зашифрувати дані, забезпечує захищеність та конфіденційності даних від не санкціонованого доступу осіб які не мають на це право. Також алгоритм який стискає дані для зменшення їх розміру.

Детально алгоритм покроково описано нижче після опису основних функцій алгоритму шифрування, які приймають участь у роботі алгоритму.

1. String_enc – розділяє вхідні дані на рядки та символи.
2. To_hx – перетворення числа у шістнадцятковий вигляд.
3. To_bin– перетворення числа у двійковий вигляд.
4. Header_key – якщо головний ключ не відкритий з файлу, то програма надає ключ, з можливістю вибору довжени ключа шифрування трьох фіксованих розмірів: 128, 192 і 256 бітів та записує ключ у масив.
5. Number_of_element – отримання номеру певного рядка та стовпця масиву головного ключа для Header_key.
6. Second_key – вибір другого ключа залежно від даних які шифрує програма.
7. ShiftRows – функція яка виконує циклічний зсув вліво всіх рядків масиву даних, за винятком нульовий.
8. MixColumns – виконує множення по модулю кожного стовпця масиву даних
9. Add_Round_Key – виконує накладення на масив даних ключа а саме на певні стовпці масиву.
10. Squeezes – функція яка займається стисканням розміру даних.
11. OpenF – відкриття файлу який потрібно зашифрувати.
12. OpenKey – відкриття головного ключа та перевірка коректності ключа для шифрування даних.
13. SaveF – збереження зашифрованого файлу.

Покрокова реалізація алгоритму

На самому початку реалізації алгоритму шифрування ми отримуємо вхідні данні які потрібно зашифрувати. Дані вибирають залежно від налаштувань які користувач встановив у візуалізованій програмі. Користувач має можливість завантажити дані з файлу чи самостійно ввести дані. Отримані дані записують у рядок. В алгоритмі шифрування який представив у вигляді DLL бібліотеки вхідні дані можна передати лише у вигляді рядка попередньо підключивши бібліотеку до проекту.

Після отримання вхідних даних, які записані у рядок, їх за допомогою функції String_enc розділяють поелементно и записують у масив 4 на 4, якщо елементів String_enc більше максимальної кількості елементів одного масиву то створюється ще один масив такого ж розміру 4 на 4. У випадку, коли дані не повністю заповнюють масив, пусті комірки масиву заповнюються числом 0. Кожен елемент масиву замінюється на відповідне число з таблиці ASCII. Наступним крок це за допомогою функції To_hx переводимо кожне число з таблиці ASCII яке записане у масив у шістнадцятковий вигляд. Після того як вхідні данні представили в потрібному вигляді можна переходити до їх шифрування.

Подальший крок є вибір головного ключа за допомогою функції Header_key, в програмі яка представлена в візуальному вигляді користувач може вибрати довжину ключа, йому на вибір довжини ключа шифрування надається один з трьох фіксованих розмірів: 128, 192 і 256 бітів. Користувач також може завантажити свій ключ з файлу. По замовченню стоїть ключ розміром 256 байт, якщо користувач в налаштуваннях не виставить інший розмір. Головний ключ представляє собою 256 шістнадцяткових символів які не повторюються між собою. За допомогою функції OpenKey користувач може завантажити свій ключ, ця функція також перевіряє коректність ключа. Після вибору головного ключа, ключ записується у масив відповідного розміру, наприклад якщо ключ розміром 256 байт то записується у масив розміром 16 на 16.

Наступний крок, виконуємо табличну заміну символу масиву вхідних даних, відповідними даними масиву головного ключа відповідно.

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Рис. 2. Приклад масиву головного ключа розміром 256 байт.

Наприклад візьмемо масив головного ключа з 256 шістнадцятковими символами (рис. 2.), и один символ вхідних даних у шістнадцятковому вигляді, наприклад “3c”. Для виконання заміни треба знайти 3 рядок та стовпець який позначений літерою “c”. Тоді X=3, Y=c, потім шукаємо комірку де перетинаються наші X та Y відповідно, вони перетинаються в комірки з шістнадцятковим символом “eb”. Таким чином замінюємо символ “3c” на “eb”. Заміну повторюється для кожного байту масиву вхідних даних.

