МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМ. В. ДАЛЯ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЕЛЕКТРОНІКИ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК ТА ІНЖЕНЕРІЇ

До захисту допускається
Завідувач кафедри
_____ І.С. Скарга-Бандурова
«____»_____ 2020 р.

ДИПЛОМНИЙ ПРОЕКТ (РОБОТА) БАКАЛАВРА
ПОЯСНЮВАЛЬНА ЗАПИСКА

НА ТЕМУ:

«Програмний комплекс криптографічного перетворення інформації на основі симетричного шифрування»

Освітнступінь «бакалавр»
Спеціальність 123 «Комп'ютерна інженерія»

Науковий керівник роботи: _____ Кардашук В. С.
(підпис) (ініціали, прізвище)

Консультант з охорони праці: _____ Критська Я. О.
(підпис) (ініціали, прізвище)

Здобувач вищої освіти: _____ Медвєдєв Б. М.
(підпис) (ініціали, прізвище)

Група: КІ-16 бд

Сєвєродонецьк 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет Інформаційних технологій та електроніки
Кафедра Комп'ютерних наук та інженерії
Освітній ступінь бакалавр
Спеціальність 123 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ:

Т.в.о. завідувача кафедри КНІ
_____ С.О. Сафонова
«_____» _____ 2020 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТУ

Медвєдєву Богдану Михайловичу
(прізвище, ім'я, по-батькові)

1. **Тема проекту (роботи):** «Програмний комплекс криптографічного перетворення інформації на основі симетричного шифрування» затверджена наказом по університету № 73/15.15 від «30» квітня 2020 р.

2. Строк здачі студентом закінченого проекту (роботи): 10.06.2020 р.

3. Вихідні дані проекту (роботи): матеріали переддипломної практики

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити):

1. Огляд методів симетричного шифрування інформації.
2. Криптографічні алгоритми.
3. Симетричні криптосистеми та потокові шифри.
4. Розроблення програмного комплексу для шифрування інформації.
5. Охорона праці.

5. Перелік графічного матеріалу (з точною назвою обов'язкових креслень):

Електронні плакати

6. Консультанти роботи, з вказівкою розділів, що до них відносяться

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Основна частина	Кардашук В.С., к.т.н., доц.		
Охорона праці	Критська Я.О., ст.викл.		

7. Дата видачі завдання _____

Керівник _____ Кардашук В. С.

(підпис)

Завдання до виконання прийняв _____ Медвєдєв Б. М.

(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітки
1.	Отримання завдання, збір матеріалів	18.05.20 – 24.05.20	
2.	Огляд літератури й обґрунтування необхідності розроблення	25.10.20 –28.10.20	
3.	Огляд методів симетричного шифрування інформації	29.05.20 – 28.05.20	
4.	Створення програмного комплексу для шифрування інформації	28.05.20 –31.05.20	
5.	Налагодження програми	03.06.20 – 04.06.20	
6.	Оформлення пояснювальної записки	05.06.20 – 08.06.20	
7.	Підготовка та подання магістерської роботи до захисту	09.06.20 – 10.06.20	

Здобувач вищої освіти _____ Медведєв Б. М.
(підпис) (ініціали, прізвище)

Керівник _____ Кардашук В. С.
(підпис) (ініціали, прізвище)

РЕФЕРАТ

Пояснювальна записка до дипломного проекту: 120 сторінок, 20 рисунків, 10 таблиць, 23 джерел посилань, 7 додатків на 40 сторінках.

У дипломному проекті розроблено програмний комплекс криптографічного перетворення інформації на основі симетричного шифрування.

Проведений огляд та аналіз методів симетричного шифрування інформації, потокових та блочних шифрів. За результатами дослідження сформульовані мета та завдання дипломного проекту.

Здійснена постановка задачі на розробку програмного комплексу криптографічного перетворення інформації на основі симетричного шифрування, організаційна побудова та програмна реалізація основних операцій блокового шифруваня.

Проведено аналіз продуктивності програмної реалізації блокового симетричного шифру

Розглянуті питання та сформульовані рекомендації щодо охорони праці в умовах виробництва.

КРИПТОГРАФІЯ, КЛЮЧ, ШИФРУВАННЯ, АЛГОРИТМ, НАЛАШТУВАННЯ, ПОТОЧНИЙ ШИФР, БЛОКОВИЙ ШИФР.

Умови отримання дипломного проекту:
СНУ ім. Володимира Даля, пр. Центральний 59а, м. Сєвєродонецьк, 93406.

ЗМІСТ

ВСТУП

XXI століття – епоха інформатики та інформатизації, основною цінністю є інформація. Технологія дає можливість передавати і зберігати все більші обсяги інформації. Це благо має й зворотний бік. Інформація стає значно вразливішою з різних причин:

– зростання обсягів збережених і переданих даних;

– розширення кола користувачів, що мають доступ до інформаційних систем;

– ускладнення інформаційних систем.

Напрям розроблення системи стандартів із захисту даних, що зберігаються і обробляються на персональних комп'ютерах, від несанкціонованого доступу визнано однією із найпріоритетніших областей.

Поширення інформаційних технологій вимусило до необхідності інтеграції в них все більш стійких механізмів безпеки, що неможливо без використання надійних криптографічних алгоритмів і це призводить до того, що криптографія проникає у всі сфери інформаційних технологій. Внаслідок такої ситуації, до недавнього часу, з причини браку інформації було важко визначити ступінь надійності криптографічних алгоритмів, та, навіть, знайти їх описи. Це призводило до використання різних примітивних методів криптографічного захисту чи до створення абсолютно ненадійних алгоритмів.

На сьогоднішній день існує величезна кількість криптографічних алгоритмів, що відрізняються як своїми загальними характеристиками, так і принципами, на яких базується їх робота. Не всі вони є однаково надійними – серед них є навіть такі, що оформлені як стандарти та при цьому не забезпечують скільки–небудь реального захисту. Насправді ж, створення надійного криптографічного алгоритму – дуже важка задача. Крім того, надійність є відносна річ – багато з раніше розроблених алгоритмів, які вважалися надійними, тепер або ненадійні, або ця надійність викликає

великий сумнів. Тому при розробці криптографічного алгоритму необхідно враховувати тенденції розвитку комп'ютерної техніки а також інші фактори, що потенційно можуть знизити його стійкість в майбутньому. Традиційний метод шифрування, що застосовується при організації захисту даних, які пересилаються інформаційно–обчислювальними мережами, базується на факті знання і використання відправником та адресатом повідомлення одного й того ж секретного ключа. Ідея цього, так званого методу криптографії з секретним ключем або симетричної криптографії, полягає в тому, що відправник використовує секретний ключ для того, щоб зашифрувати текст повідомлення, а адресат застосовує цей же ключ для його розшифрування. У випадку компрометації ключа стає можливим несанкціонований доступ до даних, тому для надійного функціонування системи необхідна його періодична заміна. Важливим питанням є спосіб конфіденційного узгодження обома сторонами ключів до використання. Якщо відправник та адресат розділені значними відстанями в просторі, постає питання надійності засобів зв'язку, що ними пересилаються секретні дані: будь–хто, випадково або цілеспрямовано перехопивши секретний ключ, зможе безконтрольно читати, змінювати чи фальсифікувати всі супроводжувані цим ключем повідомлення. Сукупність заходів по створенню (генерації), передачі та зберіганню ключів носить назву адміністрування ключів (АК) і особливо важко забезпечується у відкритих системах з великою кількістю користувачів. Для вирішення проблем, пов'язаних із АК, в 1976 році Уітфілдом Діффі та Мартіном Хелманом була запроваджена концепція системи шифрування з відкритим ключем, яка виключає необхідність відправнику та адресату ділитись секретною інформацією. Кожна зі сторін має свою персональну пару ключів, пов'язаних між собою певною математичною залежністю: відкритий, що публікується і використовується для пересилання повідомлень, і секретний що його використовують для розшифрування прийнятих даних і зберігають у таємниці. Для пересилання конфіденційного повідомлення відправник шифрує його текст за допомогою

відкритого ключа адресата, після чого відсилає за місцем призначення, а адресат розшифровує отриману інформацію, застосовуючи свій секретний ключ.

Таким чином, будь–хто може прийняти "чуже" повідомлення, але тільки безпосередній адресат може його прочитати. Генерація ключів в обох системах, як правило, здійснюється за допомогою генераторів псевдовипадкових чисел (ПВЧ), що повинні забезпечувати вихідну послідовність із достатньо довгим циклом повторювання. Це необхідно для виключення можливості появи повторюваних блоків вихідної послідовності, а відтак, її прогнозованості. Одніє з переваг системи шифрування з відкритим ключем є відсутність необхідності покладання на безпеку засобів комунікацій, оскільки між абонентами передаються лише відкриті ключі. Єдина вимога до системи полягає в підборі пари ключів таким чином, щоб секретний ключ неможливо було вирахувати з відкритого. Слід також взяти до уваги забезпечення системою можливості електронного підпису документів, коли адресатові надається додаткова можливість перевірки аутентичності та цілісності одержаного повідомлення, тобто факту, що документ передано в незмінній формі саме від означеного відправника. Недоліком в порівнянні з системою шифрування з секретним ключем є недостатня швидкість функціонування методу. Слід відзначити, що, попри доцільність окремого застосування кожного із методів при забезпеченні певних типів захисту інформації, в інформаційно–обчислювальних мережах часто застосовується комбінація обох систем для сполучення таких їх позитивних якостей, як швидкість обробки секретних ключів та безпека передачі відкритих. Реалізація такого протоколу "цифрового конверта" передбачає шифрування відкритим ключем секретного, який, в свою чергу, використовується для шифрування тексту повідомлення. Таким чином, обидва підходи при застосуванні в розподілених інформаційних середовищах є взаємодоповнюючими в забезпеченні конфіденційного обміну інформацією.

1 ОГЛЯД МЕТОДІВ СИМЕТРИЧНОГО ШИФРУВАННЯ ІНФОРМАЦІЇ

1.1 Криптографічні алгоритми

Сучасні технології використання комп’ютерної техніки зробило можливим появу складних шифрів. Більше того, комп'ютери дозволяли шифрувати будь–які дані, які можна представити в комп'ютері у двійковому виді, на відміну від класичних шифрів, які розроблялись для шифрування письмових текстів. Це зробило непридатними для застосування лінгвістичні підходи в криптоаналізі. Багато комп'ютерних шифрів можна характеризувати за їхньою роботою з послідовностями бінарних бітів (інколи в блоках або групах), на відміну від класичних та механічних схем, які, зазвичай, працюють безпосередньо з літерами. Однак, комп'ютери також знайшли застосування у криптоаналізі, що, в певній мірі, компенсувало підвищення складності шифрів. Тим не менше, гарні сучасні шифри залишались попереду криптоаналізу; як правило, використання якісних шифрів дуже ефективне (тобто, швидке і вимагає небагато ресурсів), в той час як злам цих шифрів потребує набагато більших зусиль ніж раніше, що робить криптоаналіз настільки неефективним та непрактичним, що злам стає практично неможливим [1].

Залежно від розміру блоку інформації криптоалгоритми поділяються на:

1. Потокові шифри, одиницею кодування в яких є один біт. Результат кодування не залежить від минулого раніше вхідного потоку. Схема застосовується в системах передачі потоків інформації, тобто в тих випадках, коли передача інформації починається і закінчується в довільні моменти часу і може випадково перериватися. Найбільш поширеними представниками потокових шифрів являються скремблери.

2. Блочні шифри, одиницею кодування в яких є блок з декількох байтів (в даний час 4–32). Результат кодування залежить від усіх вихідних байтів цього блоку. Схема застосовується при пакетній передачі інформації та кодування файлів.

На рис. 1.1. представлені криптографічні алгоритми та місце симетричного шифрування у цій класифікації [2].

Рисунок 1.1 – Класифікація криптографічних алгоритмів

1.2 Симетричні криптосистеми

Симетричні криптосистеми – спосіб шифрування, в якому для шифрування і дешифрування застосовується один і той же криптографічний ключ. До винаходу схеми асиметричного шифрування єдиним існуючим способом було симетричне шифрування. Ключ алгоритму повинен зберігатися в секреті обома сторонами [3].

Принцип роботи симетричних алгоритмів полягає в тому, що для зашифровки і розшифрування повідомлення використовується один і той же ключ (блок інформації).

Вся багатогранність існуючих симетричних криптографічних методів можна звести до наступних класів перетворень (рис. 1.2) [1].

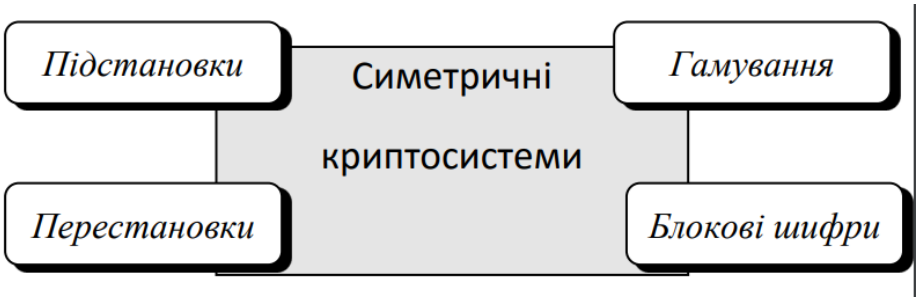


Рисунок 1.2 – Методи перетворення інформації в симетричних системах

Алгоритми шифрування і дешифрування даних широко застосовуються в комп'ютерній техніці в системах приховування конфіденційної і комерційної інформації від не коректного використання сторонніми особами. Головним принципом у них є умова, що особа яка приймає повідомлення, заздалегідь знає алгоритм шифрування, а також ключ до повідомлення, без якого інформація є всього лише набір символів, що не мають сенсу. Симетричні криптоалгоритми виконують перетворення невеликого (1 біт або 32–128 біт) блоку даних в залежності від ключа таким чином, що прочитати оригінал повідомлення можна тільки знаючи цей секретний ключ.

1.3 Потокові шифри

Головною відмінною рисою потокових шифрів є побітна обробка інформації. Як наслідок, шифрування і дешифрування в таких схемах може обриватися в довільний момент часу, як тільки з'ясовується, що потік що передається перервався, і також відновлюється при виявленні факту продовження передачі.

Подібна обробка інформації може бути представлена у вигляді автомата, який на кожному своєму такті [2]:

– генерує за будь–яким законом один біт шифрувальної послідовності;

– яким–небудь оборотним перетворенням накладає на один біт відкритого потоку даний шифрувальний біт, отримуючи зашифрований біт.

Всі сучасні потокові шифри діють за даною схемою. Біт шифрування, що виникає на кожному новому кроці автомата, як втім і цілий набір таких біт, прийнято позначати символом γ (гамма), а самі потокові шифри отримали через це друга назва – шифри гамування. Шифри гамування набагато швидші за своїх найближчих конкурентів – блокових шифрів – у тому випадку, якщо потокове шифрування реалізується апаратно. Як ми побачимо, базові схеми шифрів гамування влаштовані просто і як би "самі просяться" в апаратну реалізацію – це й не дивно, якщо взяти до уваги історію та основну мету їх створення.

У тих же випадках, коли з яких–небудь причин дані алгоритми реалізовані програмно, їх швидкість порівняна з блочними шифрами, а іноді і набагато нижче за них. Трьома основними компонентами, над якими обчислюється функція, що породжує гаму, є:

– ключ;

– номер поточного кроку шифрування;

– прилеглі від поточної позиції біти вихідного і/або зашифрованого тексту.

Ключ є необхідною частиною гаммуючого шифру. Якщо ключ і схема породження гами не є секретним, то поточний шифр перетворюється на звичайний перетворювач–кодер – скремблер (від англ. Scramble – перемішувати, збивати).

1.4 Блочні шифри

На сьогоднішній день розроблено досить багато стійких блокових шифрів. Практично всі алгоритми використовують для перетворень певний набір бієктивних (оборотних) математичних перетворень. Характерною особливістю блокових криптоалгоритмів є той факт, що в ході своєї роботи

вони виробляють перетворення блоку вхідної інформації фіксованої довжини і отримують результуючий блок того ж обсягу, але недоступний для прочитання стороннім особам, що не володіють ключем.

Блочні шифри є основою, на якій реалізовані практично всі криптосистеми. Методика створення ланцюжків із зашифрованих блочними алгоритмами байт дозволяє шифрувати ними пакети інформації необмеженої довжини.

Таку властивість блокових шифрів, як швидкість роботи, використовується асиметричними криптоалгоритмами, повільними за своєю природою. Відсутність статистичної кореляції між бітами вихідного потоку блочного шифру використовується для обчислення контрольних сум пакетів даних і в хешуванні паролів.

Криптоалгоритм іменується ідеально стійким, якщо прочитати зашифрований блок даних можна тільки перебравши всі можливі ключі, до тих пір, поки повідомлення не виявиться осмисленим. Так як по теорії ймовірності шуканий ключ буде знайдено з ймовірністю 1/2 після перебору половини всіх ключів, то на злом ідеально стійкого криптоалгоритму з ключем довжини N потрібно в середньому перевірок [3].

Таким чином, у загальному випадку стійкість блочного шифру залежить тільки від довжини ключа і зростає експоненціально з його збільшенням. Навіть припустивши, що перебір ключів проводиться на спеціально створеній багатопроцесорної системі, в якій завдяки діагональному паралелізму на перевірку 1 ключа йде тільки 1 такт, то на злом 128 бітного ключа сучасній техніці буде потрібно не менше 10^{21} років. Звичайно, все сказане стосується тільки ідеально стійких шифрів.

1.5 Моноалфавітна підстановка

Кодування інформації, що засноване на моноалфавітній підстановці – це один з самих древніх шифрів. Суть методу полягає с втому , що перш за

все вибирається нормативний алфавіт, тобто набір символів, які використовуватимуться при складанні повідомлень, що вимагають зашифровування. Потім вибирається алфавіт шифрування і встановлюється взаємно однозначна відповідність між символами нормативного алфавіту і символами алфавіту шифрування.

Щоби зашифрувати вихідне повідомлення, кожен символ відкритого тексту замінюється на відповідний йому символ алфавіту шифрування (табл. 1.1).

Таблиця 1.1 – Таблиця моноалфавітної підстановки

Нормативний алфавіт	а	б	в	г	д	є
Алфавіт шифрування	я	п	р	л	к	х

Метод моноалфавітної підстановки можна представити як числові перетворення символів вихідного тексту (табл. 1.2).

Таблиця 1.2 – Таблиця числової підстановки

Нормативний алфавіт	а	б	в	г	д	є
Алфавіт шифрування	7	9	6	3	0	2

Моноалфавітні підстановки можна описати виразом [4]:

$$Ei = (Mi + Si) \bmod L, \qquad (1.1)$$

де *Ei, Mi* – числові еквіваленти символів алфавіту шифрування і нормативного алфавіту відповідно;

Si – коефіцієнт сзуву;

L – потужність алфавіту.

1.6 Шифр Цезаря

Простим прикладом моноалфавітних підстановок є шифр Цезаря. У цьому шифрі кожен символ відкритого тексту замінюється третім після нього символом в алфавіті, замкнутому в кільце, тобто після пропуску слідує буква — «А».

Таким чином, шифр Цезаря описується наступним виразом [5]:

$$Ei = (Mi + S) \bmod L, \qquad (1.2)$$

де *Ei, Mi* – числові еквіваленти символів криптограми і відкритого тексту відповідно;

Si – коефіцієнт сзуву, однаковий для всіх символів;

L – потужність нормативного алфавіту.

Цезарь використовував величину зсуву S=3, але, звичайно, можна використовувати будь–яке ціле S: 1 ≤ S ≤ (L–1) (рис. 1.3) [5].

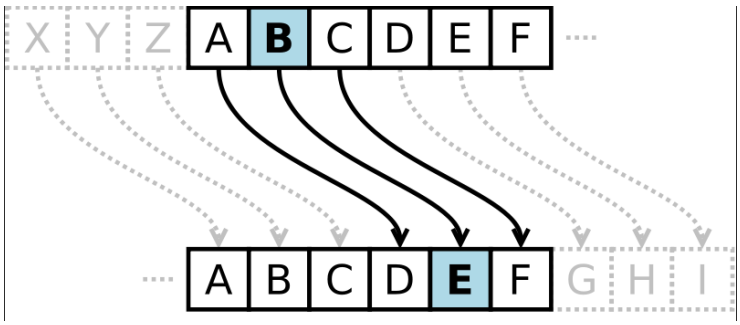


Рисунок 1.3 – Графічне зображення реалізації шифру Цезаря

Частотний аналіз криптографічного перетворення інформації показує, що всі природні мови мають характерний частотний розподіл символів. Наприклад, голосні букви зустрічаються в українській мові частіше за приголосні, а буква —Ф – найрідша. Моноалфавітні підстановки володіють

важливою властивістю: вони не порушують частот появи символів, характерних для даної мови.

Це дозволяє криптоаналітику легко отримати відкритий текст за допомогою частотного аналізу.

Для прикладу частотного аналізу на діаграмі приведена частота появи букв в англійському алфавіті (рис 1.3) [2].

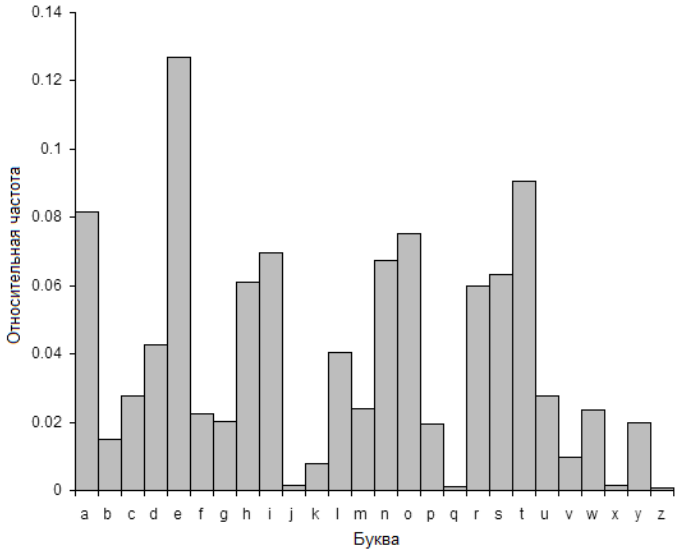


Рисунок 1.4 – Діаграма частотного аналізу появи букв в англійському алфавіті

Шифр Цезаря має замало ключів — на одиницю менше, ніж літер в абетці. Тому, перебрати усі ключі не складає особливої роботи. Дешифрування з одним з ключів дасть нам вірний відкритий текст.

Також зламати шифр Цезаря можна, як і звичайний підстановочний шифр, у зв'язку з тим, що частота появи кожної літери в шифрованому тексті збігається з частотою появи у відкритому тексті. Якщо припустити, що частота появи літер у відкритому тексті приблизно відповідає середньостатистичній відносній частоті появи літер в текстах мови, на якій написано повідомлення, тоді ключ знаходиться зіставленням перших

декількох літер, що трапляються найчастіше у відкритому та зашифрованому текстах. Тобто за допомогою методу частотного криптоаналізу.