Подальшим кроком виконується зсув рядків масиву за допомогою функції ShiftRows. Функція виконує циклічний зсув вліво всіх рядків масиву даних (рис. 3.).

Рис. 3. Операція ShiftRows.

Зсув рядка виконується за відповідним алгоритмом, для нульового рядка зсув не виконуємо, для наступних рядків виконуємо зсув вліво на кількість біт яка дорівнює номеру рядка, наприклад : рядок з індексом 1 зсуваємо вліво на 1 біт, рядок з індексом 2 зсуваємо на 2 біта вліво і так далі.

Після виконання зсуву рядків, виконуємо множення стовпців за допомогою функції MixColumns (рис. 4.). Функція виконує множення на фіксований поліном A(X):

$$A(X) = 3\times3+X\times2+X+2;$$

Де X це номер стовпця на який ми виконуємо множення.

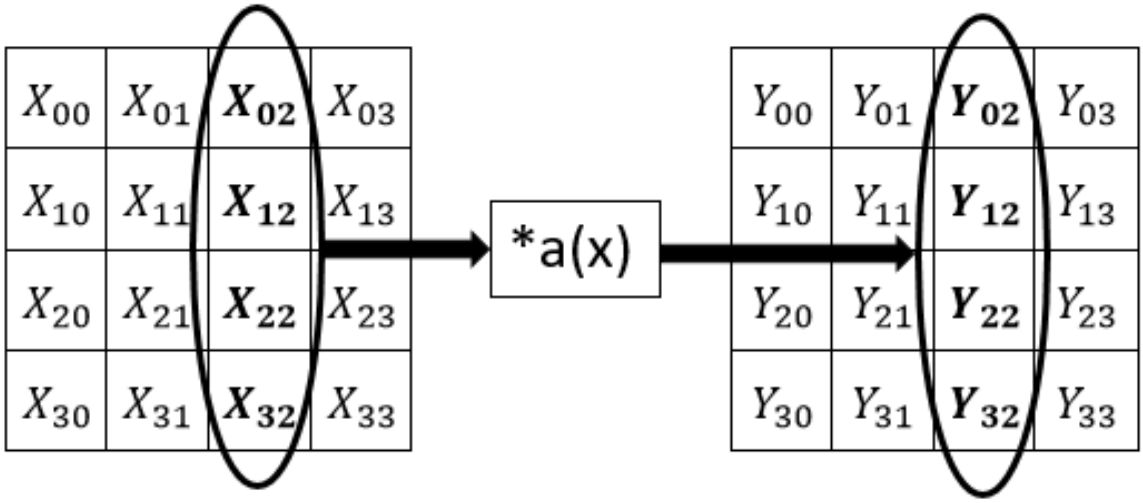

Рис. 4. Операція MixColumns.

Наступний крок виконує функція Add_Round_Key (рис. 5.). Функція виконує накладення на масив даних ключа, крім елемента нульового стовпця та нульової колонки.

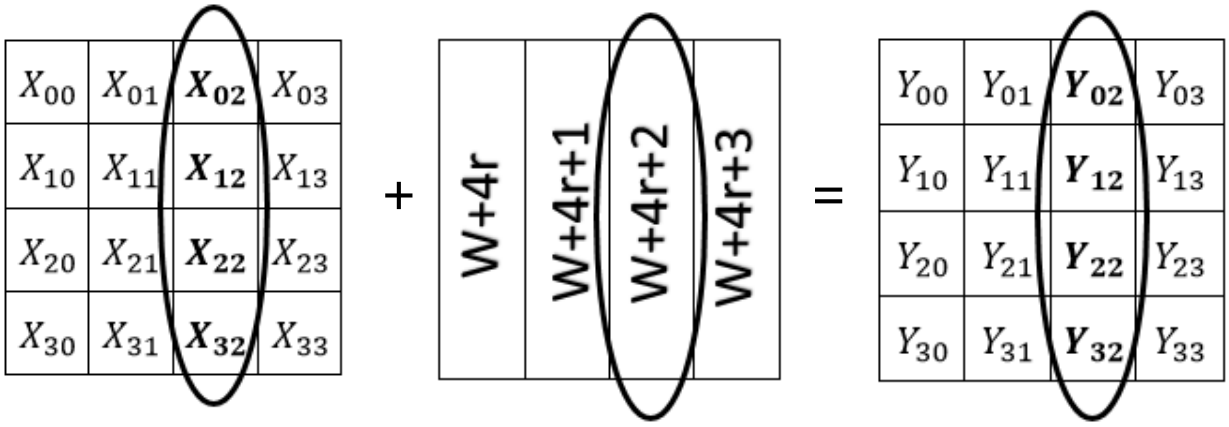

Рис. 5. Операція Add_Round_Key.

А саме, на почергово на кожен стовпець масиву даних побітової логічною операцією «виключне або» (XOR) накладається певне слово розширеного ключа W + 4r + i, де r- номер поточного раунду алгоритму, починаючи з 1; i- номер поточного стовпця; W- значення другого ключа, яке отримується з нульового стовпця та нульової колонки поточного масиву даних. Перед накладанням розширеного ключа поточний масив даних переводиться у двійковий вигляд. Кожний розширений ключ який буде накладаться на масив даних, перед самим виконанням логічної операції «виключне або» (XOR), переводиться у двійковий вигляд за допомогою функції To_bin. Після накладання на усі стовпці розширеного ключа усі дані масиву переводимо з двійкового вигляду назад у шістнадцятковий вигляд.

Ці всі виконані операції вважаються за один раунд алгоритму. В наступних раундах виконуються всі операції окрім операції MixColumns.

Таблиця 2.

Кількість раундів залежно від довжини ключа

Розмір головного ключа в бітах	Кількість раундів
128	10
192	12
256	14

Кількість раунду визначаються залежно від довжини головного ключа (Таблиця. 2.) та значення додаткового ключа. Наприклад розмір головного ключа в бітах 256, а додатковий ключ дорівнює “2d”, переводимо додатковий ключ у десятковий вигляд “2d” = 45, далі беремо першу цифру отриманого числа це 4. Після цього додаємо значення головного ключа до отриманого значення і отримаємо кількість раундів 14+4=18.

Після виконання усіх раундів алгоритму виконуємо побітове стиснення даних. Ця функція дуже корисна для даних які мають великі розміри. Наступним кроком за допомогою функції To_bin переводимо кожен елемент масиву даних у двійковий вигляд. Дані масиву які ми отримали ми записуємо у рядок один елемент за іншим и отримуємо рядок закодованих даних представлених у вигляді двійковому чисел. Далі за допомогою функції Squeezes виконуємо побітове стиснення даних. Алгоритм стиснення даних фективних при великих обсягах даних, в кращому випадку цей алгоритм стискає файл в 64 рази, в гіршому збільшує на 1/128.

Закодовані данні виводяться на екран візуалізованої програми, також є функція збереження цих даних у файл. В DLL бібліотеці повертає значення лише у вигляді закодованого рядка.

Модуль безпеки даних представлений у вигляді програми з візуальним інтерфейсом та у вигляді DLL бібліотеки.