1.7 Шифр Гронсфельда

Цей шифр складної заміни є модифікацією шифру Цезаря числовим ключем. Під буквами вихідного повідомлення записують цифри числового ключа. Якщо ключ коротше повідомлення, то його запис циклічно повторюють. Шифр-текст одержують приблизно, як у шифрі Цезаря, але відраховують за алфавітом ту букву, що зміщена на відповідну цифру ключа зліва. Наприклад, якщо ключ – 2718, для вихідного повідомлення STATE POWER одержимо наступний шифр текст (рис. 1.5).

Повідомлення	STATEPOWER
Ключ	2718271827
шифр-текст	UABBGWPE2Y

Рисунок 1. 5 – Шифрування методом складної заміни

Слід зазначити, що шифр Гронсфельда розкривається відносно легко. Оскільки в числовому ключі кожна цифра може мати тільки десять значень, маємо лише десять варіантів прочитання кожної букви шифртекста. З іншого боку, шифр Гронсфельда допускає подальші модифікації, що поліпшують його стійкість. Зокрема подвійне шифрування різними числовими ключами. Шифр Гронсфельда є окремим випадком системи шифрування Віжинера [5].

1.8 Шифр Віжинера

Система Віжинера вперше була опублікована в 1586 році і є однією з найстаріших і найбільш відомих багатоалфавітних систем. Свою назву вона

одержала від імені французького дипломата XVI століття Блеза Віжинера, який розвивав і удосконалював криптографічні системи.

Система Віжинера подібна системі шифрування Цезаря, у якої ключ підстановки міняється від букви до букви. Цей шифр багатоалфавітной заміни можна описати таблицею шифрування, яку називають таблицею (квадратом) Віжинера.

Таблиця Віжинера використовується для шифрування і розшифрування. Таблиця має два входи:

– верхній рядок використовується для зчитування чергової букви відкритого тексту;

– крайній лівий стовпець використовується для зчитування чергової букви ключа.

При шифруванні вихідного повідомлення його виписують у рядок, а під ним записують ключове слово (чи фразу). Якщо ключ виявився коротшим повідомлення, то його циклічно повторюють.

У додатку А наведена таблиця Віжинера для англійського алфавіту.

Розглянемо приклад шифрування за допомогою таблиці Віжинера. Нехай обране ключове слово DOUBT. Необхідно шифрувати повідомлення ALL IN GOOD TIME (рис. 1.6).

Повідомлення	ALLINGOODTIME
Ключ	DOUBTDOUBTDOU
шифр-текст	DZFJGJCIEMLAY

Рисунок 1.6 – Шифрування за допомогою таблиці Віжинера

У процесі шифрування знаходять у верхньому рядку таблиці чергову букву вихідного тексту й у лівому стовпці чергове значення ключа. Чергова буква шифр-текста знаходиться на перетинанні стовпця, обумовленого шифрованою буквою, і рядка, обумовленого буквою ключа.

Випишемо вихідне повідомлення в рядок і запишемо під ним ключове слово з повторенням. У третій рядок будемо виписувати букви шифр-текста, використовуючи таблицю Віжинера.

1.9 Шифр "подвійний квадрат" Уітстона

У 1854 р. англієць Чарльз Уітстон розробив новий метод шифрування біграмами. Шифр Уітстона відкрив новий етап в історії розвитку криптографії. Шифр використовує відразу дві таблиці, розміщені по одній горизонталі, а шифрування відбувається біграмами. Шифр "подвійний квадрат" виявився дуже надійним і зручним. Він застосовувався Німеччиною навіть у роки другої світової війни.

Пояснимо процедуру шифрування цим шифром на прикладі. Нехай маємо дві таблиці з випадково розташованими в них англійськими алфавітами. Перед шифруванням вихідне повідомлення розбивають на біграми. Кожна біграма шифрується окремо. Першу букву біграми знаходять у лівій таблиці, а другу букву – у правій. Потім будують прямокутник так, щоб букви біграми лежали в його протилежних вершинах. Інші дві вершини цього прямокутника дають букви біграми шифр-текста (рис. 1.7).

Якщо обидві букви біграми повідомлення лежать в одному рядку, то і букви шифротекста беруть з цього ж рядка. Першу букву біграми шифротекста беруть з лівої таблиці в стовпці, що відповідає другій букві біграми повідомлення.

Друга ж буква біграми шифротекста береться з правої таблиці в стовпці, що відповідає першій букві біграми повідомлення. Тому біграма повідомлення SW перетворюється в біграму шифротекста YF.

A	M	W	O	_	J			_	A	C	I	,	J
I	R	H	D	G	U			K	R	P	V	L	X
.	S	C	T	F	X			F	Y	G	?	W	E
N	Y	B	L	Z	V			S	.	B	U	Q	T
Q	,	K	?	E	P			N	O	D	H	Z	M

Рисунок 1.7 – Таблиці з символами англійського алфавіту для шифру "подвійний квадрат" Уітстона

Аналогічно шифруються всі біграми повідомлення. Повідомленню BETTER_LATE_THAN_NEVER відповідає шифр-текст CTLEGOG,NJ_N??Q_E_GHG0.

Шифрування методом "подвійного квадрата" дає дуже стійкий до розкриття і простий у застосуванні шифр. Зламування шифртекста "подвійного квадрата" вимагає великих зусиль, при цьому довжина повідомлення повинна бути не менш тридцяти рядків.

1.10 Порівняння з асиметричними криптосистемами

В основному, симетричні алгоритми шифрування вимагають менше обчислень, ніж асиметричні. Несиметричні алгоритми «навантажують» обчислювальні ресурси комп’ютера такими математичними операціями як множення, ділення, зведення в ступінь, що на практиці виконуються значно повільніше, ніж звичайні логічні операції [5].

На практиці, це означає, що якісні асиметричні алгоритми в сотні або в тисячі разів повільніші за якісні симетричні алгоритми.

Недоліком симетричних алгоритмів є необхідність мати секретний ключ з обох боків передачі інформації. Так як ключі є предметом можливого

перехоплення, їх необхідно часто змінювати та передавати по безпечних каналах передачі інформації під час розповсюдження.

Переваги симетричних алгоритмів:

– швидкість (за даними Applied Cryptography – на 3 порядки вище);

– простота реалізації (за рахунок більш простих операцій);

– менша необхідна довжина ключа для порівнянної стійкості;

– вивченість (за рахунок більш тривалого часу використання).

Недоліки:

– складність управління ключами у великій мережі. Це означає квадратичне зростання числа пар ключів, які треба генерувати, передавати, зберігати і знищувати в мережі. Для мережі в 10 абонентів потрібно 45 ключів, для 100 вже 4950, для 1000 – 499500 і т. д [3].

– складність обміну ключами. Для застосування необхідно вирішити проблему надійної передачі ключів кожному абоненту, тому що потрібен секретний канал для передачі кожного ключа обом сторонам.

Для компенсації недоліків симетричного шифрування в даний час широко застосовується комбінована (гібридна) криптографічний схема, де за допомогою асиметричного шифрування передається сеансовий ключ, що використовується сторонами для обміну даними за допомогою симетричного шифрування. Важливою властивістю симетричних шифрів є неможливість їх використання для підтвердження авторства (електронний цифровий підпис), так як ключ відомий кожній стороні.

1.11 Постановка задачі розроблення програмного комплексу

На основі проведено огляду та розглянутих алгоритмів розробити програмний комплекс шифрування інформації на основі алгоритмів симетричного шифрування. В програмі шифрування передбачити:

– віконний інтерфейс;

– вибір методу шифрування, що розглянуті у розділі 1;

– реалізацію шифрування приватним ключем;

– реалізацію блочного шифрування;

– можливість роботи з 64-розрядними блоками даних і ключами трьох розмірів (128, 192 і 256 розрядів).

– роботу програми у консольном додатку.

Програма має забезпечувати:

– інтуїтивно зрозумілий інтерфейс;

– функціональну повноту та гнучкість, що передбачає додавання інших методів криптографічного перетворення:

– уніфікованість у використанні.

Додатково необхідно:

– використовувати операції, які легко реалізуються як апаратно, так і програмно;

– орієнтуватися на 32/64 -розрядні процесори;

– не ускладнювати без необхідності структуру шифру.

1.12 Висновки до розділу 1

У першому розділі дипломного проекту розглянуті та проаналізовані сучасні системи симетричного кодування інформації, розглянуті та визначені поняття блокового шифру, ключа шифрування тощо.

На основі проведено огляду визначені шляхи реалізації проекту та алгоритми реалізації.

1.13 Перелік джерел посилань до розділу 1

1. Анин Б. Защита компьютерной информации. – СПб.: БХВ-Петербург, 2010. – 234 с.

2. Введение в криптографию. Под общей ред. В.В. Яценко. – СПб.: Питер, 2015. – 290 с.

3. Задірака В., Олексюк О. Методи захисту інформації. Тернопіль, «Збруч», 2018. – 190 с.

4. Зегжда Д. П., Ивашко А.М. Основы безопасности информационных систем. – М.: «Горячая линия – Телеком», 2014. – 345 с.

5. Масленников М. Практическая криптография. – Санкт-Петербург, „БХВ-Петербург”, 2010. – 267 с , ил.

2 СТРУКТУРА ПРОГРАМНОГО КОМПЛЕКСУ ШИФРУВАННЯ ІНФОРМАЦІЇ

2.1 Організаційна побудова програмних засобів захисту інформації

Для організованої побудови програмних засобів захисту інформації найхарактерніша тенденція розроблення комплексних програм, що виконують багато захисних функцій, причому найчастіше до цих функцій входить розпізнавання користувачів, розмежування доступу до масивів даних, заборона доступу до деяких областей оперативної пам’яті тощо.

Переваги таких програм очевидні: кожна з них забезпечує розв’язання низки важливих задач захисту. З описаних недоліків та переваг випливають такі вимоги до формування програмних засобів захисту інформації [1].

– функціональна повнота;
– гнучкість;
– уніфікованість використання.

Найповніше вимоги гнучкості та уніфікованості задовольняють такі сукупності принципів:

– наскрізна модульна будова;
– повна структуризація;
– представлення на машиннонезалежній мові.

Принцип наскрізної модульної побудови полягає в тому, що кожна з програм будь-якого рівня та об’єму має подаватися у вигляді системи можливих модулів, причому кожний модуль будь-якого рівня має бути повністю автономним і мати стандартні вхід та вихід, що забезпечує компонування з будь-якими іншими модулями. Очевидно, що ці умови можуть бути виконані, якщо програмне забезпечення розроблятиметься за принципом “згори донизу”, тобто відповідно до принципу повної структуризації.

Представлення на машиннонезалежній мові передбачає таке подання

програмних модулів, щоб їх із мінімальними зусиллями можна було ввести до складу програмного забезпечення будь-якої системи.

Сучасні алгоритмічні мови високого рівня повністю відповідають цим вимогам (такі мови часто є трансплатформними та підтримуються середовищами MS CryptoAPI 2.0 та .NET Framework) [2].

Використання CryptoAPI – інтерфейсу програмування додатків, який забезпечує розробників стандартним набором функцій для роботи з криптопровайдером та входить до складу операційних систем Microsoft. Більшість функцій CryptoAPI підтримуються, починаючи з ще Windows 2000.

Криптопровайдер – це зазвичай бібліотека, що реалізує певний набір криптографічних алгоритмів і забезпечує роботу із ними. Існує близько семи стандартних провайдерів, попередньо встановлених у системі. Кожен криптопровайдер належить до певного типу. Це дає змогу, перебравши всі встановлені в сисмемі провайдери, вибрати ті, які підтримують потрібні алгоритми. Криптопровайдер підтримує захищені області, що називають контейнерами ключів. Контейнери дають можливість додаткам зберігати і використовувати надалі згенеровані один раз ключі, забезпечуючи захист самих ключів від компрометації та модифікування. Більша частина криптографічних класів (не абстрактних) .NET базується на криптопровайдерах CryptoAPI.

Однак є і винятки (наприклад SHA256Managed). У принципі, ієрархія класів.NET дозволяє абстрагуватися від конкретної реалізації алгоритму, що може забезпечити простий перехід на не прив’язані до CryptoAPI класи в майбутньому.

.NETFramework містить набір криптографічних сервісів, що розширюють аналогічні сервіси Windows через CryptoAPI.

Простір імен System.Security.Cryptography відкриває програмний доступ до різноманітних криптографічних сервісів, за допомогою яких додатки можуть шифрувати та дешифрувати дані, забезпечувати їх цілісність, а також обробляти цифрові підписи та сертифікати.

На найвищому рівні простір імен Cryptography можна розділити на чотири основні частини, як показано в табл. 2.1 [3].

Таблиця 2.1 – Основні елементи простору імен Cryptography

Елемент	Опис
Алгоритми шифрування	Набір класів, що застосовується для симметричного та асиметричного шифрування, а також хешування
Допоміжні класи	Класи, що забезпечують генерацію випадкових чисел, виконання перетворень, взаємодію із бібліотеками CryptoAPI і саме шифрування на основі потокової моделі
Сертифікати X.509	Класи, визначені в просторі імен System.Security.Cryptography.X509 Certificates, які представляють цифрові сертифікати
Цифрові підписи XML	Класи, що визначають в просторі імен System.Cryptography.Xml і представляють цифрові підписи в XML-документах

Основне призначення цього простору – надавати класи з алгоритмами криптографічних операцій.

2.2 Програмна реалізація блокового симетричного шифру на основі логічної операції за модулем 2

Важливою вимогою, що висувається до сучасного блокового симетричного шифру (БСШ), є наявність простої та одночасно ефективної програмної реалізації за допомогою розповсюджених мов програмування. Особлива увага приділяється продуктивності програмної реалізації БСШ на

сучасних процесорах, що є найрозповсюдженішими у світі.

В рамках конкурсу використання алгоритму AES [1, 2] проводилась оцінка ефективності програмної реалізації та продуктивності на різних платформах кожного БСШ, що був поданий до розгляду, і перевага серед БСШ з рівним ступенем криптографічної стійкості віддавалась тим, продуктивність та простота реалізації яких була вищою.

В роботах [3, 4] запропоновано БСШ на основі арифметичних операцій за модулем, який відповідає сучасним тенденціям розвитку БСШ і побудований з використанням операцій, що можуть ефективно виконуватись сучасними 32-х та 64-х розрядними процесорами.

Метою даної роботи є підвищення ефективності програмної реалізації БСШ на сучасних процесорах шляхом використання цілочисельної арифметики за модулем 2^n. Для досягнення поставленої мети розв’язуються такі задачі [1]:

1. Пошук оптимальної програмної реалізації основних операцій, що використовуються у БСШ, запропонованому в роботах [3, 4].

2. Аналіз продуктивності програмної реалізації даного БСШ мовою C++ на сучасних процесорах.

БСШ, запропонований в роботах [3, 4], розглядає блоки даних, підключі секретного ключа та блоки шифротексту як n-розрядні цілі числа, де n — розмір блока, що має бути кратним розміру машинного слова процесора для створення ефективної програмної реалізації. Основним елементом БСШ, який забезпечує перемішування та розсіювання розрядів відкритого тексту, є логічна операція за модулем 2^n.

Криптографічна стійкість БСШ забезпечується поєднанням операцій з різних груп: множення за модулем 2^n, додавання за модулем 2 та перестановки блоків розрядів n-розрядного цілого числа. В загальному випадку пропонується здійснювати зашифрування, виконуючи такі обчислення [3]:

$$C = \left(\left(\left((M \oplus K_1) \times K_2 \right)_{\mathrm{mod}\, 2^n} \right)^{\leftrightarrow k} \times K_3 \right)_{\mathrm{mod}\, 2^n}, \tag{2.1}$$

де M, C — блоки відкритого та зашифрованого тексту відповідно;

$K1$, $K2$, $K3$ — підключі, які отримуються з секретного ключа K шляхом використанням процедури розгортання ключа, що описана в роботі [4];

$\leftrightarrow k$ — операція перестановки k-розрядних блоків n-розрядного числа.

В роботі [4] проведено статистичний аналіз БСШ з процедурою зашифрування (1) та показано, що статистично безпечним даний БСШ є у випадку використання перестановки блоків розрядів при $k = 1$, що відповідає дзеркальній перестановці розрядів n-розрядного цілого числа.

Окрім перестановки при $k = 1$ в даній роботі також розглядаються варіанти побудови БСШ з використанням перестановок при $k = 32$ та $k = 8$, оскільки програмна реалізація таких перестановок є простішою, а для перестановки при $k = 8$ виявлені в роботі [4] статистичні аномалії є незначними, що робить можливим використання БСШ з такими перестановками для вирішення певних задач, в яких важливим чинником є швидкодія БСШ при задовільному рівні криптографічної стійкості.

Для розшифрування необхідно виконати зворотні обчислення [3],

$$M = \left(\left(\left(C \times K_3^{-1} \right)_{\mathrm{mod}\, 2^n} \right)^{\leftrightarrow k} \times K_2^{-1} \right)_{\mathrm{mod}\, 2^n} \oplus K_1, \tag{2.2}$$

де — числа, що є оберненими відповідно до $K2$ та $K3$ за модулем 2^n.

Необхідність обчислення чисел, що є оберненими накладає додаткові вимоги на підключі $K2$ та $K3$, які мають бути взаємно простими з числом 2^n,

тобто непарними. Значення підключів для процедури розшифрування, що описується виразом (2), обчислюються на етапі розгортання ключа для розшифрування.

2.3 Особливості програмної реалізації основних операцій блокового шифруваня

Програмна реалізація БСШ на основі арифметичних операцій за модулем полягає у реалізації таких операцій [1]:

– додавання n-розрядних цілих чисел за модулем 2;

– перестановки блоків по k розрядів n-розрядних цілого числа при k = 32, k = 8 або k = 1;

– множення за модулем 2^n n-розрядних цілих чисел;

– знаходження числа, оберненого за модулем 2^n.

Реалізація операції додавання n-розрядних цілих чисел за модулем 2 є простою і полягає у використанні 4-х операцій додавання 32-х розрядних цілих чисел за модулем 2. Операції перестановки при k = 32 та k = 8 також реалізуються просто.

В першому випадку необхідно виконати дві операції обміну 32-х розрядними словами, які реалізуються або шляхом копіювання ділянок пам’яті, або звичайним присвоєнням. Для виконання такої операції необхідне використання додаткової пам’яті розміром одне машинне слово для зберігання проміжного результату.

Перестановка по 8 розрядів найшвидше виконується таким чином: спочатку виконується перестановка при k = 32, а далі для кожного 32-х розрядного слова — операція дзеркальної перестанов-ки байтів, для швидкого виконання якої призначена інструкція процесора BSWAP.

Коли використання машинної команди неможливе (наприклад, для реалізації перестановки при k = 8 на мові Java), така перестановка може бути виконана з використанням операцій зсуву, циклічного зсуву та логічних

операцій. Дзеркальна перестановка розрядів *n*-розрядного цілого числа (при *k* = 1) може бути виконана двома способами:

– виконується перестановка блоків по 32, 16 або 8 розрядів, далі для кожного блока вико-нується дзеркальна перестановка розрядів за допомогою логічних операцій та операцій зсуву;

– виконується перестановка блоків по 8 розрядів, далі для кожного блока виконується таблична підстановка попередньо обчисленого значення.

У другому випадку необхідно зберігати таблицю з 256-ти елементів попередньо обчислених дзеркальних перестановок розрядів 8-розрядних цілих чисел.

Такий підхід є ефективнішим, якщо немає обмежень у використанні додаткової пам’яті. Для реалізації операції множення *n*-розрядних цілих чисел за модулем використовується алгоритм множення «у стовпчик». При цьому *n*-розрядне число представляється як масив з цілих *w*-розрядних чисел, над якими виконуються операції множення та додавання, де *w* — довжина машинного слова процесора.

Нехай *A*, *B* — *n*-розрядні цілі числа, A_i — *i*-й *w*-розрядний елемент числа *A*, B_j — *j*-й *w*-розрядний елемент числа *B*, $q=n/w$.

Тоді операцію множення чисел *A* і *B* за модулем 2^n можна представити у такому вигляді [2]:

$$
\begin{array}{cccccc}
 & \times A_{q-1} & \cdots & A_1 & A_0 \\
 & B_{q-1} & \cdots & B_1 & B_0 \\
\hline
 & A_{q-1}\cdot B_0 & \cdots & A_1\cdot B_0 & A_0\cdot B_0 \\
+ & A_{q-2}\cdot B_1 & \cdots & A_0\cdot B_1 & \\
+ & \cdots & \cdots & & \\
+ & A_0\cdot B_{q-1} & & & \\
\hline
 & C_{q-1} & \cdots & C_1 & C_0
\end{array}
\qquad (2.3)
$$

Всі дії при виконанні множення «у стовпчик» виконуються з перенесенням залишку від молодших до старших розрядів (блоків), тобто значення C1 не може бути обчислене перед обчисленням значення C0 і т. д.

Таким чином виключається можливість паралельного виконання даної операції декількома процесорами або спеціалізованими обчислювальними пристроями, що може негативно вплинути на продуктивність програмної реалізації БСШ.

Прискорити виконання операції множення за модулем 2^n можна за рахунок використання додаткової пам’яті для попереднього обчислення і зберігання добутків.

У такому випадку на етапі множення можна використати переваги сучасних процесорів, здатних виконувати декілька арифметичних операцій за один такт. Також такий підхід значно спрощує програмний код і зменшує імовірність появи помилки при реалізації БСШ.

За наявності обмежень на використання додаткової пам’яті програмна реалізація операції множення за модулем 2^n потребує додатково лише $2w$ розрядів для виконання операцій множення та додавання зі збереженням перенесення, але є більш складною для реалізації і повільнішою при виконанні, оскільки містить більшу кількість операцій. Для знаходження оберненого числа за модулем 2^n можна використати модифіковані версії розширеного алгоритму Евкліда [5], операцію ділення за модулем 2^n, що докладно описана в роботі [3], та алгоритм, запропонований в роботі [6]. Оскільки перші два алгоритми працюють над визначенням кожного розряду невідомого числа окремо і потребують значних обчислень, їх програмна реалізація є неефективною.

В роботі [6] для обчислення оберненого числа за модулем пропонується використовувати більш швидкий алгоритм, який використовує властивості арифметичних операцій за модулем 2^n.

Нехай φ(a, m) — число, що є оберненим до *a* за модулем *m*; *r* — залишок від ділення цілого числа *a* на число *m*. Тоді справедливе твердження [6]:

$$\varphi(a, m^2) = (2 - a\varphi(r\ m))\ \varphi(r, m) \qquad (2.4)$$

Таким чином, знаходження оберненого числа за модулем m^2 згідно з цим алгоритмом зводиться до обчислення оберненого числа за модулем *m* і виконання додаткових арифметичних операцій з *m*-розрядними числами. Вираз (2.4) використовується допоки не стане можливим просте обчислення значення за допомогою табличної підстановки.

Оскільки *K*2 та *K*3 — непарні, достатньо зберігати таблицю з 128 елементів, в якій будуть міститись попередньо обчислені обернені значення за модулем 2^8 для всіх непарних цілих чисел з множини *z*.