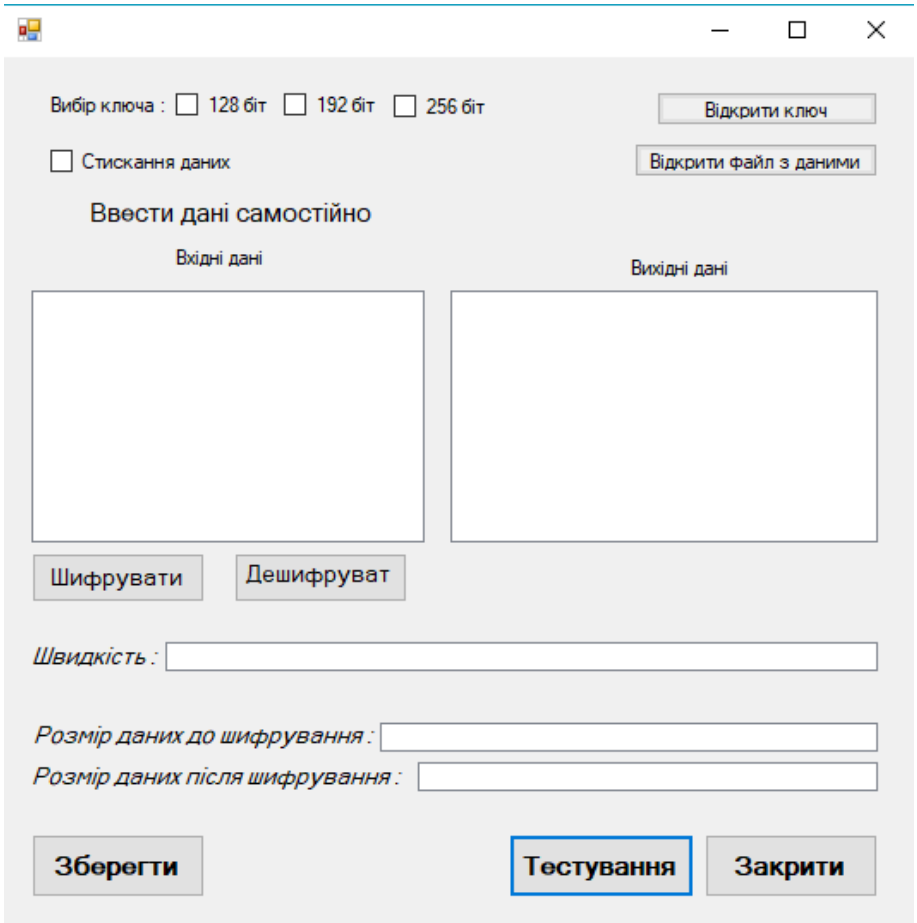

Рис. 6. Візуалізація програми.

Візуалізована програма шифрування та дешифрування – програма представлена з візуальним інтерфейсом на windows form з коритувацьким інтерфейсом з низкою налаштувань які може змінювати користувач залежно від своїх потреб (рис. 6.). Програму реалізовано в такому вигляді для зручності користування та наочності роботи самої програми. Там ми можемо побачити доступні до використання наступні опції :

1. **Вибір ключа** – вибір розміру головного ключа в бітах. Розмір ключа можливо вибрати лише одного з трьох видів. Ключ довжиною 128 біт, 192 біт або 256 біт. За замовченням встановлення 256 біт.
2. **Кнопка відкриття ключа** – можливість завантаження та подальше використання для шифрування та дешифрування свого ключа, якщо він підходить до вимог програми.
3. **Стискання даних –** після виконання алгоритму шифрування, якщо вибраний цей параметр виконує стискання даних. Ця опція особливо корисна, якщо шифруються данні великих розмірів. Також ця функція корисна, якщо виконуються шифрування об’єктів класу, що в свою чергу дозволить зменшити розмір файлу і зменшить пам'ять яку займають об’єкти класу на електронних носіях пам’яті.
4. **Кнопка відкрити файл з даними –** надає можливість користувачу відкривати файли з даними для подальшої роботи з ними.
5. **Вхідні дані –** якщо вхідні дані відкриті з файлу то в цьому полі виводяться дані з файлу. Якщо дані не відкриті, то користувач самостійно повинен ввести дані в це поле.
6. **Вихідні дані –** дані які демонструють результат шифрування або дешифрування.
7. **Кнопка шифрування** – виконує шифрування даних.
8. **Кнопка дешифрування** – виконує дешифрування даних.
9. **Кнопка зберегти –** зберігає результат роботи програми у файл у дерикторію яку вибирає користувач.

10. **Кнопка тестування** – запускає блок тестування можливостей програми.
11. **Кнопка закрити** – завершує роботу програми.
12. **Швидкість** – демонструє швидкість роботи алгоритму шифрування або дешифрування.
13. **Розмір даних до та після шифрування** – демонструє початковий та кінцевий розмір даних після виконання алгоритму шифрування.

Блок тестування – призначений для тестування та демонстрування можливостей програми шифрування та дешифрування залежно від встановлених певних налаштувань користувачем (рис. 7.). Блок відкривається у новому вікні і не впливає на роботу основної програми.

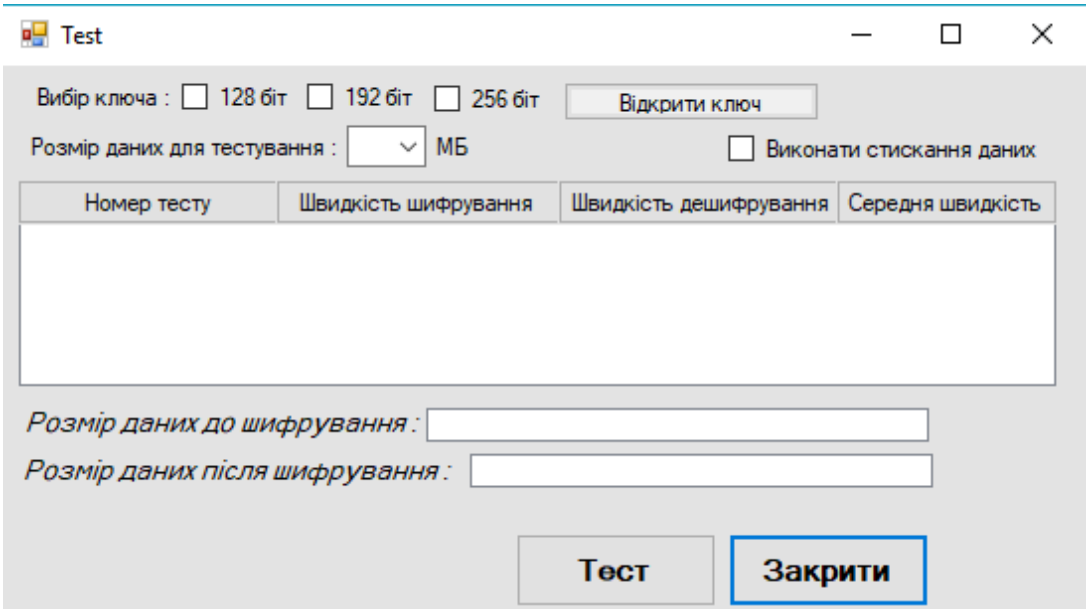

Рис. 7. Тестування роботи програми.

В блоці тестування можемо побачити доступні до використання наступні опції :

1. **Вибір ключа** – вибір розміру головного ключа в бітах. Розмір ключа можливо вибрати лише одного з трьох видів. Ключ довжиною 128 біт, 192 біт або 256 біт. За замовченням встановлення 256 біт.

2. **Кнопка тест –** розпочинає тестування.
3. **Кнопка відкриття ключа** – можливість завантаження та подальше використання для шифрування та дешифрування свого ключа якщо він підходить до вимог програми.
4. **Стискання даних –** після виконання алгоритму шифрування, якщо вибраний цей параметр виконує стискання даних. Ця опція особливо корисна, якщо шифруються данні великих розмірів. Також ця функція корисна, якщо виконуються шифрування об’єктів класу, що в свою чергу дозволить зменшити розмір файлу і зменшить пам'ять яку займають об’єкти класу на електронних носіях пам’яті.
5. **Розмір даних для тестування –** користувач має можливість вибрати розмір вхідних даних над якими буде проводитися тестування.
6. **Розмір даних до та після шифрування –** демонструє початковий та кінцевий розмір даних після виконання алгоритму шифрування.
7. **Кнопка закрити –** завершує роботу програми.
8. **Поле виводу результату –** поле в яке виводиться результат роботи програми згідно встановлених параметрів. В це поле виводиться порядковий номер тесту, швидкість шифрування та дешифрування кожного тесту та їх середню швидкість.

Програма представлена у вигляді DLL бібліотеки – при використанні бібліотеки в інших проектах перш за все потрібно підключити її до проекту в якому планують подальше використання DLL бібліотеки. Для використання алгоритмів шифрування та дешифрування потрібно спочатку звернутися до класу My_library, потім звернутися до функцій цього класу My_Decryption або My_Encryption і передавши до них рядок даних які потрібно шифрувати або дешифрувати відповідно, потім передати до My_library користувацькі налаштування які в подальшому буде використовувати DLL бібліотека. Після цього функція поверне результат роботи у вигляді рядка. В хмарному сервісі організації роботи з відеопотоками у реальному часі, використовується програма представлена у вигляді DLL бібліотеки.