Наведений алгоритм містить арифметичні операції віднімання та множення чисел з різною розрядністю.

Якщо $n = 128$, то для реалізації множення 64-розрядних чисел із збереженням старшої частини може бути використаний алгоритм множення за модулем 2^{128}. Арифметичні дії над числами меншої розрядності реалізуються на 32-х розрядних процесорах без використання додаткових алгоритмів.

2.4 Аналіз продуктивності програмної реалізації блокового симетричного шифру

Методика оцінки продуктивності програмної реалізації БСШ докладно описана в роботах [1, 2]. Вона полягає у визначенні середньої кількості тактів процесора (або часу), необхідних для виконання таких операцій:

– зашифрування блока даних;

– розгортання ключа для зашифрування;

– розшифрування блока шифротексту;

– розгортання ключа для розшифрування.

Згідно з цією методикою, для визначення кількості тактів використовується інструкція RDTSC, що призначена для отримання значення внутрішнього лічильника тактів процесора. Для того, щоб зменшити вплив процесів, що виконуються паралельно, на кінцевий результат, перед виконанням обчислень відключаються внутрішні переривання, а також використовується інструкція процесора CPUID для запобігання отримання значення лічильника тактів до повного завершення виконання коду [1].

В проведених дослідженнях розглядався БСШ на основі арифметичних операцій за модулем 2^n при $n = 128$ з процедурою зашифрування, розшифрування та процедурами розгортання ключа для зашифрування та розшифрування, що описані в роботі [4].

Оскільки процедури зашифрування та розшифрування містять однакову кількість операцій і відрізняються тільки порядком їх виконання, різниця в кількості тактів, необхідних для зашифрування та розшифрування, або відсутня, або не перевищує кількох тактів процесора (що було також підтверджено при проведенні експериментів), тому результати для операцій зашифрування та розшифрування подаються одним значенням

Як і очікувалось, БСШ з перестановками k = 32 та k = 8 розрядів є набагато швидшим, ніж з використанням дзеркальної перестановки розрядів. Причому різниця між варіантами БСШ з перестановками k = 32 та k = 8 дуже незначна.

Щоб показати результати тестування програмної реалізації БСШ мовою C++ з використанням компілятора Microsoft compiler, з метою визначення кількості тактів, необхідної для виконання операцій на різних процесорах, скористаємось таблицею з роботи (табл. 2.1) [5].

Таблиця 2.1 – Кількість тактів, необхідних для виконання операцій

Операція	Кількість тактів				
	Intel Pentium i5	Intel Core i3	Intel Celeron	AMD Athlon 64	AMD Fenom 64
Зашифрування/розшифрування к=32	211	230	277	190	172
Зашифрування/розшифрування к=8	215	250	279	192	173
Зашифрування/розшифрування к=1	282	300	481	323	316
Розгортання ключа для зашифрування	850	1010	1433	989	952
Розгортання ключа для розшифрування	1504	1686	2287	1562	1448

Це пояснюється тим, що операція перестановки байтів32-х розрядного числа виконується дуже швидко з використанням інструкції процесора BSWAP.

Якщо кількість тактів, необхідних для зашифрування та розшифрування блока даних приблизно однакова, то операція розгортання ключа для розшифрування є повільнішою ніж операція розгортання ключа для зашифрування, оскільки додатково обчислюються значення $K2^{-1}$ та $K3^{-1}$.

Найкращі результати отримані для 64-х бітного процесора Athlon Fenom 64, що є швидшим в операціях зашифрування та розшифрування при k = 32 та k = 8.

Висока ефективність зазначених процесорів в 32-х розрядному режимі роботи до сягається за рахунок низької латентності виконання арифметичних та логічних операцій, короткого конвеєра та наявності ефективного алгоритму попередження переходів та декодування інструкцій.

Низькі результати, отримані для процесорів Intel Celeron, пов’язані насамперед з високою латентністю операції множення (14 тактів для

процесорів на ядрі Northwood і 10—11 — для процесорів на основі ядра Prescott) та найдовшим серед сучасних процесорів конвеєром (від 20 для ядра Northwood до 31 для ядра Prescott), що негативно впливає на продуктивність не оптимізованогоспеціальним чином коду.

В роботі [1] аналізується ефективність програмної реалізації мовою Ansi C БСШ, що є фіналістами конкурсу AES, для різних тестових платформ з використанням компілятора C++ 5.0 та Microsoft Visual Studio.

Серед наведених результатів є результати для процесора архітектури Intel, що робить можливим порівняння наведених в роботі [1] результатів із отриманими в даній роботі для аналогічного процесора Pentium II.

Для порівняння результатів продуктивності БСШ на основі арифметичних операцій за модулем 2^n та відомих БСШ, був обраний статистично безпечний варіант БСШ на основі арифметичних операцій за модулем з процедурою перестановки розрядів при $k = 1$, який є одночасно і найповільнішим.

Зауважимо, що таке порівняння є приблизним, оскільки продуктивність БСШ може відрізнятись навіть для процесорів одного сімейства з деякими відмінностями в побудові.

Розроблені, зазвичай, програми спроектовані як багатофункціональна система візуалізації, аналізу і, власне, реалізації шифрування даних з використанням криптографічних алгоритмів.

Основними компонентами системи є:

Підсистема шифрування за стандартом AES, що містить модулі для реалізації шифрування і дешифрування з підтримкою режиму електронної кодової книги (Electronic codebook, ECB).

Підсистема візуалізації, яка розкриває суть алгоритму і принципів перетворень, реалізованих у ньому, за допомогою демонстрації процесу шифрування та відповідного графічного матеріалу.

Система має володіє всіма необхідними засобами для виконання операцій блокового перетворення 128-бітних ключів та блоків даних

незалежно один від одного, і, відповідно, раундових перетворень.

В табл. 2.2 наведена кількість тактів, необхідна для виконання операцій на процесорах Intel для різних БСШ [6].

Таблиця 2.2 – Кількість тактів, необхідних для виконання БСШ

Операція	Кількість тактів					
	БСШ на основі операції за модулем 2^n	MARS	RC6	Rijndael	Serpent	Twofish
Зашифрування	330	669	330	826	1281	800
Розшифрування	330	583	320	796	1122	628
Розгортання ключа для зашифрування	1038	4937	2283	1292	6947	9226
Розгортання ключа для розшифрування	1686	4938	2284	1722	6935	9249

Обмеження на розмір даних, що шифруються, залежать тільки від кількості адресованої віртуальної пам'яті, доступної системі.

Основними інформаційними блоками в роботі програми є стани відкритого та шифрованого тексту. На основі введеного користувачем ключа система виконує планування підключів і генерує ключові послідовності для шифрування та дешифрування. Реалізацію системи виконується засобами об'єктно-орієнтованого програмування, на основі програми у вигляді сукупності об’єктів, кожний з яких є реалізацією визначеного типу, що використовує механізм пересилання повідомлень та класи, організовані в ієрархію успадкування.

Інструментом для створення програми вибрано мову програмування C++. Програми на C++ транслюються у байтовий код, що виконує віртуальна машина – програма, яка обробляє байтовий код і передає інструкції

обладнанню як інтерпретатор, однак байтовий код, на відміну від тексту, обробляється значно швидше.

У процесі шифрування використовується режим електронної кодової книги (ECB), також відомий як метод простої заміни. У загальному випадку його рекомендують як найбільш простий і швидкий режим роботи блочного шифру.

Режим ECB – найпростіший режим шифрування. Всі блоки відкритого тексту шифруються незалежно один від одного. Це важливо для шифрованих файлів з довільним доступом, наприклад, файлів баз даних. Якщо база даних зашифрована в режимі ECB, будь-який запис може бути доданий, вилучений, зашифрований або розшифрований незалежно від будь-якого іншого запису (за умови, що кожен запис складається із цілої кількості блоків шифрування). Окрім того, обробка може бути паралельною: якщо використовуються декілька шифрувальних процесорів, вони можуть шифрувати або розшифровувати різні блоки незалежно один від одного.

Серед функцій системи варто також відзначити:

– наявність інструментарію візуалізації алгоритму AES, який розкриває суть алгоритму і принципів перетворень, реалізованих у ньому.

– наявність журналу раундових перетворень, який містить докладний звіт про трансформацію даних кожного шифрованого і дешифрованого блока на всіх етапах раундових перетворетворень.

Отже, підсумовуючи усе вищеописане, можна зробити висновок, що програмний модуль повинен мати такі риси:

– варіативність – проявляється у можливості вибрати алгоритм шифрування та ступінь його стійкості (довжину ключа) безпосередньо в процесі шифрування кожного файла.

– гнучкість – модуль написано об’єктно-орієнтованою мовою високого рівня і реалізовано на платформі .NET Framework, що уможливлює широке застосування у різних системах як фірми Microsoft, так і у Unix-подібних середовищах.

– простота використання – постає у вигляді доступного простого інтерфейсу.

Оскільки використані у модулі алгоритми є міжнародними стандартами шифрування, то це означає, що такий модуль можна використовувати як засіб забезпечення безпеки даних під час зберігання їх та передавання по незахищеній мережі, дотримуючись вимог стандарту PCI DSS.

2.5 Висновки до розділу 2

У другому розділі дипломного проекту розглянута організаційна побудова програмних засобів захисту інформації, здійснено вибір структури програмного комплексу.

Розглянуті особливості програмної реалізації основних операцій блокового шифруваня та проведний аналіз продуктивності програмної реалізації блокового симетричного шифру.

Здійснено аналіз використання блокового симетричного шифру на основі логічної операції додавання за модулем 2

2.6 Перелік джерел посилань до розділу 2.

1. Петров А.А. Компьютерная безопасность. Криптографические методы защиты. – М.: ДМК, 2000.

2. Романец Ю.В., Тимофеев П.А., Шаньгин В.Ф. Защита информации в компьютерных системах и сетях. – М.: Радио и связь, 1999.

3. Столингс В. Криптография и защита сетей: принципы и практика. 2-е изд. : Пер. с англ. – М.: Издательский дом «Вильямс», 2001.

4. Шнаейр Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си. – М.: Триумф, 2003.

1. Ruohonen K. Mathematical Cryptology. – Tampere: Tampere University of Technology. – 2010. – 136 p.

5. Панасенко С. П. Алгоритмы шифрования: спец. справочник / С. П. Панасенко. – СПб.: БХВ-Петербург, 2009. – 576 с.

6. Зензин О. С., Иванов М. А. Стандарт криптографической защиты – AES. Конечные поля / под ред. М. А. Иванова. – М.: КУДИЦ-ОБРАЗ, 2014. – 176 с.

3. РОЗРОБЛЕННЯ ПРОГРАМНОГО КОМПЛЕКСУ

3.1 AES шифрування

Винайдене у Національному Інституті Стандатртів та Технологій (NIST), за Федеральним Стандартом Обробки Інформації (FIPS PUB 197), метод шифрування Advanced Encryption Standard (AES) надає надійний криптографічний алгоритм для захисту електронних даних.

AES-алгоритм (також відомий під назвою Rijndael) – це симетричний блоковий шифр, що може зашифровувати та розшифровувати дані (рис. 3.1) [1].

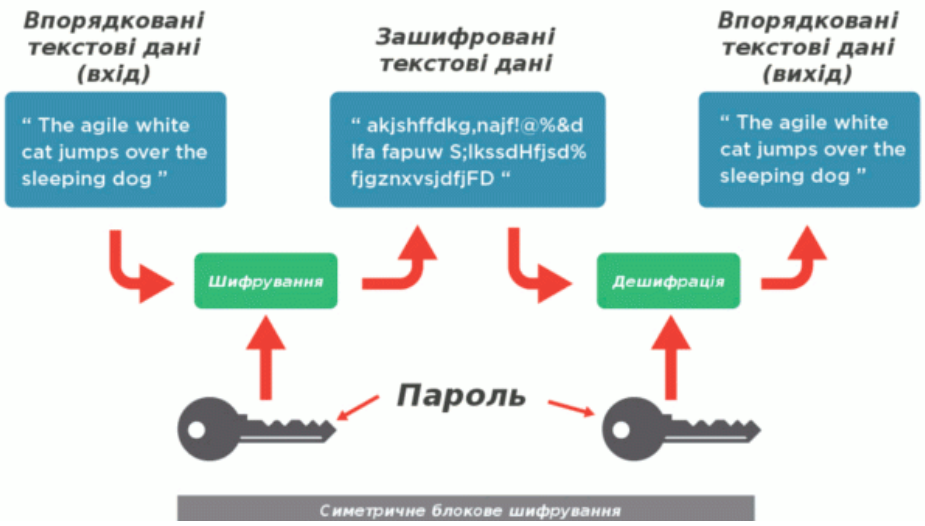


Рисунок 3.1 – Узагальнена структура алгоритму симетричного кодування інформації

Шифрування конвертує дані (впорядкований текст) до нечитаємої форми, що називається зашифрованим текстом, а процес розшифрування конвертує дані у зворотньому порядку. Криптографічний ключ має довжину 128, 192 або 256 біт, що дозволяє зберігати дані розбитими на зашифровані блоки по 128 біт.

Підтримка AES введена фірмою Intel у сімейство процесорів x86, починаючи з Intel Core i7-980X Extreme Edition, а потім на процесорах Sandy Bridge [2].

Переваги AES:

– найбільш досліджений у світі криптографічний алгоритм, висвітлений у відкритих публікаціях;

– забезпечує високу практичну стійкість, включений до набору Suite-B Агентства національної безпеки США, дозволений для захисту інформації з обмеженим доступом уряду США;

– ефективний для реалізації на 32-бітових платформах; наявність низки апаратних акселераторів (включаючи старші моделі процесорів загального призначення).

Є багато областей, в яких AES зараз комерційно використовується. VPN програмне забезпечення найвищого класу містить реалізацію AES, зокрема пропозицій від Checkpoint, Cisco і Symantec. AES тепер часто застосовують у мережевих пристроях. Voice Over IP виробники використовують AES для безпеки телефону. Продавці тепер використовують AES для убезпечення управління технологічними процесами (SCADA) систем. AES навіть додано в загальний файл стиснення програм, таких як WinZip.

3.2 Алгоритм шифрування по ДСТУ ГОСТ 28147:2009

Шифр є перевиданням російського стандарту симетричного шифрування ДСТУ ГОСТ 28147-89:2009 [3].

Особливості шифру:

– блоковий шифр;

– 256-бітовий ключ;

– 32 цикли пертворення;

– 64-бітні блоки;

– основа алгоритму – мережа Фейстеля

Базовим режимом шифрування є режим простої заміни (визначені також складніші режими – гамування, гамування зі зворотним зв'язком і режим імітовставки). Для зашифрування в цьому режимі відкритий текст спочатку розбивається на дві половини (молодші біти — A, старші біти — B).

Дані шифруються за схемою Фейстеля 64-бітними блоками з використанням 256-бітного ключа шифрування. При цьому виконується 32 раунди перетворень.

У структурі алгоритму розрізняють :

– основний крок – послідовність дій, що виконується в кожному базовому циклі (з різними значеннями підключів)

– базові цикли 4 («32-З», «32-Р», («16-З»,) - відрізняються числом повторень основного кроку і порядком використання елементів ключа.

– режими роботи – спеціальні методи забезпечення криптостійкості, що використовують результати шифрування попередніх блоків для шифрування наступних.

Основний крок складається з наступних операцій (рис. 2.2):

1. Один з 32-бітових субблоків даних сумується з 32-бітним значенням ключа раунду Ki по модулю 2^{32} .

2. Результат попередньої операції розбивається на 8 фрагментів по 4 біта, які паралельно «проганяються» через 8 таблиць замін S1 ... S8 розміром 8x16.

3. 4-бітові фрагменти (після замін) об'єднуються назад у 32-бітний субблок.

4. Значення отриманого якого субблоку циклічно зсувається вліво на 11 біт.

Операції основного кроку алгоритму ДСТУ ГОСТ 28147-89:2009 наведені на рисунку 3.2 [3].

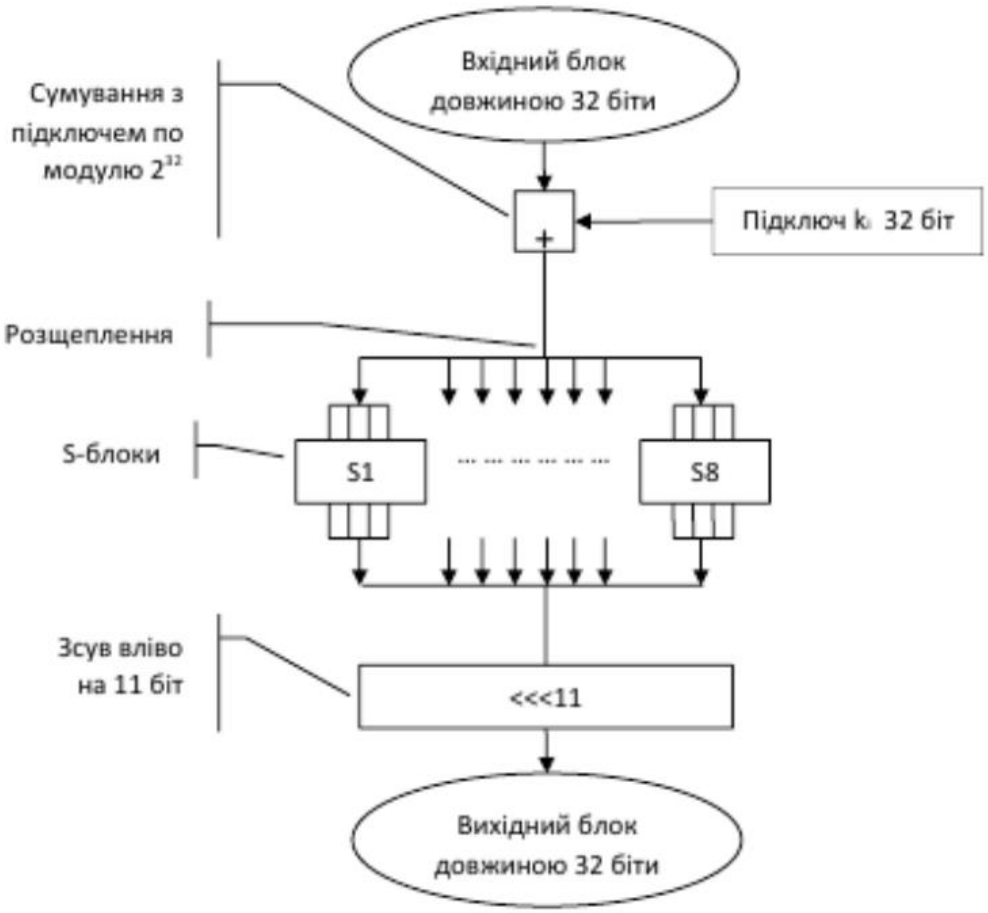


Рисунок 3.2 – Операції основного кроку алгоритму по ДСТУ ГОСТ 28147-89:2009

Структуру алгоритму можна описати багатораундовою схемою (рис. 2.3).

Генерація ключів проста. 256-бітний ключ розбивається на вісім 32-бітових підключів. Оскільки алгоритм має 32 раунди, кожен підключ використовується в чотирьох раундах за схемою, представленою в таблиці 3.3 [3].

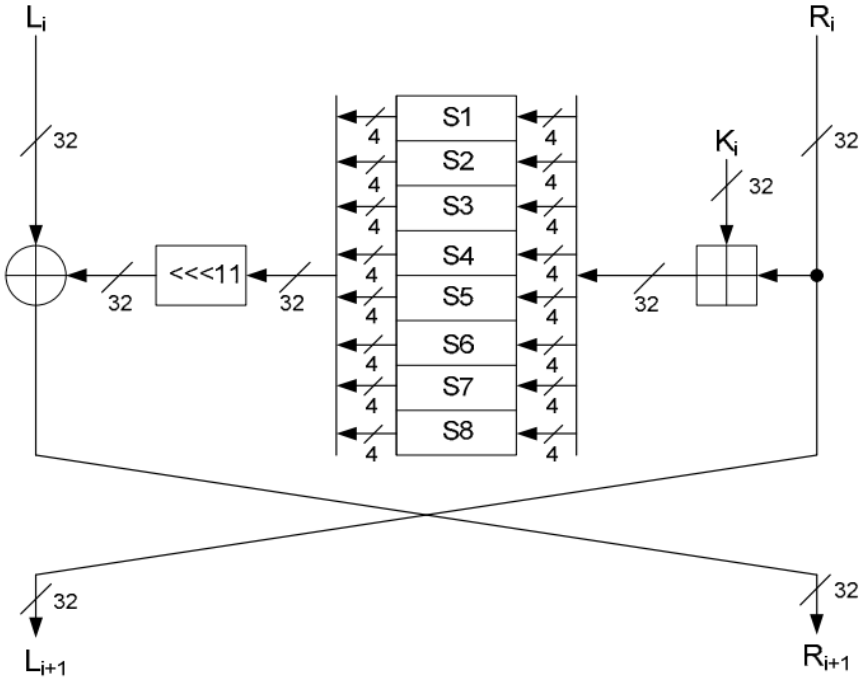


Рисунок 3.3 – Схема одного з 32-х раундів алгоритму по ДСТУ ГОСТ 28147:2009.

В таблиці 3.1 наведена процедура вибору ключів відповідного раунду

Таблиця 3.1 – Використання ключів при шифруванні

Раунд	1	2	3	4	5	6	7	8
Підключ	1	2	3	4	5	6	7	8
Раунд	9	10	11	12	13	14	15	16
Підключ	1	2	3	4	5	6	7	8
Раунд	17	18	19	20	21	22	23	24
Підключ	1	2	3	4	5	6	7	8
Раунд	25	26	27	28	29	30	31	32
Підключ	8	7	6	5	4	3	2	1

Тобто в раундах шифрування послідовно використовуються 32-бітові фрагменти K1 ... K8 вихідного 256-бітного ключа шифрування в наступному порядку: K1, K2, K3, K4, K5, K6, K7, K8, - за винятком останніх 8 раундів - в раундах з 25-го по 31-й фрагменти використовуються в зворотному порядку. Така послідовність використання підключів при за шифруванні отримала назву базовий цикл «32З».

Вважається, що стійкість алгоритму визначається структурою S-блоків. Входом і виходом S-блоків є 4-бітні числа, тому кожен S-блок може бути представлений у вигляді рядка чисел від 0 до 15, розташованих в деякому порядку. Тоді порядковий номер числа буде вхідним значенням S- блоків, а саме число - вихідним значенням S- блоків.

Довгий час структура S- блоків в відкритій пресі не публікувалася.

Усі вісім S-блоків можуть бути різними. Фактично, вони можуть бути додатковим ключовим матеріалом, але частіше є параметром схеми, загальним для певної групи користувачів. Для цілей тестування рекомендуються наведені такі S-блоки, які показані в таблиці 3.2 [3].

Таблиця 3.2 – Приклад таблиці заміни

Номер S-блоку	Значення															
1	4	10	9	2	13	8	0	14	6	11	1	12	7	15	5	3
2	14	11	4	12	6	13	15	10	2	3	8	1	0	7	5	9
3	5	8	1	13	10	3	4	2	14	15	12	7	6	0	9	11
4	7	13	10	1	0	8	9	15	14	4	6	12	11	2	5	3
5	6	12	7	1	5	15	13	8	4	10	9	14	0	3	11	2
6	4	11	10	0	7	2	1	13	3	6	8	5	9	12	15	14
7	13	11	4	1	3	15	5	9	0	10	14	7	6	8	2	12
8	1	15	13	0	5	7	10	4	9	2	3	14	6	11	8	12

Така послідовність використання підключів при розшифруванні отримала назву базовий цикл «32-Розшифрування».

Стандарт також передбачає і описує різні режими застосування алгоритму:

– режим простої заміни – є аналогом режиму ECB в DES;

– режим гамування – є аналогом режиму OFB в DES;

– режим гамування зі зворотним зв'язком або режим гамування із зціпленням блоків – є аналогом режиму CFB в DES.

– режим обчислення імітовставки – є аналогом режиму CBC в DES.

Режим простої заміни приймає на вхід дані, розмір яких кратний 64-м бітам. Результатом шифрування є вхідний текст, перетворений блоками по 64 біта у випадку зашифрування циклом «32-Зашифрування», а в разі розшифрування - циклом «32-Розшифрування».

Режим гамування приймає на вхід дані будь-якого розміру, а також додатковий 64-бітовий параметр синхропосилку. В ході роботи синхропосилка перетвориться в циклі «32-Зашифрування», результат ділиться на дві частини. Перша частина складається по модулю 2^{32} з постійним значенням 1010101_{16}. Якщо друга частина дорівнює 2^{32-1}, то її значення не змінюється, інакше вона складається по модулю 2^{32-1} з постійним значенням 1010101_{16}.

Отримане об'єднанням обох перетворених частин значення, зване гамою шифру, надходить у цикл «32-Зашифрування», його результат порозрядно складається по модулю 2 з 64-розрядним блоком вхідних даних.

Якщо останній менше 64-х розрядів, то зайві розряди отриманого значення відкидаються. Отримане значення подається на вихід. Якщо ще є вхідні дані, то дія повторюється: складений з 32-розрядних частин блок перетвориться по частинах і так далі.

Режим гамування зі зворотним зв'язком також приймає на вхід дані будь-якого розміру і синхропосилку. Блок вхідних даних поразрядно сумується по модулю 2 з результатом перетворення в циклі «32-Зашифрування» синхропосилки. Отримане значення подається на вихід. Значення синхропосилки замінюється в разі зашифрування вихідним блоком,

а в разі розшифрування - вхідним, тобто зашифрованим. Якщо останній блок вхідних даних менше 64 розрядів, то зайві розряди гами (виходу циклу «32-Зашифрування») відкидаються. Якщо ще є вхідні дані, то дія повторюється: з результату зашифрування заміненого значення утворюється гамма шифру і т.д.

Режим вироблення імітовставки приймає на вхід дані, розмір яких становить не менше двох повних 64-розрядних блоків, а повертає 64-розрядний блок даних, що називається імітовставкою. Тимчасове 64-бітове значення встановлюється в 0, далі, поки є вхідні дані, воно поразрядно складається по модулю 2 з результатом виконання циклу «16-З», на вхід якого подається блок вхідних даних. Для базового циклу «16-Зашифрування» двічі використовуються елементи ключа з першого по останній. Після закінчення вхідних даних тимчасове значення повертається як результат.

У відкритій літературі можна знайти досить мало робіт, присвячених криптоаналізу алгоритму ДСТУ ГОСТ 28147:2009 [2]. Це особливо помітно в порівнянні з величезним числом робіт, присвячених криптоаналізу DES і AES або криптоаналізу деяких з широко використовуваних алгоритмів шифрування, наприклад, IDEA або SAFER. За результатами відкритих робіт можна зробити висновок про досить високу криптостійкість вітчизняного стандарту шифрування.

Основними перевагами алгоритм ДСТУ ГОСТ 28147:2009 є [4]:

– ефективність реалізації і відповідно висока швидкодія.

– безперспективність силової атаки;

– наявність захисту від нав'язування помилкових даних (вироблення імітовставки).

3.3 Шифрування в режимі простої заміни

У криптографії, режими дії (англ. modes of operation) — це математичні операції для уможливлення повторюваного і убезпеченого використання блочного шифру з одним ключем. Блочний шифр дозволяє шифрування тільки одного блоку даних встановленої довжини. Для роботи з блоками різних довжин, дані спочатку потрібно розбити на окремі блоки встановленої даним шифром довжини.

Зазвичай, останній блок треба доповнити до відповідної довжини підхожим доповненням. Режими дій описують процес шифрування кожного з цих блоків і звичайно використовують рандомізацію основану на додатковому значені на вході, відомим як ініціалізаційний вектор, з ціллю зробити це безпечно.

Режими дії первісно були розроблені для шифрування й автентфікації.

Історично, режими шифрування широко вивчались зважаючи на їх властивості поширення помилок при різних сценаріях зміни даних. Подальший розвиток поставив цілісність інформації як цілком окрему від шифрування ціль криптографії. Деякі сучасні режими дії поєднують шифрування і автентифікацію ефективним чином, і відомі як режими автентифікованого шифрування [2].

Хоча режими дії звичайно пов'язують з симетричним шифруванням, в принципі, їх також можна застосувати до примітивів шифрування з відкритим ключем таких як RSA (хоча на практиці шифрування довгих повідомлень з відкритим ключем найчастіше здійснюють із використанням гібридного шифрування).

Реалізація програми шифрування в режимі простої заміни можна розглядати як варіант ДСТУ ГОСТ 24187:2009. Таку реалізацію процесу шифрування називають ще режимом електронної кодової книги.

Розмір синхропосилки та гамми дорівнює 64 байтам, тому шифрування в режимі простої заміни можна розглядати в рамках одного блоку. Якщо

блоків є декілька, то вони шифруються по черзі.

Режим електронної книги – режим простої заміни (англ. electronic codebook, ECB), – найпростіший з режимів шифрування є кодів.

Повідомлення розбивається на блоки і кожен блок шифрується окремо (рис. 3.4 [5].

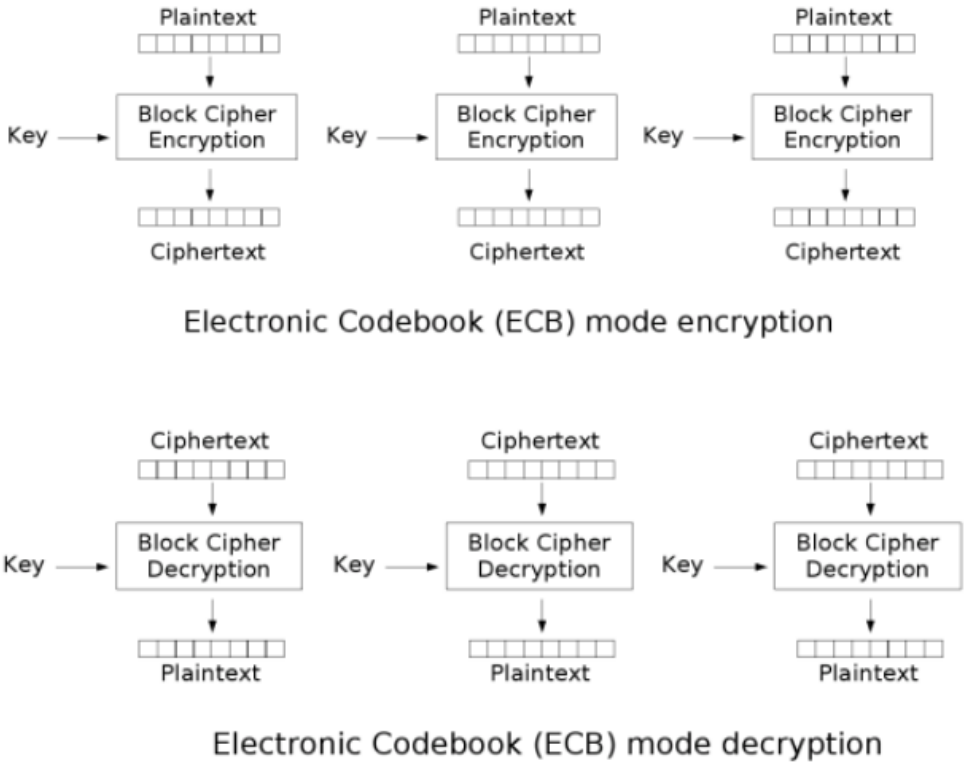


Рисунок 3.4 – Шифрування та дешифрування в режимі простої заміни

Недоліком цього методу є те, що однакові блоки відкритого тексту шифруються в однакові блоки шифротексту; отже шаблон погано приховується. Цей режим не забезпечує серйозну безпеку повідомленням, тому його можна застосовувати тільки для навчальних цілей.

Рекомендується для шифрування застосовувати більш складні алгоритми реалізації , наприклад, ланцюгування шифроблоків, зворотний зв'язок по шифротексту або виходу тощо.

Шифр простої заміни не завжди передбачає заміну літери на якусь іншу літеру. Допускається використовувати заміну літери на число.

Лістинг програми шифрування в режимі простої заміни наведено у додатку А.

Загальний вигляд програми шифрування наведено на рис. 3.5

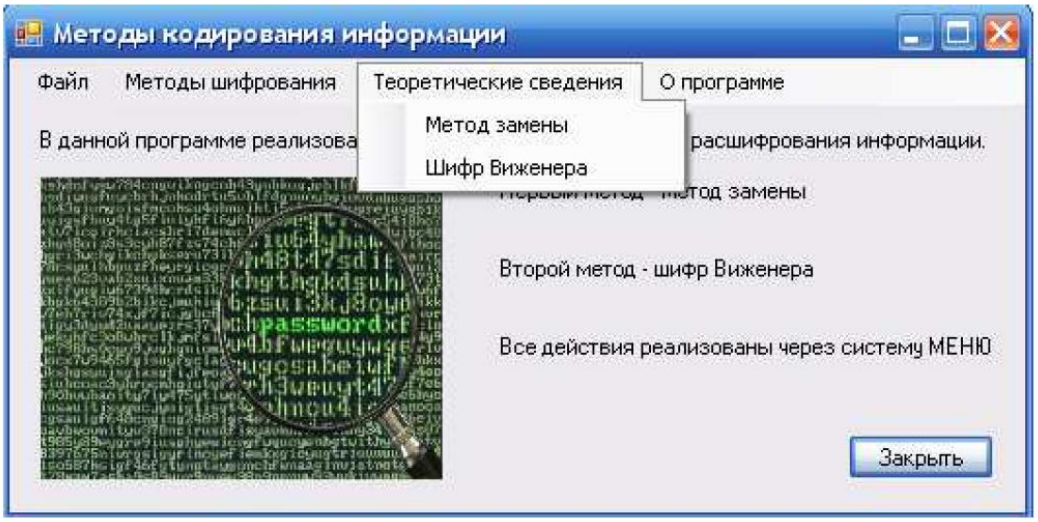


Рисунок 3.5 – Загальний вигдя програми шифрування

У програмі є можливість ознайомитись з короткими теоретичними відомостями щодо методу шифрування (рис. 3.6)

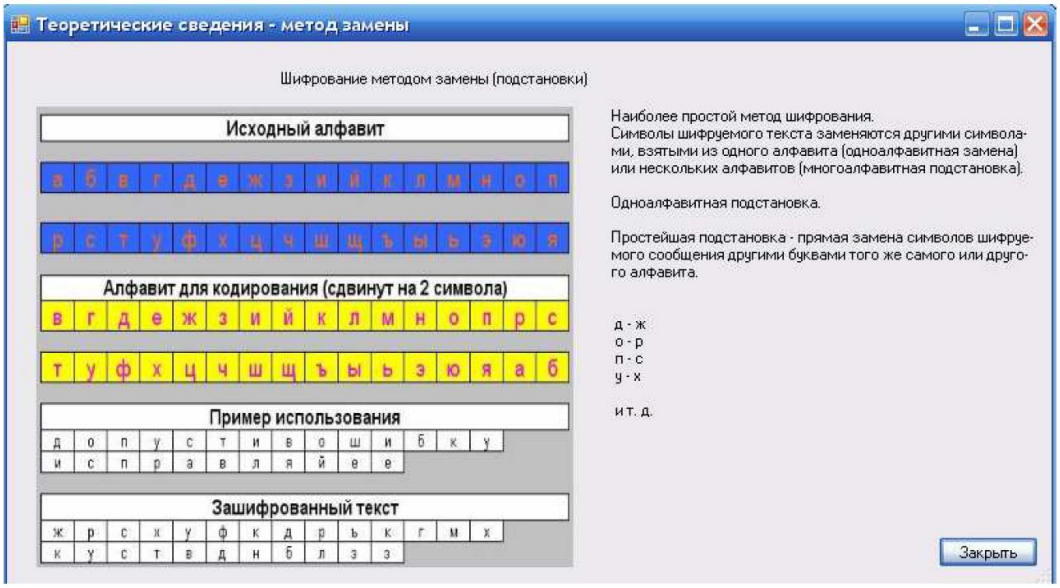


Рисунок 3.6 – Теоретичні відомості шифрування методом простої заміни

Результат виконання програми методом заміни наведено на рисунку 3.7.

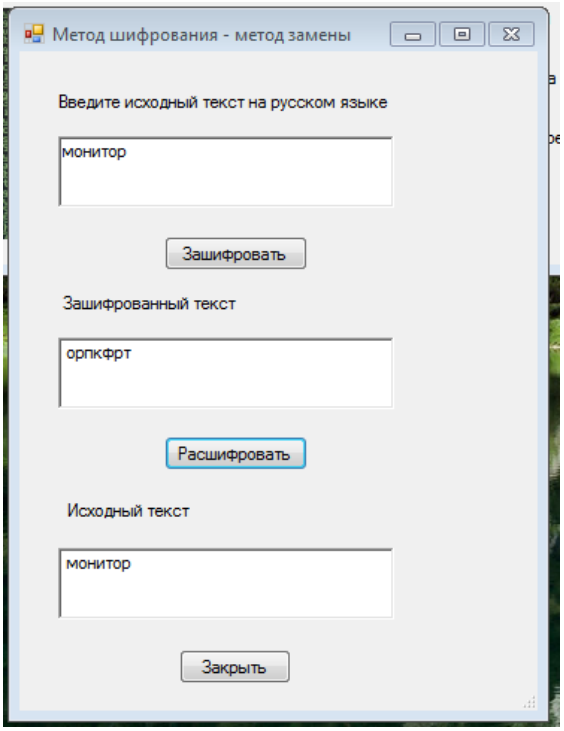


Рисунок 3.7 – Результат виконання шифрування методом заміни

3.4 Шифрування з використання логічної операції додавання по модулю 2

Бітова операція в програмуванні — деякі операції над ланцюжками бітів. У програмуванні, як правило, розглядаються лише деякі види цих операцій: логічні побітові операції та бітові зсуви.

Додавання за модулем два (або двомісна операція виключне АБО) — це бінарна операція, результат дії якої дорівнює 1, якщо число складаємих одиничних бітів непарне, якщо ж їх число парне, то результат дорівнює 0.

Перша назва операції обумовлена тим, що результат цієї операції відрізняється від результату «АБО» лише в одному з 4 випадків входу (1 випадок одночасної істинності аргументів «виключається»). Ще значення цієї логічної зв'язки передається союзом «або».

Друга назва — тим, що дійсно є складанням в кільці вирахувань за модулем два, з чого слідують деякі цікаві властивості.

Наприклад, на відміну від логічних операцій «І» та «АБО», ця операція є оборотною, або інволютивною:

В таблиці 3.3 наведено реалізацію логічної операції додавання по модулю 2

Таблиця 3.3 – Логічна операція додавання по модулю 2

Операнд X_1	Операнд X_2	Результат операції
0	0	0
0	1	1
1	0	1
1	1	0

Завдяки інволютивності ця операція знайшла застосування в криптографії як найпростіша реалізація ідеального шифру (шифру Вернама). «Додавання за модулем два» також може використовуватися для обміну двох змінних значеннями, використовуючи алгоритм обміну XOR.

Також ця операція може називатися «інверсією по масці», тобто у вихідного двійкового числа інвертуються біти, які збігаються з 1 в масці.

Лістинг реалізації програми Шифрування з використання логічної операції додавання по модулю 2 наведено у додатку Б

3.5 Програмна реалізація шифру Цезаря

Юлій Цезар використовував адитивний шифр, щоб зв’язатися зі своїми чиновниками. Із цієї причини адитивні шифри згадуються іноді як шифри Цезаря. Цезар для свого зв’язку використовував цифру 3.

Принцип дії полягає в тому, щоб циклічно зсунити алфавіт, а ключ — це кількість літер, на які робиться зсув.

Припустимо, що початковий текст складається з маленьких букв (від a до z) і зашифрований текст складається із заголовних букв (від A до Z). Щоб забезпечити застосування математичних операцій до початкового і зашифрованого текстів, ми привласнимо кожній букві числове значення (для нижнього і верхнього регістра), як це показано на рисунку 3.8 [2].

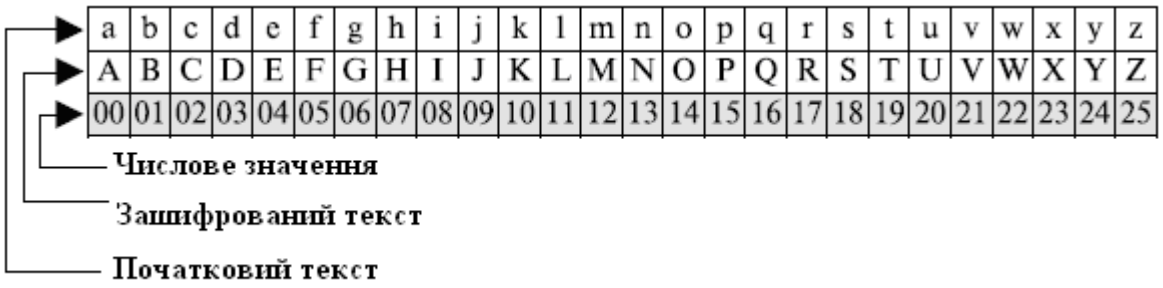


Рисунок 3.8 – Графічна реалізація шифру Цезеря

Якщо зіставити кожному символу алфавіту його порядковий номер (нумеруючи з 0), то шифрування й дешифрування можна виразити формулами [2]:

$$Y = (X + k) \bmod n \quad i \quad X = (Y - k) \bmod n, \tag{3.1}$$

де: X — символ відкритого тексту,

Y — символ шифрованого тексту,

n — потужність алфавіту,

k — ключ.

Детально шифр Цезаря описано в літературі [2].

Лістинг програмної реалізації шифру Цезаря наведено у додатку Д.

3.6 Програмна реалізація шифрування з використання шифру Віжинера

Шифр Віженера — поліалфавітний шифр, який як ключ використовує слово.

Якщо пронумерувати літери алфавіту від 0 до 32 (а → 0, б → 1, в → 2, …), то шифрування Віженера є можливим представити формулою [3]:

$$Ci = (Pi + Kj) \bmod 33, \tag{3.2}$$

де Kj — j-та літера ключового слова,

Pi — i-а літера вихідного слова.

Ключове слово повторюється, поки не отримано гаму, рівну довжині повідомлення.

Дешифрування відбувається за наступною формулою [3]:

$$Ci = (Pi + 33 - Kj) \bmod 33. \tag{3.3}$$

Детально шифр Віжинера описано у пункті 1.8 дипломного проекту.

Лістинг програмної реалізації шифру Віжинера наведено у додатку Д.

Скріншот теоретичних відомостей програми шифру Віжинера наведено на рис. 3.9.

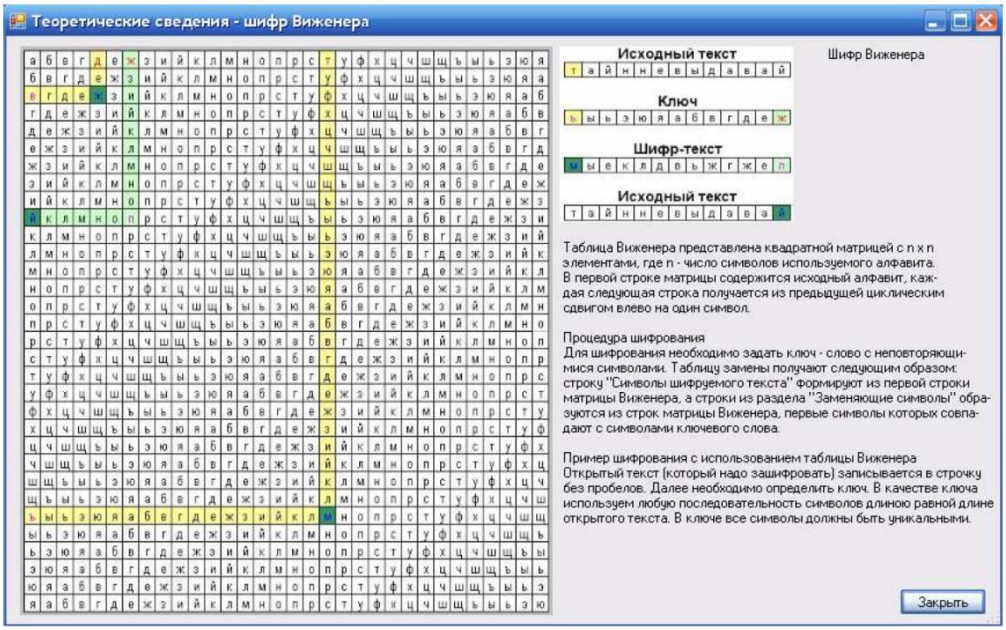


Рисунок 3.9 – Скріншот теоретичних відомостей програми шифру Віжинера

Скріншот виконання програми шифру Віжинера наведено на рис. 3.10.

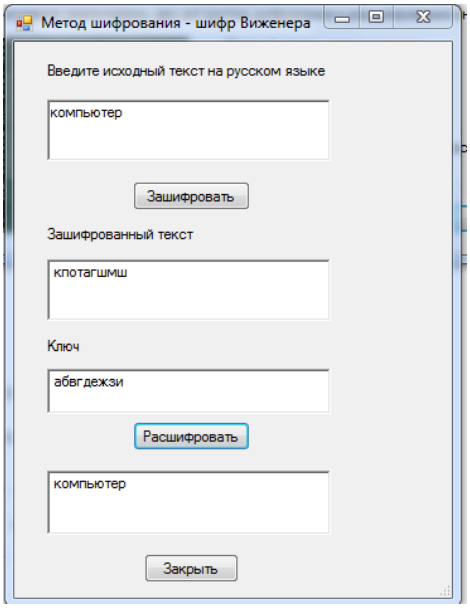


Рисунок 3.10 – Скріншот виконання програми шифру Віжинера

Виконання програми у консольному додатку наведено на рисунку 3.11.

а) шифрування

б) дешифрування

Рисунок 3.11 – Виконання шифрування та дешифрування методом шифру Віжинера у консольному додатку

3.7 Програмна реалізація гамування зі зворотним зв’язком

Даний режим носить назву Cipher Feedback Mode (CFB) – режим з обереженим зв’язком по шифротексту (рис. 3.12) [3] .

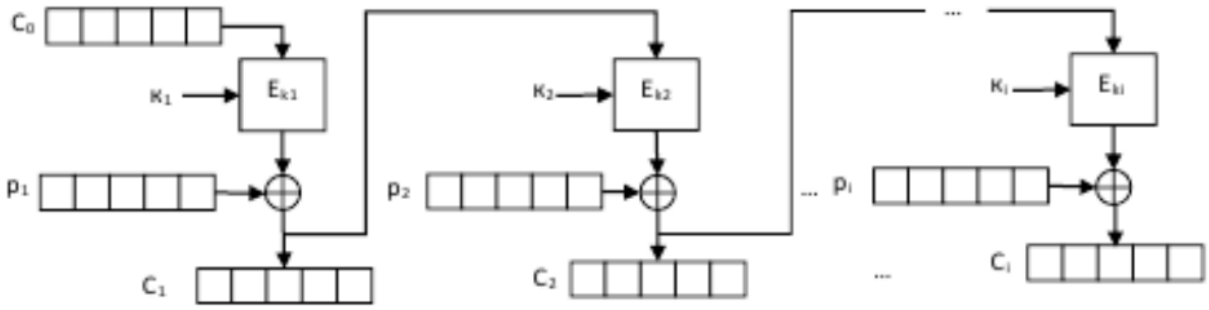


Рисунок 3.12 – Схема зашифрування в режимі CFB

Режим CFB дуже схожий на режим гамування і відрізняється від нього тільки способом вироблення елементів гами – черговий елемент гами виробляється як результат перетворення по циклу 32-зашифрування попереднього блоку зашифрованих даних, а для шифрування першого блоку масиву даних елемент гами виробляється як результат перетворення за тим же циклу сінхропосилки.

Цим досягається зачеплення блоків - кожен блок шифротекста в цьому режимі залежить від відповідного і всіх попередніх блоків відкритого тексту. Тому даний режим іноді називається гамуванням з зачепленням блоків.

Шифрування описується рівняннями [3]:

$$Ci = E_{ki}(C_i\text{-}1)\ XOR\ p_i\,, \qquad (3.4)$$

для $i=2 \div N$, $C0$ – синхропосилка.

На рисунку 3.13 наведена схема розшифрування в режимі зчеплення блоків

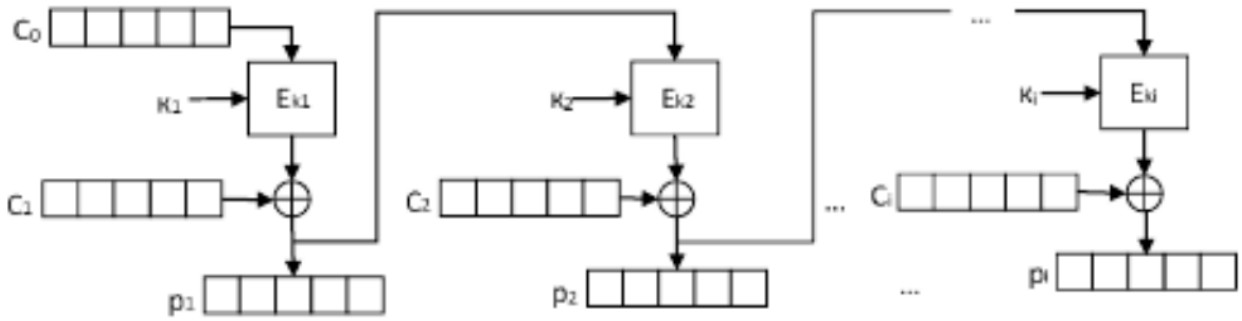


Рисунок 3.13 – Схема розшифрування в режимі CFB

Розшифрування описується рівняннями [3]:

$$pi = E_{ki}(C_i\text{-}1)\ XOR\ C_i\,, \tag{3.5}$$

для $i=2\div N$, $C0$ – синхропосилка.

В цьому режимі алгоритму блокового шифрування черговий блок відкритого тексту сумується по модулю 2 з блоком попереднього шифротексту, але тільки після того, як він буде додатково оброблений за допомогою функції шифрування. Важливо відмітити, що в цьому режимі як для шифрування, так і розшифрування використовується тільки функція шифрування, а обернена до неї функція розшифрування взагалі не потрібна.

На стійкість шифру факт зачеплення блоків не робить ніякого впливу.

Переваги режиму CFB:

– критичність по відношенню до вставки і втрати блоків в процесі передачі;•

– використовується тільки функція шифрування;

– симетрія операцій шифрування/розшифрування.

Недоліком режиму є те, що шифрування (розшифрування) не піддається розпаралеленню.

Слід відмітити, що програмна реалізація цього алгоритму є

найбільшою за кількістю команд та використовує графічний інтерфейс користувача *Win32 GUI*.

Лістинг програмної реалізації гамування зі зворотним зв’язком (режим CFB) наведено у додатку Е.

3.8 Висновки до розділу 3

Детально у розгорнутому вигляді проаналізовано організаційну побудову програмних засобів захисту інформації, реалізацію блокового симетричного шифру на основі логічної операції за модулем 2, особливості програмної реалізації основних операцій блокового шифруваня, апаратні можливості реалізації симетричних алгоритмів. Визначі подальші задачі для розроблення програмного комплексу.

Проведений огляд математичних та логічних операцій, що використовуються в алгоритмах блочного шифрування інформації.

Розроблено програмний комплекс кодування інформації з використанням методів:

– простої заміни;

– логічної операції додавання по модулю 2;

– шифру Цезаря;

– шифру Віженера;

– гамування зі зворотним зв’язком.

Реалізація проекту здійснена в інтегральному середовищі Visual Studio 2019 з використанням мови програмування С++, об’єм програми 709 кБ.

3.9 Перелік джерел посилань до розділу 3

1. Daemen J., Rijmen V. Design of Rijndael. AES – The Advanced Encryption Standard. – Berlin: Springer–Verlag. – 2002. – 238 p.

2. Хорошко В., Чекатков А. Методы и средства защиты информации. – К.: Юниор, 2003. – 504 с.

3. Чмора А. П. Современная прикладная криптография. – 2-е изд. – Гелиос АРВ, 2012. Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua . – 256 с.

4. Бакай О. В., Брич Т. Б., Лах Ю. В. Технології захисту банківських міжнародних платіжних карток // Вісник Національного університету “Львівська політехніка”: Автоматика, вимірювання та керування. – 2012. – № 741. – С. 184–187.

5. Dudykevych V., Bakay O., Lakh Y. Investigation of Payment Cards Systems Information Security Control // Proceedings of the 2013 IEEE 7th International Conference on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS), Volume 2, (September 12–14, 2013, Berlin, Germany) P. 651–654.

4 ОХОРОНА ПРАЦІ

4.1 Загальні питання з охорони праці

Умови праці на робочому місці, безпека технологічних процесів, машин, механізмів, устаткування та інших засобів виробництва, стан засобів колективного та індивідуального захисту, що використовуються працівником, а також санітарно-побутові умови повинні відповідати вимогам нормативних актів про охорону праці. В законі України «Про охорону праці» [1] визначається, що охорона праці - це система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів та засобів, спрямованих на збереження життя, здоров'я і працездатності людини у процесі трудової діяльності.

При роботі з обчислювальною технікою змінюються фізичні і хімічні фактори навколишнього середовища: виникає статична електрика, електромагнітне випромінювання, змінюється температура і вологість, рівень вміст кисню і озону в повітрі. Неправильна організація робочого місця сприяє загальному і локальній напрузі м'язів шиї, тулуба, верхніх кінцівок, викривлення хребта і розвитку остеохондрозу. На всіх підприємствах, в установах, організаціях повинні створюватися безпечні і нешкідливі умови праці. Забезпечення цих умов покладається на власника або уповноважений ним. Умови праці на робочому місці, безпека технологічних процесів, машин, механізмів, устаткування та інших засобів виробництва, стан засобів колективного та індивідуального захисту, що використовуються працівником, а також санітарно-побутові умови повинні відповідати вимогам нормативних актів про охорону праці. Роботодавець повинен впроваджувати сучасні засоби техніки безпеки, які запобігають виробничому травматизмові, і забезпечувати санітарно-гігієнічні умови, що запобігають виникненню професійних захворювань працівників. Він не має права вимагати від працівника виконання роботи, поєднаної з явною небезпекою для життя, а

також в умовах, що не відповідають законодавству про охорону праці. Працівник має право відмовитися від дорученої роботи, якщо створилася виробнича ситуація, небезпечна для його життя чи здоров'я або людей, які його оточують, і навколишнього середовища.

Основним організаційним напрямом у здійсненні управління в сфері охорони праці є усвідомлення пріоритету безпеки праці і підвищення соціальної відповідальності держави і особистої відповідальності працівників.

Державна політика в галузі охорони праці визначається відповідно до Конституції України Верховною Радою України і спрямована на створення належних, безпечних і здорових умов праці, запобігання нещасним випадкам та професійним захворюванням. Відповідно до статті 3 Закону України «Про охорону праці» законодавство про охорону праці складається з Закону, Кодексу законів про працю України, Закону України "Про загальнообов'язкове державне соціальне страхування від нещасного випадку на виробництві та професійного захворювання, які спричинили втрату працездатності" [2] та прийнятих відповідно до них нормативно-правових актів, норм міжнародного договору.

На законодавчому рівні визначено такі пріоритетні напрямки з безпеки праці:

- кожен працівник несе безпосередню відповідальність за порушення зазначених Законом, нормами і правилами вимог;
- напрямки реалізації конституційного права громадян на їх життя і здоров’я в процесі трудової діяльності:
- пріоритет життя і здоров’я працівників по відношенню до результатів виробничої діяльності підприємства;
- повна відповідальність роботодавця за створення належних – безпечних і здорових умов праці;

- соціальний захист працівників, повне відшкодування збитків особам, які потерпіли від нещасних випадків на виробництві та професійних захворювань;
- соціальний захист працівників, повне відшкодування збитків особам, які потерпіли від нещасних випадків на виробництві та професійних захворювань;
- використання світового досвіду організації роботи щодо поліпшення умов і підвищення безпеки праці на основі міжнародної співпраці.

Наявні трудові відносини між працівниками і роботодавцями в Україні за темою дипломного проекту регулюються Кодексом законів про працю (КЗпП) [3] України, відповідно до якого права працюючої людини на охорону праці охороняються всебічно та норми охорони праці неухильно інтегровані до правил внутрішнього розпорядку організації/підприємства.

4.2 Аналіз стану умов праці

Робота над розробленням програмного комплексу криптографічного перетворення інформації на основі симетричного шифрування буде проходити у офісному приміщенні з використанням персонального комп’ютера.

Отже, для виконання даної роботи достатньо одне робоче місце зі стаціонарним комп’ютером або ноутбуком.

4.2.1 Вимоги до приміщень

Геометричні розміри приміщення зазначені в табл. 4.1.

Таблиця 4.1 – Розміри офісного приміщення

Найменування	**Значення**
Довжина, м	8
Ширина, м	4
Висота, м	2,5
Площа, $м^2$	32
Об’єм, $м^3$	80

Згідно з ДСН 3.3.6.042-99 [4] розмір площі для одного робочого місця оператора персонального комп’ютера має бути не менше 6 кв. м, а об’єм — не менше 20 куб. м. Отже, дане приміщення цілком відповідає зазначеним нормам.

Для забезпечення потрібного рівного освітленості кімната має вікно та систему загального рівномірного освітлення, що встановлена на стелі. Для дотримання вимог пожежної безпеки допущена можливість швидкого виходу з приміщення через вікно з можливістю вентилювання. Також у приміщенні завжди наявна ковдра, яку можна використати при пожежі.

4.2.2 Вимоги до організації місця праці

При порівнянні відповідності характеристик робочого місця з нормативними основними вимогами до організації робочого місця за ДСанПіН 3.3.2.007-98 [5] (табл. 4.2) і відповідними фактичними значеннями для робочого місця, констатуємо повну відповідність.

Таблиця 4.2 – Характеристика робочого місця

Найменування параметра	Фактичне значення	Нормативне значення
Висота робочої поверхні, мм	700	680 ÷ 800
Висота простору для ніг, мм	650	не менше 600
Ширина простору для ніг, мм	630	не менше 500
Глибина простору для ніг, мм	670	не менше 650
Висота поверхні сидіння, мм	420	400 ÷ 500
Ширина сидіння, мм	400	не менше 400
Глибина сидіння, мм	400	не менше 400
Висота поверхні спинки, мм	500	не менше 300
Ширина опорної поверхні спинки, мм	470	не менше 380
Радіус кривини спинки в горизонтальній площині , мм	400	400
Відстань від очей до екрану дисплея, мм	720	700 ÷ 800

4.3 Виробнича санітарія

На підставі аналізу небезпечних та шкідливих факторів при виробництві (експлуатації), пожежної безпеки можуть бути надалі вирішені питання необхідності забезпечення працюючих достатньою кількістю освітлення, вентиляції повітря, організації заземлення, тощо.

4.3.1 Аналіз небезпечних та шкідливих факторів при виробництві (експлуатації) виробу

Аналіз небезпечних та шкідливих виробничих факторів виконується у табличній формі (табл. 4.3). Роботу, пов'язану з ПК з ВДТ, у тому числі на тих, які мають робочі місця, обладнані ПК з ВДТ і ПП, виконують із забезпеченням виконання НПАОП 0.00-7.15-18 [6], яке встановлюють вимоги

безпеки до обладнання робочих місць, до роботи із застосуванням ЕОМ з ВДТ і ПП. Переважно роботи за проектами виконують у кабінетах чи інших приміщеннях, де використовують різноманітне електрообладнання, зокрема персональні комп’ютери (ПК) та периферійні пристрої. Основними робочими характеристиками персонального комп'ютера є:

- робоча напруга U=+220В +-5%;
- робочий струм I=2А;
- споживана потужність Р=350 Вт.

Таблиця 4.3 – Аналіз небезпечних і шкідливих виробничих факторів

Небезпечні шкідливі виробничі фактори	Джерела факторів	Кількісна оцінка	Нормативні документи
1	2	3	4
Фізичні			
Підвищена або знижена вологість повітря	Експлуатація ЕОМ, принтерів, сканерів або серверного обладнання для роботи	3	[1]
Підвищений рівень напруги електричної мережі, замикання якої може відбуватися через тіло людини	Експлуатація ЕОМ, принтерів, сканерів або серверного обладнання для роботи	2	[2]
Недостатнє освітлення робочої зони	Порушення гігієнічних параметрів виробничого середовища	1	[4]
Психофізіологічні:			
Нервово-психічне перевантаження	- формулювання теми; - пошук інформація про предметну область; - виконання роботи; - оформлення записки.	4	[3]
Фізичні(статичне-сидіння)	Порушення умов праці організації робочого місця(безперервна робота)	2	[2] [3]

4.3.2 Пожежна безпека

Пожежна безпека при застосуванні ПК забезпечується:

1) системою запобігання пожежі;
2) системою протипожежного захисту;

3) організаційно-технічними заходами.

Потенційними джерелами запалювання можуть бути:

1) іскри і дуги короткого замикання;
2) електрична іскра при замиканні і розмиканні ланцюгів;
3) перегріви від тривалого перевантаження;
4) відкритий вогонь і продукти горіння;
5) наявність речовин, нагрітих вище за температуру самозаймання;
6) розрядна статична електрика.

Причинами можливого загоряння і пожежі можуть бути:

1) несправність електроустановки;
2) конструктивні недоліки устаткування;
3) коротке замикання в електричних мережах;
4) запалювання горючих матеріалів, що знаходяться в безпосередній близькості від електроустановки.

4.3.3 Електробезпека

На робочому місці виконуються наступні вимоги електробезпеки: ПК, периферійні пристрої та устаткування для обслуговування, електропроводи і кабелі за виконанням та ступенем захисту відповідають класу зони за ПУЕ (правила улаштування електроустановок), мають апаратуру захисту від струму короткого замикання та інших аварійних режимів. Лінія електромережі для живлення ПК, периферійних пристроїв і устаткування для обслуговування, виконана як окрема групова три-провідна мережа, шляхом прокладання фазового, нульового робочого та нульового захисного провідників. Нульовий захисний провідник використовується для заземлення (занулення) електроприймачів. Штепсельні з'єднання та електророзетки крім контактів фазового та нульового робочого провідників мають спеціальні контакти для підключення нульового захисного провідника. Електромережа штепсельних розеток для живлення персональних ПК, укладено по підлозі

поруч зі стінами відповідно до затвердженого плану розміщення обладнання та технічних характеристик обладнання. Металеві труби та гнучкі металеві рукави заземлені. Захисне заземлення включає в себе заземлюючих пристроїв і провідник, який з'єднує заземлюючий пристрій з обладнанням, яке заземлюється - заземлюючий провідник.

4.4 Гігієнічні вимоги до параметрів виробничого середовища

4.4.1 Освітлення

Розрахунок освітлення.

Для виробничих та адміністративних приміщень світловий коефіцієнт приймається не менше -1/8, в побутових – 1/10:

$$\sqrt{a^2 + b^2} \cdot \text{Sb} = \left(\frac{1}{8} \div \frac{1}{10}\right) \times \text{Sn}, \tag{4.1}$$

де S_b – площа віконних прорізів, м^2;

S_n – площа підлоги, м^2.

$S_n = a \cdot b = 8 \cdot 4 = 32$ м^2,

$S_{вік} = 1/8 \cdot 15 = 4$ м^2.

Приймаємо вікно площею S = 4 м^2.

Розрахунок штучного освітлення виробляється по коефіцієнтах використання світлового потоку, яким визначається потік, необхідний для створення заданої освітленості при загальному рівномірному освітленні. Розрахунок кількості світильників n виробляється за формулою (4.2):

$$n = \frac{E \cdot S \cdot Z \cdot K}{F \cdot U \cdot M} \tag{4.2}$$

де n – кількість світильників;

E – нормована освітленість робочої поверхні, визначається нормами – 300 лк;

S – освітлювана площа, м^2 (S = 15 м^2);

Z – поправочний коефіцієнт світильника (1.15 для ламп розжарювання та ДРЛ; 1.1 для люмінесцентних ламп), приймаємо рівним 1.1;

K – коефіцієнт запасу, що враховує зниження освітленості в процесі експлуатації – 1.5;

U – коефіцієнт використання, залежний від типу світильника, показника індексу приміщення і т.п., 0.575M ;

M – число люмінесцентних ламп в світильнику, 2 одиниці;

F – світловий потік - 5400лм (для ЛБ-80).

Підставивши числові значення у формулу (4.2), отримуємо:

$$n = \frac{300 \cdot 32 \cdot 1.1 \cdot 1.5}{5400 \cdot 0.575 \cdot 2} = 2.55 \approx 3$$

Приймаємо освітлювальну установку, яка складається з 3-х світильників, оснащених двома лампами типу ЛБ (одна – 80 Вт) зі світловим потоком 5400 лм.

4.5 Вентилювання

Здійснюється провітрювання приміщення, в залежності від погодних умов, тривалість повинна бути не менше 10 хв. Найкращий обмін повітря здійснюється при наскрізному провітрюванні.

4.6 Розрахунок захисного заземлення (забезпечення електробезпеки будівлі)

Згідно з класифікацією приміщень за ступенем небезпеки ураження електричним струмом НПАОП 40.1-1.01-97 [7], приміщення в якому проводяться всі роботи відноситься до першого класу (без підвищеної небезпеки). Під час роботи використовуються електроустановки з напругою живлення 36 В, 220 В, та 360 В. Опір контуру заземлення повинен мати не більше 4 Ом.

Послідовність розрахунку.

1) Визначається необхідний опір штучних заземлювачів $R_{шт.з.}$:

$$R_{шт.з.} = \frac{R_{\delta.} \cdot R_{пр.з.}}{R_{пр.з.} - R_{\delta.}}, \qquad (4.3)$$

де $R_{пр.з.}$ – опір природних заземлювачів;

Rδ – допустимий опір заземлення.

Якщо природні заземлювачі відсутні, то $R_{шт.з.} = R_{д.}$

Підставивши числові значення у формулу (4.3), отримуємо:

$$R_{шт.з.} = \frac{4 \cdot 40}{40 - 4} \approx 4 \text{ Ом}.$$

2) Опір заземлення в значній мірі залежить від питомого опору ґрунту ρ, Ом·м. Приблизне значення питомого опору глини приймаємо ρ = 40 Ом•м (табличне значення).

3) Розрахунковий питомий опір ґрунту, $P_{розр}$, Ом·м, визначається відповідно для вертикальних заземлювачів $P_{розр.в}$, і горизонтальних $P_{розр.г.}$, Ом·м за формулою:

$$P_{розр.} = \psi \cdot \rho, \qquad (4.4)$$

де ψ – коефіцієнт сезонності для вертикальних заземлювачів і кліматичної зони з нормальною вологістю землі, приймається для вертикальних заземлювачів $\rho_{розр.в} = 1.7$ і горизонтальних $\rho_{розр.г} = 5.5$ Ом·м.

$P_{розр.в.} = 1.7 \cdot 40 = 68$ Ом/м,

$P_{розр.г.} = 5.5 \cdot 40 = 220$ Ом/м.

4) Розраховується опір розтікання струму вертикального заземлювача $R_в$, Ом, за (4.5).

$$R_в = \left(\ln\frac{2 \cdot 1в}{dcT} + \frac{1}{2} \cdot \ln\frac{4 \cdot t + 1в}{4 \cdot t - 1в}\right), \quad (4.5)$$

де $l_в$ – довжина вертикального заземлювача (для труб - 2−3 м; $l_в$=3 м);

dст – діаметр стержня (для труб - 0,03−0,05 м; dст=0,05 м);

t – відстань від поверхні землі до середини заземлювача, яка визначається за ф. (4.6);

$$t = h_в + \frac{1_в}{2} \quad (4.6)$$

де $h_в$ – глибина закладання вертикальних заземлювачів (0,8 м); тоді

$t = 0.8 + \frac{3}{2} = 2{,}3$ м,

$$R_в = \frac{68}{2 \cdot \pi \cdot 3} \cdot \left(\ln\frac{2 \cdot 3}{0.05} + \frac{1}{2} \cdot \ln\frac{4 \cdot 2.3 + 3}{4 \cdot 2.3 - 3}\right) = 18.5 \text{ Ом}.$$

5) Визначається теоретична кількість вертикальних заземлювачів n штук, без урахування коефіцієнта використання ηв:

$$n = \frac{2 \cdot R_в}{R_д} = \frac{2 \cdot 18.5}{4} = 9.25. \quad (4.7)$$

6)Визначається необхідна кількість вертикальних заземлювачів з урахуванням коефіцієнта використання $n_в$:

$$n_в = \frac{2 \cdot R}{R_д \cdot n_в} = \frac{2 \cdot 18.5}{4 \cdot 0.57} = 16.2 \approx 16. \quad (4.8)$$

7)Визначається довжина з'єднувальної стрічки горизонтального заземлювача lc:

$$lc = 1.05 \cdot Lв \cdot (nв - 1) \quad (4.9)$$

де Lв – відстань між вертикальними заземлювачами, (прийняти за Lв = 3м);

nв – необхідна кількість вертикальних заземлювачів.

$lc = 1.05 \cdot 3 \cdot (16 - 1) \approx$ 48м.

8)Визначається опір розтіканню струму горизонтального заземлювача $R_г$, Ом:

$$R_г = \frac{P_{розр.г}}{2 \cdot \pi \cdot 1c} \cdot \ln \frac{2 \cdot 1^2}{d_{см} \cdot h}, \quad (4.10)$$

де $d_{см}$ – еквівалентний діаметр смуги шириною b, $d_{см}$ =0,95b, b = 0,15 м;

h_r – глибина закладання горизонтальних заземлювачів (0.5 м);

1_c – довжина з'єднувальної стрічки горизонтального заземлювача 1_c , м

$$R_г = \frac{220}{2 \cdot \pi \cdot 48} \cdot \ln \frac{2 \cdot 48^2}{0.95 \cdot 0.15 \cdot 0.5} = 8.1 \text{ Ом.}$$

9) Визначається коефіцієнт використання горизонтального заземлювача ηс. відповідно до необхідної кількості вертикальних заземлювачів nв.

Коефіцієнт використання з'єднувальної смуги η_c=0,3 (табличне значення).

10) Розраховується результуючий опір заземлювального електроду з урахуванням з'єднувальної смуги:

$$R_{заг} = \frac{R_в \cdot R_г}{R_в \cdot n_c \cdot R_г \cdot n_в \cdot n_в} \leq R_д. \qquad (4.11)$$

Висновок: дане захисне заземлення буде забезпечувати електробезпеку будівлі, так як виконується умова: Rзаг <4 Ом, а саме:

$$R_{заг} = \frac{18.5 \cdot 8.1}{18.5 \cdot 0.3 \cdot +8.1 \cdot 16 \cdot 0.57} = 1.9 \leq R_д.$$

4.7 Висновки до розділу 4

В результаті проведеної роботи було зроблено аналіз умов праці, шкідливих та небезпечних чинників, з якими стикається робітник. Було визначено параметри і певні характеристики приміщення для роботи над запропонованим проектом написаному в кваліфікаційній роботі, описано, які заходи потрібно зробити для того, щоб дане приміщення відповідало необхідним нормам і було комфортним і безпечним для робітника.

Приведені рекомендації щодо організації робочого місця, а також важливу інформацію щодо пожежної та електробезпеки. Були наведені розміри приміщення та значення температури, вологості й рухливості повітря, необхідна кількість ламп та інші параметри, значення яких впливає на умови праці робітника, а також – наведені інструкції з охорони праці, техніки безпеки при роботі на комп’ютері

4.8 Перелік джерел посилань до розділу 4

1. Закон України "Про охорону праці". Вводиться в дію Постановою ВР № 2695-XII від 14.10.92, ВВР, 1992, № 49, ст.669. – Режим доступу: https://zakon.rada.gov.ua/laws/show/2694-12.

2. Закон України "Про загальнообов'язкове державне соціальне страхування від нещасного випадку на виробництві та професійного захворювання, які спричинили втрату працездатності". Наказ від 21 грудня 2000 року N 2180-III. Режим доступу: https://zakon.rada.gov.ua/laws/show/1105-14.

3. Кодекс законів про працю України. Затверджується Законом № 322-VIII від 10.12.71 ВВР, 1971. Режим доступу: https://zakon.rada.gov.ua/laws/show/322-08.

4. Санітарні норми мікроклімату виробничих приміщень ДСН 3.3.6.042-99. Постанова N 42 від 01.12.99. Режим доступу: https://zakon.rada.gov.ua/rada/show/va042282-99.

5. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин ДСанПІН 3.3.2.007-98. Затверджено Постановою Головного державного санітарного лікаря України 10 грудня 1998 р. N 7. Режим доступу: www. URL: https://zakon.rada.gov.ua/rada/show/v0007282-98.

6. НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями». Зареєстровано в

Міністерстві юстиції України 25 квітня 2018 р. за № 508/31960. Режим доступу: https://zakon.rada.gov.ua/rada/show/v0007282-98.

7. НПАОП 40.1-1.01-97 «Про затвердження Правил безпечної експлуатації електроустановок». Зареєстровано в Міністерстві юстиції України 13 січня 1998 р. за № 11/2451 Режим доступу: https://zakon.rada.gov.ua/laws/show/z0011-98.

ВИСНОВКИ

У дипломному проекті здійснено розроблення програмного комплексу криптографічного перетворення інформації на основі алгоритмів симетричного шифруванняю.

З метою реалізації у першому розділі дипломного проекту розглянуті та проаналізовані сучасні системи симетричного кодування інформації, визначені поняття блокового шифру, ключа шифрування тощо. На основі проведено огляду, визначені шляхи реалізації проекту та алгоритми реалізаці, задачі для розроблення програмного комплексу. Проведений огляд математичних та логічних операцій, що використовуються в алгоритмах блочного шифрування інформації.

У другому розділі дипломного проекту визначено організаційну побудову програмних засобів захисту інформації.

Здійснено програмну реалізацію блокового симетричного шифру на основі логічної операції за модулем 2, реалізації основних операцій блокового шифруваня, проаналізовані апаратні можливості реалізації симетричних алгоритмів.

Розроблений програмний комплекс кодування інформації з використанням методів:

– простої заміни;

– логічної операції додавання по модулю 2;

– шифру Цезаря;

– шифру Віжинера;

– гамування зі зворотним зв’язком.

Реалізація проекту здійснена в інтегральному середовищі Microsoft Visual Studio 2019 з використанням мови програмування C++, об’єм програми 709 кБ. Результати даного проекту можуть бути використані у навчальному процесі кафедри комп’ютерних наук та інженерії при вивчені дисципліни «Захист інформації в комп’ютерних системах».

ДОДАТОК А

Таблиця Віжинера

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

ДОДАТОК Б

Шифрування по ДСТУ ГОСТ 28147-89:2009 в режимі простої заміни

```
// Req.cpp: определяет точку входа для консольного приложения.
//

#include "stdafx.h"


/*int _tmain(int argc, _TCHAR* argv[])
{
     return 0;
}*/

//---------------------------------------------------------------
-------------
//#include <vcl.h>
#include <stdio.h>
#include <conio.h>

//---------------------------------------------------------------
-------------

// визначаємо розмір файлу
long filesize(FILE *stream)
{
 long curpos, length;
 curpos = ftell(stream);
 fseek(stream, 0L, SEEK_END);
 length = ftell(stream);
 fseek(stream, curpos, SEEK_SET);
 return length;
}

// функція, яка реалізує роботу ДСТУ ГОСТ 28147-89 в режимі
простої заміни
void rpz(int rezh, char* opener, char* saver)
{
 FILE *f_begin, *f_end; // потоки для початкового та кінцевого
файлів

 // таблиця замін
 unsigned char Tab_Z[8][16] =
 {

0x0,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8,0x9,0xA,0xB,0xC,0xD,0xE,0xF,

0x0,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8,0x9,0xA,0xB,0xC,0xD,0xE,0xF,

0x0,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8,0x9,0xA,0xB,0xC,0xD,0xE,0xF,
```

```
0x0,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8,0x9,0xA,0xB,0xC,0xD,0xE,0xF,

0x0,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8,0x9,0xA,0xB,0xC,0xD,0xE,0xF,

0x0,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8,0x9,0xA,0xB,0xC,0xD,0xE,0xF,

0x0,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8,0x9,0xA,0xB,0xC,0xD,0xE,0xF,

0x0,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8,0x9,0xA,0xB,0xC,0xD,0xE,0xF
 };

// ключ
 unsigned long key[8] =
 {
  0x0123,
  0x4567,
  0x89AB,
  0xCDEF,
  0x0123,
  0x4567,
  0x89AB,
  0xCDEF
 };

 char N[4]; // 32-разрядный буфер,

 unsigned long n1=0, n2=0, SUM232=0; // буфер N1, N2 та суматор

 // відкриваєм файли
 f_begin = fopen (opener,"rb");
 f_end = fopen (saver,"wb");

 // визначаєм кількість блоків
 float blokoff;
 blokoff = 8*filesize(f_begin);
 blokoff = blokoff/64;
 int block = blokoff;
 if (blokoff-block>0) block++;

 int sh;
 if (filesize(f_begin)>=4) sh = 4; else sh = filesize(f_begin);
 int sh1 = 0;
 int flag=0;

 // починаємо зчитування та  перетворення блоків
 // перевіряємо на повноту блоків
 for (int i=0; i<block; i++)
 {
  // записуємо в буфер N1
  for (int q=0; q<4; q++) *((unsigned char *)&N+q) = 0x00;
  if ((sh1+sh)<filesize(f_begin))
  {
```

```
 fread (N,sh,1,f_begin);
 sh1+=sh;
}
else
{
 sh=filesize(f_begin)-sh1;
 fread (N,sh,1,f_begin);
 flag=1;
}
n1 = *((unsigned long *)&N);

// записуємо в буфер N2
for (int q=0; q<4; q++) *((unsigned char *)&N+q) = 0x00;
if ((sh1+sh)<filesize(f_begin))
{
 fread (N,sh,1,f_begin);
 sh1+=sh;
}
else
{
 if (flag==0)
 {
  sh=filesize(f_begin)-sh1;
  fread (N,sh,1,f_begin);
 }
}
n2 = *((unsigned long *)&N);

// 32 цикла простої заміни
// зчитуємо ключ
int c = 0;
for (int k=0; k<32; k++)
{
 if (rezh==1) { if (k==24) c = 7; }
  else { if (k==8) c = 7; }

 // додаємо в суматорі СМ1
 SUM232 = key[c] + n1;

 // замінюємо по таблиці замін
 unsigned char first_byte=0,second_byte=0,zam_symbol=0;
 int n = 7;
 for (int q=3; q>=0; q--)
 {
  zam_symbol = *((unsigned char *)&SUM232+q);
  first_byte = (zam_symbol & 0xF0) >> 4;
  second_byte = (zam_symbol & 0x0F);
  first_byte = Tab_Z[n][first_byte];
  n--;
  second_byte = Tab_Z[n][second_byte];
  n--;
  zam_symbol = (first_byte << 4) | second_byte;
  *((unsigned char *)&SUM232+q) = zam_symbol;
```

```
   }

   SUM232 = (SUM232<<11)|(SUM232>>21); // циклічний зсув на 11
розрядів
   SUM232 = n2^SUM232; // додаємо в суматорі СМ2

   if (k<31)
   {
    n2 = n1;
    n1 = SUM232;
   }
   if (rezh==1)
   {
    if (k<24)
    {
     c++;
     if (c>7) c = 0;
    }
    else
    {
     c--;
     if (c<0) c = 7;
    }
   }
   else
   {
    if (k<8)
    {
     c++;
     if (c>7) c = 0;
    }
   else
   {
    c--;
    if (c<0) c = 7;
   }
  }
 }
 n2 = SUM232;

 // вивід результату в файл
  char sym_rez;
  for (int q=0; q<=3; q++)
  {
   sym_rez = *((unsigned char *)&n1+q);
   fprintf(f_end, "%c", sym_rez);
  }
  for (int q=0; q<=3; q++)
  {
   sym_rez = *((unsigned char *)&n2+q);
   fprintf(f_end, "%c", sym_rez);
  }
 }
```

```
 fclose (f_begin);
 fclose (f_end);
}

//-------------------------------------------------------------------
-------------
int main()
{
 // обираємо шифрування або дешифрування
 int rezhim = 0;
 do
 {
  printf("Regimrabotu:\nShifrovanie - 1\nRasShifrovanie - 2\n");
   scanf("%d", &rezhim);
 } while ((rezhim!=1)&&(rezhim!=2)); // повторюємо до тих пір,
пока не будет введено
 // выбираем початковий та кінцевий файлы (слэш '\' в шлях
зарисати як '\\')
 char open_str[50], save_str[50];
 printf("\nvvedite pyt do isxodnogo faqla\n");
 scanf("%s", &open_str);
 printf("\n Vvedite pyt do faqla v kotoriq treb zapisatb
rez\n");
 scanf("%s", &save_str);

 rpz(rezhim, open_str, save_str); // запускаем РПЗ
 return 0;
}
//-------------------------------------------------------------------
-------------
```

ДОДАТОК В

Шифрування з використання логічної операції додавання по модулю 2

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace lr2
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        string Key, LongKey, InputText, OutputText;

        private void button1_Click(object sender, EventArgs e)
        {
            InputText = textBox1.Text;//исходное сообщение
            Key = textBox3.Text;//ключ
            LongKey = "";
            OutputText = "";
            char ch;
            int k = InputText.Length | Key.Length;// сколько раз
должен повториться цикл
            for (int i = 0; i <= k; i++)
            {
                LongKey = LongKey + Key;// Для повторения
кодировки в цикле
            }
            for (int i = 0; i < InputText.Length; i++)
            {
                {
                    ch = (char)((byte)(InputText[i]) ^
(byte)(LongKey[i])); //byte - код символа, умножаем  по модулю
2, и переводим в символьное
                }
                OutputText = OutputText + ch;
            }
            textBox2.Text = OutputText;// вывод закодированного
сообщения
        }

        private void button2_Click(object sender, EventArgs e)
```

```
        {
            InputText = textBox2.Text;
            Key = textBox3.Text;
            LongKey = "";
            OutputText = "";
            char ch;
            int k = InputText.Length | Key.Length;
            for (int i = 0; i <= k; i++)
            {
                LongKey = LongKey + Key;
            }
            for (int i = 0; i < InputText.Length; i++)
            {
                {
                    ch = (char)((byte)(InputText[i]) ^
(byte)(LongKey[i]));
                }
                OutputText = OutputText + ch;
            }
            textBox4.Text = OutputText;
        }}}
```

ДОДАТОК Д

Шифрування з використання шифру Цезаря

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Cezar
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void BtnOk_Click(object sender, EventArgs e)
        {
            string AlphabetIn =
"абвгдеёжзийклмнопрстуфхцчшщъыьэюя";
            int ALenght = AlphabetIn.Length;
            int shift = 3;
            if (TextBoxChoseShift.Text != "")
                shift = Convert.ToInt32(TextBoxChoseShift.Text);
            shift = (ALenght - shift % ALenght)%ALenght;
            char[] OldAlpabet = AlphabetIn.ToCharArray();
            char[] NewAlpabet = new char[ALenght];
            foreach (char Ch in OldAlpabet)
            {
                NewAlpabet[shift] = Ch;
                if (shift == ALenght -1)
                    shift = -1;
                shift++;
            }
            TextBoxOut.Text = "";

            string TextIn = TextBoxIn.Text;
                        foreach (char Ch in TextIn)
            {
                int ChNum = -1;
                ChNum = Array.IndexOf(OldAlpabet, Ch);

                if (ChNum == -1)
                    TextBoxOut.Text += Ch;
                else
             TextBoxOut.Text += NewAlpabet[ChNum];
            }}}}
```

ДОДАТОК Е

Шифрування з використання шифру Віженера

```
// projekt.cpp: определяет точку входа для консольного
приложения.
//

#include "stdafx.h"
#include <iostream>
#include <string>
#include <conio.h>
#include <stdlib.h>
#include <sstream>
#include <fstream>

using namespace std;
    int main()
    {
   int k; //Переменная выбора - шифрование/дешифрование
   string result = ""; //Строка - результат
   string key = ""; //Строка - ключ
  string key_on_s = "";
    bool flag;
    int x = 0, y = 0; //Координаты нового символа из таблицы
Виженера
   int registr = 0; //Регистр символа
   char dublicat; //Дубликат прописной буквы
   //Формирование таблицы Виженера на алфавите латиницы
   int shift = 0;
    char **tabula_recta = new char *[26]; //Таблица Виженера
    for (int i=0;i<26;i++)
    tabula_recta[i] = new char [26];
    string alfabet = "abcdefghijklmnopqrstuvwxyz"; //Алфавит
латиницы
    //Формирование таблицы
    for (int i = 0; i < 26; i++)
    for (int j = 0; j < 26; j++)
 {
           shift = j + i;
           if (shift >= 26) shift = shift % 26;
         tabula_recta[i][j] = alfabet[shift];
    }
    cout<<"Enter 1 for encryption and decryption of 2\n";
   cin>>k;
      switch (k) //Если k
        {
        case 1: //Если выбрано шифрование
            {
            cout<<"Enter key encryption\n";
            cin>>key;
            cout<<"Read of file...\n";
            setlocale(LC_ALL,"Russian");//Чтение файла
```

```
            string s; //Строка считанная из файла
            ifstream in("e:\Test.txt");
            getline(in,s);
            cout<<"Text of file: \n"<<s<<endl;
            in.close();
            cout<<"Reading complete!\n";
            cout<<"Encryption...\n";
            //Формирование строки, длиной шифруемой, состоящей
из повторений ключа
            for (int i = 0; i < s.length(); i++)
                {
                    key_on_s += key[i % key.length()];
                }
            //Шифрование при помощи таблицы
            for (int i = 0; i < s.length(); i++)
     {
           //Если нешифруемый символ
            if (((int)(s[i]) < 65) || ((int)(s[i]) > 122))
                    else
   {
                 //Поиск в первом столбце строки, начинающейся с
символа ключа
        int l = 0;
             flag = false;
                    //Пока не найден символ
                  while ((l < 26) && (flag == false))
         {
           //Если символ найден
                   if (key_on_s[i] == tabula_recta[l][0])
                   {
                 x = l;
                       flag = true;
                        }
                    l++;
                   }
                  //Уменьшаем временно регистр прописной буквы
                     if (((int)(s[i]) <= 90) && ((int)(s[i]) >=
65))
                    {
                         dublicat = (char)((int)(s[i]) + 32);
                      registr = 1;
                          }
                          else
                          {
                              registr = 0;
                              dublicat = s[i];
                          }
                          l = 0;
                          flag = false;
                          //Пока не найден столбец в первой
строке с символом строки
                          while ((l < 26) && (flag == false))
                          {
```

```
                                    if (dublicat ==
tabula_recta[0][l])
                                    {
     //Запоминаем номер столбца
         y = l;
           flag = true;
           }
            l++;
              }
       //Увеличиваем регистр буквы до прописной
   if (registr == 1)
         {
     //Изменяем символ на первоначальный регистр
          dublicat = char((int)(tabula_recta[x][y]) - 32);
         result += dublicat;
    }
       else
     result += tabula_recta[x][y];
      }
   }
cout<<"Encryption complete!\n";
cout<<"Result:\n";
cout<<result; //Вывод результата
}
case 2: //Если выбрано дешифрование
   {
  cout<<"Enter key decryption\n";
     cin>>key;
    cout<<"Read of file...\n";
    setlocale(LC_ALL,"Russian");
   string s;
    ifstream in("Test.txt");
   getline(in,s);
    cout<<"Text of file: \n"<<s<<endl;
    in.close();
   cout<<"Reading complete!\n";
   cout<<"Decryption...\n";
//Формирование строки, длиной дешифруемой, состоящей из
повторений ключа
            for (int i = 0; i < s.length(); i++)
                {
                    key_on_s += key[i % key.length()];
                }
            //Дешифрование при помощи таблицы
            for (int i = 0; i < s.length(); i++)
                {
                //Если недешифруемый символ
                if (((int)(s[i]) < 65) || ((int)(s[i]) > 122))
                   result += s[i];
               else
                 {
                    //Поиск в первом столбце строки,
начинающейся с символа ключа
```

```
                int l = 0;
              flag = false;
              //Пока не найден символ
                while ((l < 26) && (flag == false))
                 {
                    //Если символ найден
                  if (key_on_s[i] == tabula_recta[l][0])
                 {
                //Запоминаем в x номер строки
                    x = l;
                 flag = true;
              }
               l++;
           }
            //Уменьшаем временно регистр прописной буквы
        if (((int)(s[i]) <= 90) && ((int)(s[i]) >= 65))
     {
          dublicat = (char)((int)(s[i]) + 32);
         registr = 1;
        }
          else
              {
                   registr = 0;
                dublicat = s[i];
                          }
         l = 0;
         flag = false;
//Пока не найден столбец в x строке с символом строки
            while ((l < 26) && (flag == false))
           {
              if (dublicat == tabula_recta[x][l])
            {
             //Запоминаем номер столбца
                    y = l;
                flag = true;
              }
            l++;
           }
          //Увеличиваем регистр буквы до прописной
            if (registr == 1)
         {
         //Изменяем символ на первоначальный регистр
            dublicat = char((int)(tabula_recta[0][y]) - 32);
           result += dublicat;
       }
           else
              result += tabula_recta[0][y];
       }
    }
        cout<<"Decryption complete!\n";
        cout<<"Result:\n";
        cout<<result; //Вывод результата
        break;
```

```
            }
        default: //Если ошибочное значение
            {
            cout<<"Error value\n";
            break;
            }
        }
        getch();
        return 0;
    }

int _tmain(int argc, _TCHAR* argv[])
{
     return 0;
}
```

ДОДАТОК Ж

Гамування зі зворотним зв’язком

```
// XCrypt Win32 GUI version
// By Vlad Danylenko
// Released May 29, 2019

#include <windows.h>
#include <windowsx.h>
#include <stdio.h>
#include <commctrl.h>
#include "defmenu.h"

// Define control identifiers
#define IDC_FILE 1001
#define IDC_KEY 1002
#define IDC_BUTTON 1003
#define IDC_STATUS 1004
#define IDC_CHECK 1005
#define IDC_ABORT 1014
#define IDC_TFILE 1015
#define IDC_TKEY 1016
#define IDC_FBROWSE 1017
#define IDC_KBROWSE 1018
#define IDC_PBAR 1019
#define IDC_PB10 1020

//subroutines
void DisableMenus();
void EnableMenus();
void RstBttnStates();
void CloseFiles();
void CloseFilesKey();
void BrowseFWnd();
void ProgressBar();
//void helpmsgbox();

//Declare global variables
int filelength = 0;
int keylength = 0;
int keycount = 0;
char filename[65537];
char key[65537], keyupd, databyte, encrypt, filecache[] =
"enccache";
FILE *file, *cache, *keyfile;
HFILE filesize;

bool bAbort=false,bPB10=true,bEncrypting=false;
int flbin=0, fltxt=0;
char cBrowse[_MAX_PATH],cProgress[100];
DWORD dwSize=0, dwBytecount=0, dwPercent=0,dwPercentOld=0,
dwUpdcount;
```

```
HWND hwnd;               /* This is the handle for our window */
MSG msgHandler; //Message handler
OPENFILENAME ofn;//Open File dialog
MSGBOXPARAMS msgbox;//For message boxes
//static HINSTANCE hInstance = NULL; //revision
HINSTANCE hThisInstance; //revision

char szClassName[] = "ControlClasses";

// A function to set a control's text to the default font
int SetDefaultFont(int identifier, HWND hwnd)
{
    SendDlgItemMessage(
    hwnd,
    identifier,
    WM_SETFONT,
    (WPARAM)GetStockObject(DEFAULT_GUI_FONT),
    MAKELPARAM(TRUE, 0));
    return 0;
}

// Create a file browser dialog
    HWND hOpenFile;
//Focus controller
    HWND g_hwndLastFocus;

// A function to create static text
HWND CreateStatic(char* tempText, int x, int y, int width, int
height, int identifier, HWND hwnd)
{
    HWND hStaticTemp;

    hStaticTemp = CreateWindowEx(
    0,
    "STATIC",
    tempText,
    WS_CHILD | WS_VISIBLE,
    x, y,
    width, height,
    hwnd, (HMENU)identifier, hThisInstance, NULL);

    return hStaticTemp;
}
// A function to create a textarea
HWND CreateEdit(char* tempText, int x, int y, int width, int
height, int identifier, HWND hwnd)
{
    HWND hEditTemp;

    hEditTemp = CreateWindowEx(

    WS_EX_CLIENTEDGE,
```

```
    "EDIT",
    tempText,
    WS_CHILD | WS_VISIBLE | ES_LEFT | ES_AUTOHSCROLL |
WS_TABSTOP,// ES_LOWERCASE | ES_MULTILINE
    x, y,
    width, height,
    hwnd, (HMENU)identifier, hThisInstance, NULL);

    g_hwndLastFocus = hEditTemp;//for user tab presses
    return hEditTemp;
}
// A function to create a button
HWND CreateButton(char* tempText, int x, int y, int width, int
height, int identifier, HWND hwnd)
{
    HWND hButtonTemp;

    hButtonTemp = CreateWindowEx(
    0,
    "BUTTON",
    tempText,
    WS_CHILD | WS_VISIBLE | WS_TABSTOP,
    x, y,
    width, height,
    hwnd, (HMENU)identifier, hThisInstance, NULL);

    return hButtonTemp;
}

// A function to create a checkbox
HWND CreateCheck(char* tempText, int x, int y, int width, int
height, int identifier, HWND hwnd)
{
    HWND hCheckTemp;

    hCheckTemp = CreateWindowEx(
    0,
    "BUTTON",
    tempText,
    WS_CHILD | WS_VISIBLE | BS_AUTOCHECKBOX | WS_TABSTOP,
    x, y,
    width, height,
    hwnd, (HMENU)identifier, hThisInstance, NULL);

    return hCheckTemp;
}

// A function to handle dialog boxes
BOOL CALLBACK DlgProc(HWND hwnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
     switch(message)     {
          case WM_INITDIALOG:
```

```
{
 SetDlgItemText(hwnd,IDC_EDIT,"XCrypt is a XOR file encryption
tool. Use XCrypt to encrypt/decrypt any file type. A user
specified key is needed to encrypt/decrypt a file. Always
remember your key so you can decrypt the file you have
encrypted.\r\n\r\nEncryption Modes:\r\n\r\nStandard:\r\n   Uses
standard XOR encryption.\r\n\r\nBinary:\r\n   Improves
encryption in binary files when a small key is\r\n   used by not
encrypting or creating null characters\r\n   and
spaces.\r\n\r\nText:\r\n   Improves encryption in text files
when a small key is\r\n   used by not encrypting or creating
spaces.\r\n\r\nThe same encryption mode must be enabled to
decrypt a file. Ensure that the encryption file is not read-
only.\r\n\r\n\r\nAdditional Info:\r\n\r\nBinary and Text modes
ensure that a small key such as one that is 10 bytes in length
is not revealed in any obvious form, but comes at the cost of
revealing a small number of bytes of the encryption
file.\r\n\r\nIn Binary mode, by not encrypting or creating null
characters and spaces, single bytes of a key will be dispersed
randomly throughout the file where a key-byte equals a data-
byte. With standard XOR encryption a small key can end up being
revealed in its entirety in certain places.\r\n\r\nIn Text mode
the key will not be revealed in any place, but at the cost of
revealing where spaces exist.\r\n\r\nThe strength of encryption
depends on the size of the key in relation to the size of the
encryption file. If the key is close to the same size as the
encryption file, it is impossible to decrypt without knowing the
key.\r\n\r\nMultiple keys can be used for encryption. To decrypt
using multiple keys, apply the keys in reverse order that they
were originally applied for encryption.\r\n\r\n\r\nThank you for
using XCrypt.");
                }
          return TRUE;

          case WM_COMMAND:
               switch(LOWORD(wParam))
               {
                    case IDOK:
                         EndDialog(hwnd, IDOK);
                    break;
                    case IDCANCEL:
                         EndDialog(hwnd, IDOK);
                    break;
               }
          break;
          default:
               return FALSE;
     }
     return TRUE;
}

// subroutines to get tab focusing to work
void OnSetFocus(HWND hwnd, HWND hwndOldFocus)
```

```
{
    if (g_hwndLastFocus) {
        SetFocus(g_hwndLastFocus);
    }
}
void OnActivate(HWND hwnd, UINT state,
                HWND hwndActDeact, BOOL fMinimized)
{
    if (state == WA_INACTIVE) {
        g_hwndLastFocus = GetFocus();
    }
}


// The window procedure
HMENU menu;              /* Handle of the menu */
HWND hwndPBar;
    // Control handles -- all were originally static and inside
windowproc
HWND hFile;
HWND hKey;
HWND hStatus;
HWND hwndButton;
HWND hwndAbort;
HWND hwndFBrowse;
HWND hwndKBrowse;
HWND hwndCheck;
HWND hwndTFile;
HWND hwndTKey;
HWND hwndPB10;

LRESULT CALLBACK WindowProcedure(HWND hwnd, UINT message, WPARAM
wParam, LPARAM lParam)
{
    switch (message){
        case WM_CREATE:
        // Creates a static text box
        hStatus = CreateStatic("Enter file name and encryption
key.", 25, 10, 210, 20, IDC_STATUS, hwnd);
        SetDefaultFont(IDC_STATUS, hwnd);
        // Creates a textarea to input encryption file
        hFile = CreateEdit("", 25, 30, 185, 20, IDC_FILE, hwnd);
        SetDefaultFont(IDC_FILE, hwnd);
        SetFocus(hFile);// sets focus for a control
        //SendMessage(hFile,
WM_SETFONT,(WPARAM)GetStockObject(DEFAULT_GUI_FONT),0); //for
reference
        // Creates a button labeled ".." for browsing encryption
file
        hwndFBrowse = CreateButton("..", 211, 32, 8, 16,
IDC_FBROWSE, hwnd);
        //Creates a textarea to input keyfile
        hKey = CreateEdit("", 25, 52, 185, 20, IDC_KEY, hwnd);
```

```
        SetDefaultFont(IDC_KEY, hwnd);
        // Creates a button labeled ".." for browsing keyfile
        hwndKBrowse = CreateButton("..", 211, 54, 8, 16,
IDC_KBROWSE, hwnd);
        // Creates a checkbox
        hwndCheck = CreateCheck("Key is a file", 130, 77, 75,
20, IDC_CHECK, hwnd);
        SetDefaultFont(IDC_CHECK, hwnd);
        // Creates a button labeled "Encrypt"
        hwndButton = CreateButton("Encrypt", 25, 75, 85, 25,
IDC_BUTTON, hwnd);
        SetDefaultFont(IDC_BUTTON, hwnd);
        // Creates a button labeled "Abort"
        hwndAbort = CreateButton("Abort", 25, 75, 85, 25,
IDC_ABORT, hwnd);
        SetDefaultFont(IDC_ABORT, hwnd);
        ShowWindow(hwndAbort, SW_HIDE);
        //create a button for disabling/enabling progress bar
        hwndPB10 = CreateButton("X", 211, 103, 8, 8, IDC_PB10,
hwnd);
        SetDefaultFont(IDC_PB10, hwnd);
        //static text box - File
        hwndTFile = CreateStatic("File:", 3, 33, 22, 20,
IDC_TFILE, hwnd);
        SetDefaultFont(IDC_TFILE, hwnd);
        //static text box - Key
        hwndTKey = CreateStatic("Key:", 3, 55, 22, 20, IDC_TKEY,
hwnd);
        SetDefaultFont(IDC_TKEY, hwnd);
        //create area for progress bar
        hwndPBar =
CreateWindowEx(WS_EX_CLIENTEDGE,PROGRESS_CLASS,"",WS_CHILD|WS_VI
SIBLE,25,103,185,8,hwnd,NULL,hThisInstance,NULL);
        SendMessage(hwndPBar, PBM_SETRANGE, 0,
MAKELPARAM(0,100));

        HANDLE_MSG(hwnd, WM_ACTIVATE, OnActivate);// part of the
tab functions
        HANDLE_MSG(hwnd, WM_SETFOCUS, OnSetFocus);// part of the
tab functions

        break;

        case WM_COMMAND:
        {
             //switch( wParam )
             switch(LOWORD(wParam))
             {
                     case IDM_STANDARD:
                          flbin=0, fltxt=0;
                          menu = LoadMenu(hThisInstance,
MAKEINTRESOURCE(ID_MENU));
                          SetMenu(hwnd, menu);
```

```
                              CheckMenuItem(menu, IDM_STANDARD,
MF_CHECKED);
                              SetWindowText(hStatus, "Standard
encryption mode enabled.");
                              return 0;
                         case IDM_BINARY:
                              flbin=1, fltxt=0;
                              menu = LoadMenu(hThisInstance,
MAKEINTRESOURCE(ID_MENU));
                              SetMenu(hwnd, menu);
                              CheckMenuItem(menu, IDM_BINARY,
MF_CHECKED);
                              SetWindowText(hStatus, "Binary file
mode enabled.");
                              return 0;
                         case IDM_TEXT:
                              flbin=0, fltxt=1;
                              menu = LoadMenu(hThisInstance,
MAKEINTRESOURCE(ID_MENU));
                              SetMenu(hwnd, menu);
                              CheckMenuItem(menu, IDM_TEXT,
MF_CHECKED);
                              SetWindowText(hStatus, "Text file mode
enabled.");
                              return 0;
                         case IDM_XCRYPT:
                              DialogBox(GetModuleHandle(NULL),
MAKEINTRESOURCE(IDD_XCRYPT), hwnd, DlgProc);
       /* msgbox for reference   MessageBox( hwnd, (LPSTR)
"\nXCrypt is a XOR file encryption tool.\n\nUse XCrypt to
encrypt/decrypt any file type. A user specified key is needed to
encrypt/decrypt a file.\nAlways remember your key so you can
decrypt the file you have encrypted.\n\nEncryption modes:\n\n
Standard:    Uses standard XOR encryption.\n\n      Binary:
Improves encryption in binary files when a small key is
used.\n\n      Text:           Improves encryption in text files
when a small key is used.\n\nThe same encryption mode must be
enabled to decrypt a file.",
                                        "XCrypt",
                                          // (LPSTR) szClassName,
                                           MB_ICONINFORMATION | MB_OK
);*/
                              return 0;
                         case IDM_ABOUT:
                                DialogBox(GetModuleHandle(NULL),
MAKEINTRESOURCE(IDD_ABOUT), hwnd, DlgProc);
                                //msgbox.lpszText = "XCrypt was
written by Chris Mailloux. mailloux@gmail.com"; //for reference
                               // MessageBoxIndirect(&msgbox); //for
msgbox reference
                              return 0;

                 case IDC_FBROWSE:
```

```
                    ofn.lpstrTitle = TEXT("Select Encryption
File");
                    GetOpenFileName(&ofn);//call file browser
function
                    if(cBrowse[0] != '\0')
                        SetWindowText(hFile, cBrowse);//update
input box
                    cBrowse[0] = '\0';
                    return 0;
                case IDC_KBROWSE:

SendMessage(hwndCheck,BM_SETCHECK,BST_CHECKED,0);//check the
checkbox
                    ofn.lpstrTitle = TEXT("Select Keyfile");
                    GetOpenFileName(&ofn);//call file browser
function
                    if(cBrowse[0] != '\0')
                        SetWindowText(hKey, cBrowse);//update
input box
                    cBrowse[0] = '\0';
                    return 0;

        // If a button is clicked...
                case IDC_BUTTON:
                {
                filelength = GetWindowTextLength(hFile);
                    keylength = GetWindowTextLength(hKey);
                    keycount = 0;
                dwSize=0, dwBytecount=0, dwPercent=0,
dwPercentOld=0, dwUpdcount=0;
                SendMessage(hwndPBar,PBM_SETPOS,(WPARAM)0,0);

                LRESULT checkState; //checks state of checkbox

                    // ...Then get the text from the text area..
                GetWindowText(hFile, filename, filelength + 1);
                filename[filelength] = '\0';
                GetWindowText(hKey, key, keylength + 1);
                key[keylength] = '\0';

                filesize = (HFILE)CreateFile(filename,
GENERIC_READ, FILE_SHARE_READ, NULL, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, 0);
                dwSize = GetFileSize((void*)filesize,NULL);//
Obtain encryption file size
                CloseHandle((void *)filesize);
                //if (dwSize == 0xFFFFFFFF)
                //    return 0;

                file = fopen(filename, "rb");
                cache = fopen(filecache, "wb");
                if(file == 0){
```

```
                    SetWindowText(hStatus, "Error opening
file.");
                    MessageBeep(0);
                     fclose (cache);
                     remove (filecache);
                     break;
                   }
                if(cache == 0){
                     SetWindowText(hStatus, "Error creating
cache file.");
                     MessageBeep(0);
                      fclose(file);
                      break;
                   }

                SetWindowText(hStatus, "Encrypting... Please
wait.");
                bEncrypting = true;
                EnableWindow(hwndCheck, false); // disable
checkbox
                ShowWindow(hwndButton, SW_HIDE); // hide encrypt
button
                ShowWindow(hwndAbort, SW_SHOW);  // show abort
button
                DisableMenus();   // call subroutine
DisableMenus
                SetFocus(hwndAbort);

                // Check the checkbox...
                checkState =
SendDlgItemMessage(hwnd,IDC_CHECK,BM_GETCHECK,0, 0);

                // ...if it's not checked...
                if(checkState != BST_CHECKED){
                        while (!feof(file)){
                       if(key[keycount] == 0)
                           keycount = 0;
                             databyte = fgetc(file);
                             encrypt = databyte ^ key[keycount];
                          if(flbin == 1){
                           if(encrypt == key[keycount] ||
encrypt == 0 || encrypt == ' ' || databyte == ' ')
                               encrypt = databyte;
                          }
                          if(fltxt == 1){
                               if(encrypt == ' ' || databyte ==
' ')
                                   encrypt = databyte;
                          }
                             if(!feof(file)){
                                  fputc(encrypt, cache);
                                  keycount++;
                                  ProgressBar();
```

```
                                }
                                if(ferror(cache)){
                                  SetWindowText(hStatus, "Aborted.
Insufficient disk space.");
                              CloseFiles();
                              RstBttnStates(); // restore original
button states
                              EnableMenus(); // Enable menus

                                  return 0;
                             }

                                //Check for user commands
                                if(PeekMessage(&msgHandler, NULL,
0, 0, PM_REMOVE))
                                {
                           //   if (IsDialogMessage(hwnd,
&msgHandler)) {//this is for keeping track of tab presses by the
user
                              /* Already handled by dialog manager
*/   // but it slows down encryption about 8%
                             /* } else {
// not needed, leaving it for reference
                                  TranslateMessage(&msgHandler);
                                  DispatchMessage(&msgHandler);
                              }*/

                                     TranslateMessage(&msgHandler);
                                     DispatchMessage(&msgHandler);
                                }
                           if(bAbort){
                                  SetWindowText(hStatus, "Encrytion
Aborted.");
                                  CloseFiles();
                              bAbort = false;

SendMessage(hwndPBar,PBM_SETPOS,(WPARAM)0,0);
                              RstBttnStates(); // restore original
button states
                              EnableMenus(); // Enable menus

                                 return 0;
                            }

                          if(dwPercentOld != dwPercent &&
bPB10==true){
                           dwPercentOld = dwPercent;
                           sprintf(cProgress, "Encrypting... %d
Percent", dwPercent);
                           SetWindowText(hStatus, cProgress);
                           }

                            }
```

```
                }

                // ...if its checked...
                else if(checkState == BST_CHECKED){

                    if(strcmp(key,filename)==0){
                          SetWindowText(hStatus, "Encryption file
cannot be same as Keyfile.");
                          CloseFiles();

SendMessage(hwndPBar,PBM_SETPOS,(WPARAM)0,0);
                       RstBttnStates(); // restore original
button states
                       EnableMenus(); // Enable menus

                       return 0;//break;
                     }

                    keyfile = fopen(key, "rb");
                    if(keyfile == 0){
                        SetWindowText(hStatus, "Error opening
keyfile.");
                        CloseFiles();
                        RstBttnStates(); // restore original
button states
                        EnableMenus(); // Enable menus
                           return 0;
                    }
                    while (!feof(file)){
                           keyupd = fgetc(keyfile);
                           if(ferror(keyfile)){
                                SetWindowText(hStatus, "Error
reading keyfile.");
                                CloseFilesKey();
                            RstBttnStates(); // restore original
button states
                            EnableMenus(); // Enable menus

                                return 0;
                        }
                        if(!feof(keyfile)){
                            databyte = fgetc(file);
                               encrypt = databyte ^ keyupd;
                              if(flbin == 1){
                                 if(encrypt == keyupd || encrypt
== 0 || encrypt == ' ' || databyte == ' ')// dont read in nulls,
dont create nulls
                                              encrypt = databyte;
                                }
                                if(fltxt == 1){
                                     if(encrypt == ' ' ||
databyte == ' ')
                                              encrypt = databyte;
```

```
                                    }
                                    if(!feof(file)){
                                         fputc(encrypt, cache);
                                         ProgressBar();
                                }
                           }
                           if(ferror(cache)){
                                SetWindowText(hStatus, "Aborted.
Insufficient disk space.");
                                CloseFilesKey();
                            RstBttnStates(); // restore original
button states
                            EnableMenus(); // Enable menus

                               return 0;
                           }
                           if(feof(keyfile)){
                                fclose(keyfile);
                                keyfile = fopen(key, "rb");
                           }

                           //Check for user commands
                                if(PeekMessage(&msgHandler, NULL,
0, 0, PM_REMOVE))
                                {

                                   TranslateMessage(&msgHandler);
                                   DispatchMessage(&msgHandler);
                                }
                           if(bAbort){
                                SetWindowText(hStatus, "Encrytion
Aborted.");
                                CloseFilesKey();
                                   bAbort = false;

     SendMessage(hwndPBar,PBM_SETPOS,(WPARAM)0,0);
                            RstBttnStates(); // restore original
button states
                            EnableMenus(); // Enable menus

                               return 0;
                           }

                           if(dwPercentOld != dwPercent &&
bPB10==true){
                           dwPercentOld = dwPercent;
                           sprintf(cProgress, "Encrypting... %d
Percent", dwPercent);
                           SetWindowText(hStatus, cProgress);
                           }

                      }
                     fclose(keyfile);
```

```
                }

                    fclose(file);
                    fclose (cache);
                    remove (filename);
                    rename (filecache,filename);
                SetWindowText(hStatus, "Done.");
                bEncrypting = false;
                SendMessage(hwndPBar,PBM_SETPOS,(WPARAM)100,0);
                RstBttnStates(); // restore original button
states
                EnableMenus(); // Enable menus
                MessageBeep(0);

                }break;

        case IDC_PB10://enables/disables progress updates
             if(bPB10 == true){
                 if(bEncrypting == true)
                     SetWindowText(hStatus, "Encrypting...
Please wait.");
                     ShowWindow(hwndPBar, SW_HIDE);
                 bPB10 = false;
             }
             else if(bPB10 == false){
                 bPB10 = true;
                 ShowWindow(hwndPBar, SW_SHOW);
             }
             break;
        case IDC_ABORT:
            bAbort = true;

        }break;

    //part of tab detection
    case DM_GETDEFID: break;
    case DM_SETDEFID: break;
        }

        case WM_DESTROY:
            if(bEncrypting == true){
                LRESULT checkState;
                checkState =
SendDlgItemMessage(hwnd,IDC_CHECK,BM_GETCHECK,0, 0);
                if(checkState == BST_CHECKED)
                    CloseFilesKey();
                if(checkState != BST_CHECKED)
                    CloseFiles();
            }
            PostQuitMessage(0);
            break;

        default:
```

```
            return DefWindowProc(hwnd, message, wParam, lParam);

    }

    return 0;
}

int WINAPI WinMain(HINSTANCE hThisInstance, HINSTANCE
hPrevInstance, LPSTR lpszArgument, int nFunsterStil)
{
   // HWND hwnd;               /* This is the handle for our
window */
    MSG messages;            /* Here messages to the application
are saved */
    WNDCLASSEX wincl;        /* Data structure for the
windowclass */
    HMENU menu;              /* Handle of the menu */
    /* The Window structure */
    wincl.hInstance = hThisInstance;
    wincl.lpszClassName = szClassName;
    wincl.lpfnWndProc = WindowProcedure;      /* This function
is called by windows */
    wincl.style = CS_DBLCLKS;                 /* Catch double-
clicks */
    wincl.cbSize = sizeof (WNDCLASSEX);

    /* Use default icon and mouse-pointer */
    wincl.hIcon = LoadIcon (hThisInstance,
MAKEINTRESOURCE(IDI_ICON));
    wincl.hIconSm = LoadIcon (hThisInstance,
MAKEINTRESOURCE(IDI_ICON));
    wincl.hCursor = LoadCursor (NULL, IDC_ARROW);
    wincl.lpszMenuName = NULL; /* No menu */
    wincl.lpszMenuName = NULL;                 /* No menu */
    wincl.cbClsExtra = 0;                      /* No extra bytes
after the window class */
    wincl.cbWndExtra = 0;                      /* structure or
the window instance */
    /* Use Windows's default color as the background of the
window */
    wincl.hbrBackground = (HBRUSH) COLOR_BTNSHADOW;

    /* Register the window class, and if it fails quit the
program */
    if (!RegisterClassEx (&wincl))
        return 0;

    /* The class is registered, let's create the program*/
    hwnd = CreateWindowEx (
           0,                  /* Extended possibilites for
variation */
           szClassName,        /* Classname */
```

```
           "XCrypt",// - XOR File Encryption Tool",       /*
Title Text */
           WS_MINIMIZEBOX | WS_SYSMENU, /*window without
maxbutton. WS_OVERLAPPEDWINDOW = default window */
           CW_USEDEFAULT,       /* Windows decides the position
*/ //CW_USEDEFAULT
           CW_USEDEFAULT,       /* where the window ends up on
the screen */
           230,                 /* The programs width */
           163,                 /* and height in pixels */
           HWND_DESKTOP,        /* The window is a child-window
to desktop */
           NULL,                /* No menu */
           hThisInstance,       /* Program Instance handler */
           NULL
    );
    ShowWindow(hwnd, nFunsterStil);

    menu = LoadMenu(hThisInstance, MAKEINTRESOURCE(ID_MENU));
    SetMenu(hwnd, menu);
    CheckMenuItem(menu, IDM_STANDARD, MF_CHECKED);

    BrowseFWnd();    // call browse window structure
//    helpmsgbox();    // call messagebox structure

  /*  while(GetMessage(&messages, NULL, 0, 0))
    {
        TranslateMessage(&messages);
        DispatchMessage(&messages);
    }*/
    while (GetMessage(&messages, NULL, 0, 0)) {
        if (IsDialogMessage(hwnd, &messages)) {
            /* Already handled by dialog manager */
        } else {
            TranslateMessage(&messages);
            DispatchMessage(&messages);
        }
    }
    return messages.wParam;
}

// Subroutines
void CloseFiles(){
    MessageBeep(0);
    fclose (file);
    fclose (cache);
    remove (filecache);
    bEncrypting = false;
}
void CloseFilesKey(){
    MessageBeep(0);
    fclose (file);
    fclose (cache);
```

```
    fclose (keyfile);
    remove (filecache);
    bEncrypting = false;
}
void DisableMenus(){
     menu = LoadMenu(hThisInstance, MAKEINTRESOURCE(ID_MENU));
     SetMenu(hwnd, menu);
     EnableMenuItem(menu,IDM_STANDARD, MF_GRAYED);
     EnableMenuItem(menu,IDM_BINARY, MF_GRAYED);
     EnableMenuItem(menu,IDM_TEXT, MF_GRAYED);
     EnableMenuItem(menu,IDM_XCRYPT, MF_GRAYED);
     EnableMenuItem(menu,IDM_ABOUT, MF_GRAYED);
     if(flbin == 0 && fltxt == 0){
         CheckMenuItem(menu, IDM_STANDARD, MF_CHECKED);
     }
     if(flbin == 1 && fltxt == 0){
         CheckMenuItem(menu, IDM_BINARY, MF_CHECKED);
     }
     if(flbin == 0 && fltxt == 1){
         CheckMenuItem(menu, IDM_TEXT, MF_CHECKED);
     }
}
void EnableMenus(){
     menu = LoadMenu(hThisInstance, MAKEINTRESOURCE(ID_MENU));
     SetMenu(hwnd, menu);
     if(flbin == 0 && fltxt == 0){
         CheckMenuItem(menu, IDM_STANDARD, MF_CHECKED);
     }
     if(flbin == 1 && fltxt == 0){
         CheckMenuItem(menu, IDM_BINARY, MF_CHECKED);
     }
     if(flbin == 0 && fltxt == 1){
         CheckMenuItem(menu, IDM_TEXT, MF_CHECKED);
     }
}
void RstBttnStates(){
    SetFocus(hwndButton);// sets focus for a control
    EnableWindow(hwndCheck, true); // enable checkbox
    ShowWindow(hwndButton, SW_SHOW); // show encrypt button
    ShowWindow(hwndAbort, SW_HIDE);  // hide abort button
}
void BrowseFWnd(){
         //Openfilename function
          ofn.lStructSize       = sizeof (OPENFILENAME);
          ofn.hwndOwner         = hwnd;
          ofn.hInstance         = NULL;
          ofn.lpstrFilter       = "All Files (*.*)\0*.*\0\0";
          ofn.lpstrCustomFilter = NULL;
          ofn.nMaxCustFilter    = 0;
          ofn.nFilterIndex      = 0;
          ofn.lpstrFile         = cBrowse;
          ofn.nMaxFile          = _MAX_PATH;
     //   ofn.lpstrFileTitle     //not needed
```

```
          ofn.nMaxFileTitle     = _MAX_FNAME + _MAX_EXT;
          ofn.lpstrInitialDir   = NULL;
//        ofn.lpstrTitle        = TEXT("Select File"); //defined
elsewhere
          ofn.Flags             = OFN_HIDEREADONLY |
OFN_FILEMUSTEXIST;
          ofn.nFileOffset       = 0;
          ofn.nFileExtension    = 0;
          ofn.lpstrDefExt       = NULL;
          ofn.lCustData         = 0L;
          ofn.lpfnHook          = NULL;
          ofn.lpTemplateName    = NULL;
}
void ProgressBar(){
    dwBytecount++;
    dwUpdcount++;
    if(bPB10 == true && dwUpdcount >= 100000){
        dwPercent = (dwBytecount*100)/dwSize;
        SendMessage(hwndPBar,PBM_SETPOS,(WPARAM)dwPercent,0);
        dwUpdcount=0;
    }
}
/*void helpmsgbox(){ // a function that can handle messageboxes,
not used.
    msgbox.cbSize = sizeof(MSGBOXPARAMS);
    msgbox.hwndOwner = hwnd;
    msgbox.hInstance = hThisInstance;
    //msgbox.lpszText = ;
    msgbox.lpszCaption = "XCrypt";
    msgbox.dwStyle = MB_OK | MB_SETFOREGROUND | MB_USERICON;
    msgbox.lpszIcon = MAKEINTRESOURCE(IDI_ICON);
    //  HICON icon = (HICON)LoadImage(hThisInstance,
MAKEINTRESOURCE(IDI_ICON), IMAGE_ICON, 0, 0, LR_SHARED);
   // msgbox.lpszIcon =
LoadImage(hThisInstance,MAKEINTRESOURCE(IDI_ICON),IMAGE_ICON,0,0
,LR_DEFAULTCOLOR);
}*/
```

ДОДАТОК К

ПРЕЗЕНТАЦІЯ

Міністерство освіти і науки України
Східноукраїнський національний університет ім. Володимира Даля

«Програмний комплекс криптографічного перетворення інформації на основі симетричного шифрування»

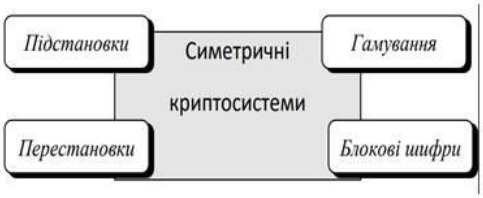
Студент гр. КІ – 16 бд Медвєдєв Б. М.
Керівник Кардашук В. С.

Актуальність теми

- Причини вразливості інформації - зростання обсягів збережених і переданих даних; розширення кола користувачів, що мають доступ до інформаційних систем; ускладнення інформаційних систем.
-
- Напрям розроблення системи стандартів із захисту даних, що зберігаються і обробляються на персональних комп’ютерах, від несанкціонованого доступу визнано однією із пріоритетних областей інформаційних технологій.
- Створення надійного криптографічного алгоритму та систем – складна задача, оскільки необхідно враховувати тенденції розвитку комп'ютерної техніки, а також фактори, що потенційно можуть знизити його стійкість в майбутньому.
- Мета дипломного проекту – розроблення програмного комплексу криптографічного перетворення на основі симетричного шифрування.

Класифікація криптографічних алгоритмів

Методи перетворення інформації в симетричних системах



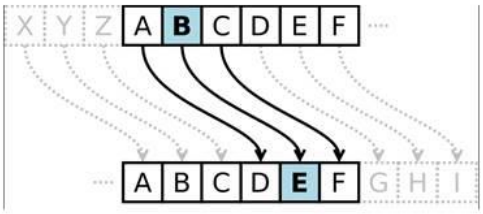
Методи перетворення інформації

Шифр Гронсфельда

Шифр Віжинера

Шифр "подвійний квадрат" Уітстона

Шифр Цезаря



Шифр на основі логічної операції за модулем 2

Шифр Вернама

AES шифрування

Шифрування по ДСТУ ГОСТ 28147:2009

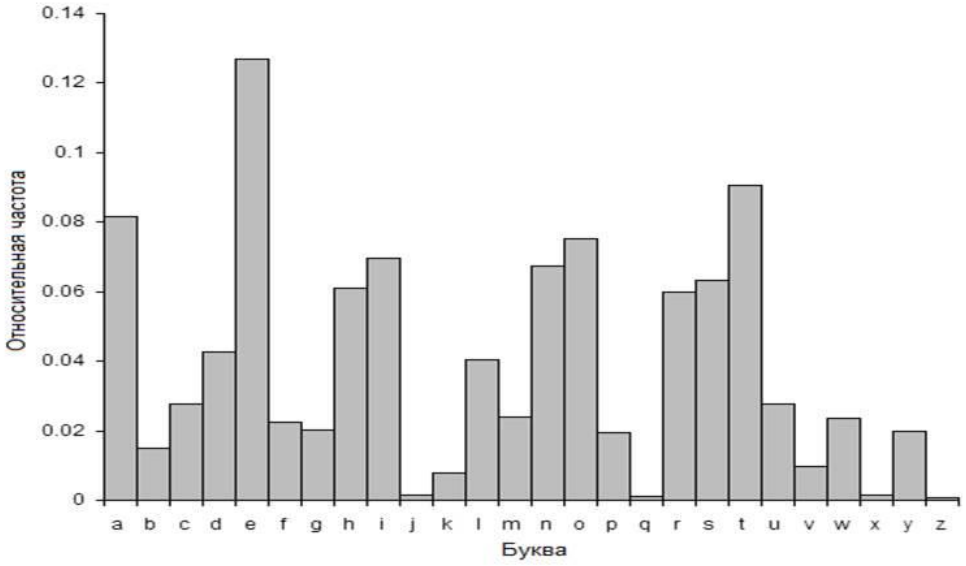
Симетричні криптосистеми – спосіб шифрування, в якому для шифрування і дешифрування застосовується один і той же криптографічний ключ.

Постановка задачі розроблення програмного комплексу

- **В програмі шифрування передбачити:**
- віконний інтерфейс;
- вибір методу шифрування;
- реалізацію шифрування приватним ключем;
- реалізацію блочного шифрування;
- можливість роботи з 64-розрядними блоками даних і ключами трьох розмірів (128, 192 і 256 розрядів).
- роботу програми у консольном додатку.

- **Програма має забезпечувати:**
- – інтуїтивно зрозумілий інтерфейс;
- – функціональну повноту та гнучкість, що передбачає додавання інших методів криптографічного перетворення:
- – уніфікованість у використанні.
- Додатково необхідно:
- використовувати операції, які легко реалізуються програмно;
- орієнтуватися на 32/64 -розрядні процесори;
- не ускладнювати без необхідності структуру шифру.

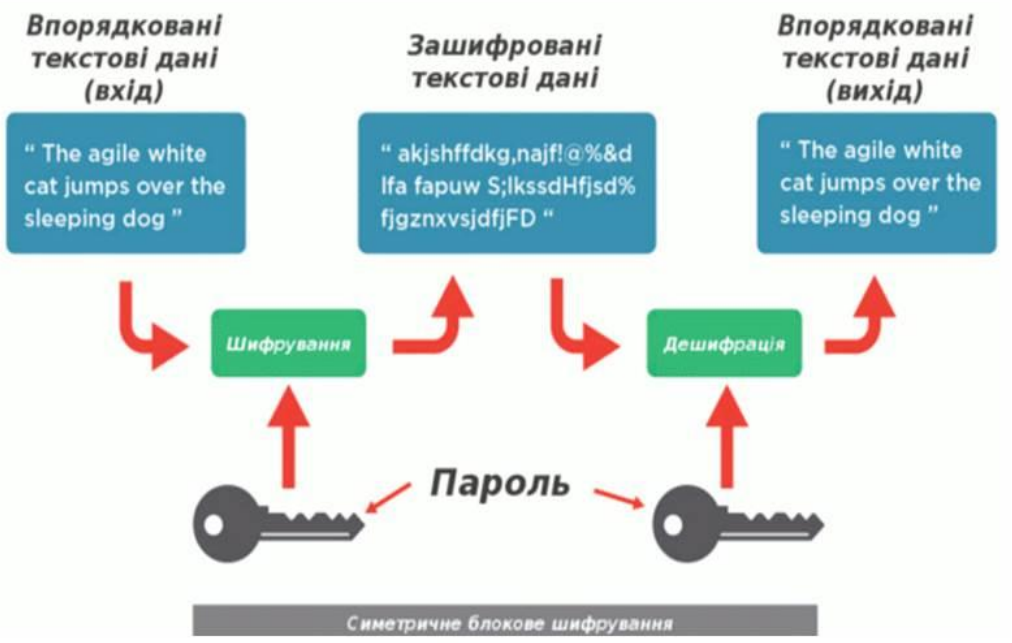
Діаграма частот появи букв в англійському алфавіті у тексті



Організаційна побудова програмних засобів захисту інформації

Елемент	Опис
Алгоритми шифрування	Набір класів, що застосовується для симметричного та асиметричного шифрування, а також хешування
Допоміжні класи	Класи, що забезпечують генерацію випадкових чисел, виконання перетворень, взаємодію із бібліотеками CryptoAPI і саме шифрування на основі потокової моделі
Сертифікати X.509	Класи, визначені в просторі імен System.Security.Cryptography.X509 Certificates, які представляють цифрові сертифікати
Цифрові підписи XML	Класи, що визначають в просторі імен System.Cryptography.Xml і представляють цифрові підписи в XML-документах

Узагальнена структура алгоритму симетричного кодування інформації



Особливості програмної реалізації основних операцій блокового шифрування

- Програмна реалізація БСШ на основі арифметичних операцій за модулем полягає у реалізації таких операцій:
- додавання n-розрядних цілих чисел за модулем 2;
- перестановки блоків по k розрядів n-розрядних цілого числа при $k = 32$, $k = 8$ або $k = 1$;
- множення за модулем 2^n n-розрядних цілих чисел;
- знаходження числа, оберненого за модулем 2^n.

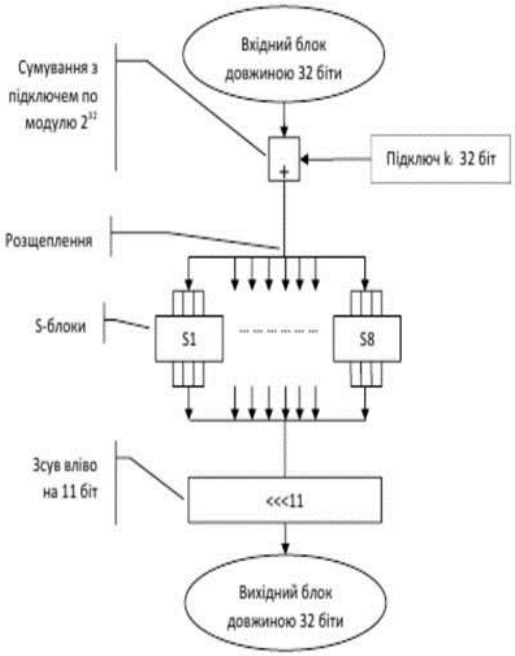
Аналіз продуктивності програмної реалізації блокового симетричного шифру в залежності від розміру блоку

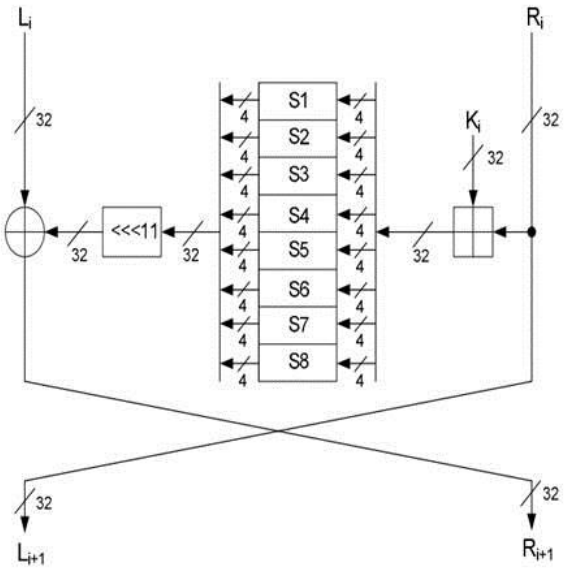
Операція	Кількість тактів				
	Intel Pentium i5	Intel Core i3	Intel Celeron	AMD Athlon 64	AMD Fenom 64
Зашифрування/розшифрування к=32	211	230	277	190	172
Зашифрування/розшифрування к=8	215	250	279	192	173
Зашифрування/розшифрування к=1	282	300	481	323	316
Розгортання ключа для зашифрування	850	1010	1433	989	952
Розгортання ключа для розшифрування	1504	1686	2287	1562	1448

Кількість тактів, необхідна для виконання операцій на процесорах Intel для різних блокових симетричних шифрів

Операція	Кількість тактів					
	БСШ на основі операції за модулем 2^n	MARS	RC6	Rijndael	Serpent	Twofish
Зашифрування	330	669	330	826	1281	800
Розшифрування	330	583	320	796	1122	628
Розгортання ключа для зашифрування	1038	4937	2283	1292	6947	9226
Розгортання ключа для розшифрування	1686	4938	2284	1722	6935	9249

Операції алгоритму по ДСТУ ГОСТ 28147-89:2009

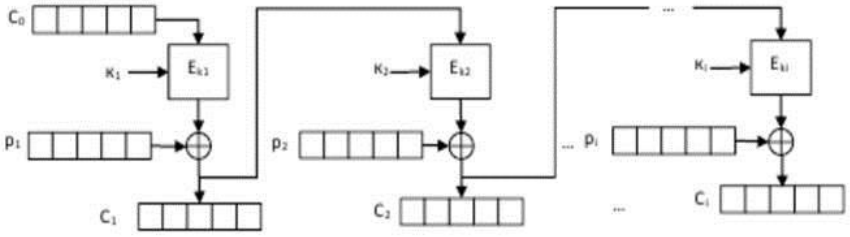




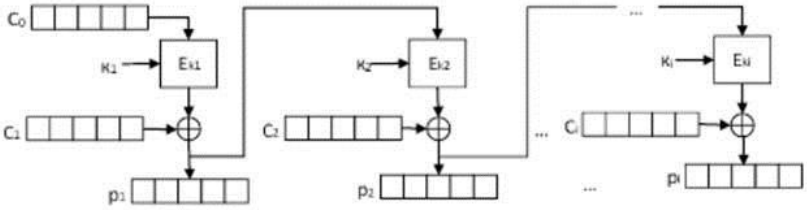
Програмна реалізація гамування зі зворотним зв'язком

Даний режим носить назву Cipher Feedback Mode (CFB) – режим з обереженим зв'язком по шифротексту

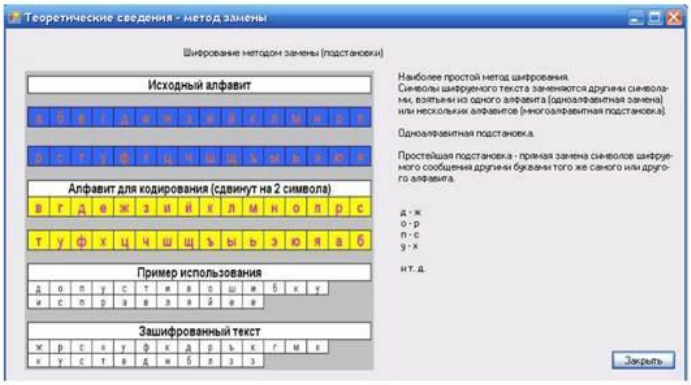
зашифрування



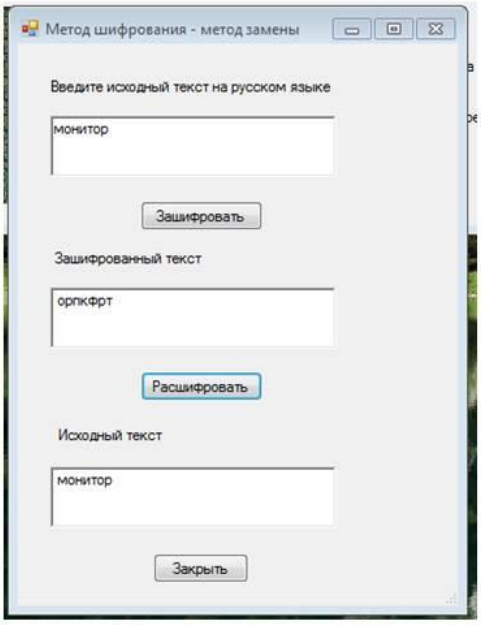
розшифрування



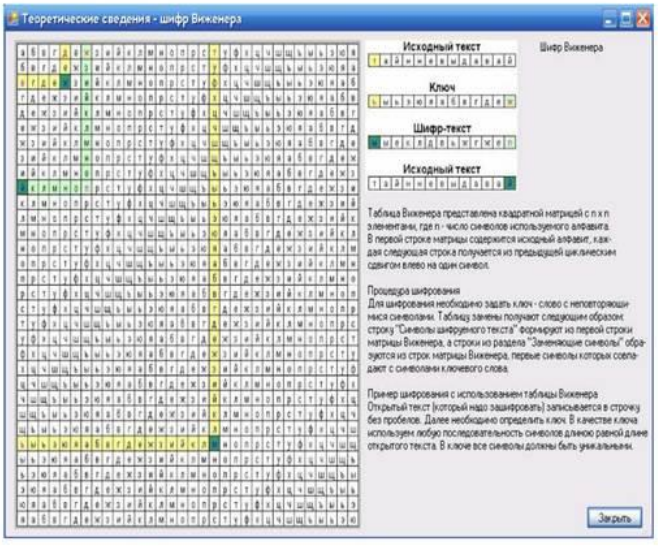
Загальний вигляд програми шифрування

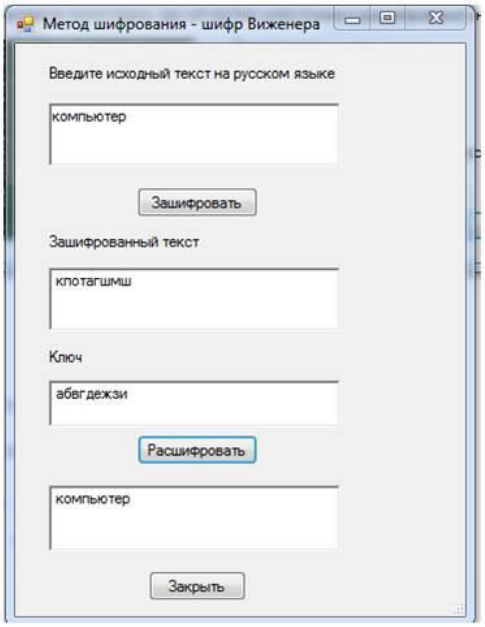


Результат виконання програми методом заміни



Скріншот теоретичних відомостей програми шифру Віжинера





Приклад виконання програми у консольному додатку

шифрування

дешифрування

Висновки

- Розроблено програмний комплекс кодування інформації з використанням методів:
- простої заміни;
- логічної операції додавання по модулю 2;
- шифру Цезаря;
- шифру Віженера;
- гамування зі зворотним зв’язком.
- Реалізація проекту здійснена в інтегральному середовищі Visual Studio 2019 з використанням мови програмування C++, об’єм програми 709 кБ.
- Розроблені заходи щодо охорони праці.

- Результати даного проекту можуть бути використані у навчальному процесі кафедри комп’ютерних наук та інженерії при вивчені дисципліни «Захист інформації в комп’ютерних системах».