Р. С. Белзецький, А. А. Яровий, Я. В. Іванчук

ТЕХНОЛОГІЇ ЗАХИСТУ ІНФОРМАЦІЇ

ЛАБОРАТОРНИЙ ПРАКТИКУМ

Міністерство освіти і науки України
Вінницький національний технічний університет

ТЕХНОЛОГІЇ ЗАХИСТУ ІНФОРМАЦІЇ

ЛАБОРАТОРНИЙ ПРАКТИКУМ

Вінниця
ВНТУ
2022

УДК 004.056.5 (075.8)
Б 43

Рекомендовано до видання Вченою радою Вінницького національного технічного університету Міністерства освіти і науки України (протокол № 9 від 28.04.2022 р.)

Рецензенти:
І. Г. Цмоць, доктор технічних наук, професор
Т. Б. Мартинюк, доктор технічних наук, професор
О. П. Войтович, кандидат технічних наук, доцент

Белзецький, Р. С.

Б 43 Технології захисту інформації : Лабораторний практикум [Електронний ресурс] / Р. С. Белзецький, А. А. Яровий, Я. В. Іванчук – Вінниця: ВНТУ, 2022. – 118 с.

В лабораторному практикумі з дисципліни «Технологія захисту інформації» наведено теоретичні відомості та запропоновані практичні рекомендації з формування компетентностей щодо реалізації алгоритмів кодування сигналів та реалізації крипто алгоритмів захисту інформації, а також інформацію щодо створення програмних засобів в середовищі програмування Arduino IDE.

Видання розроблено відповідно до плану кафедри комп'ютерних наук та програми дисципліни «Технологія захисту інформації» для виконання лабораторних робіт студентами за спеціальністю «Комп'ютерні науки» (освітній ступінь – бакалавр). Лабораторний практикум може бути використаний при підготовці студентів інших спеціальностей.

УДК 004.056.5 (075.8)

ЗМІСТ

ВСТУП

Сучасний розвиток світового інформаційного простору характеризується все більшою залежністю ринку від значного обсягу інформаційних потоків. Тому постає питання про збереження інформації в цілісності та захисті її від несанкціонованого доступу Незважаючи на всі зростаючі зусилля для створення технологій захисту даних, їх вразливість не тільки не зменшується, але й постійно зростає. Тому актуальність проблем, пов'язаних із захистом потоків даних і забезпеченням інформаційної безпеки їх обробки й передачі, усе більше зростає. Проблема захисту інформації є багатоплановою і комплексною й охоплює ряд важливих завдань. Інтенсивний розвиток сучасних інформаційних технологій, і особливо мережних технологій, для цього створює всі передумови. Сюди необхідно також віднести бурхливий розвиток технічних і програмних засобів забезпечення інформаційних технологій, обумовлене прогресом, що відбулись в галузі мікроелектронної технології, і появою нових мультипроцесорних систем обробки даних. У результаті розширилися функціональні можливості й підвищився «інтелект» засобів обробки й передачі даних, а також технічних засобів, застосовуваних для захисту інформації. При вирішенні перерахованих вище завдань обов'язковим є комплексний підхід, що потребує необхідного сполучення застосовуваних законодавчих, організаційних і програмно-технічних засобів.

Навчальний посібник. Лабораторний практикум з дисципліни «Технології захисту інформації» призначений для студентів II курсу бакалаврату спеціальності 122 – «Комп’ютерні науки», але може стати у пригоді студентами інших спеціальностей.

Процес виконання кожної лабораторної роботи складається з розробки алгоритму розв’язання поставленої задачі; складання структурних блок-схем алгоритмів; їх програмної реалізації; тестування та налагодження розробленої програми та складання інструкції щодо її експлуатації; аналізу результатів; підготовки та оформлення звіту.

Звіт виконується на папері формату А4 і повинен мати наскрізну нумерацію (титульний лист входить до наскрізної нумерації, але не нумерується). Структура звіту:

1) титульний аркуш (обов’язково містить номер лабораторної роботи, тему, прізвище та групу виконавця, прізвище та посаду викладача);
2) мета роботи;
3) індивідуальне завдання;
4) алгоритм роботи програми;
5) текст (лістинг) програми;
6) результати виконання поставленого завдання
7) висновки за підсумками виконання лабораторної роботи.

Базове програмне забезпечення – MS Office (Word, Excel), Arduino IDE, Мова програмування – C/C++.

ТЕМАТИКА ЛЕКЦІЙНИХ ЗАНЯТЬ

Розділ 1. Основні види і джерела атак на інформацію

1.1 Сучасна ситуація в області інформаційної безпеки

1.2 Категорії інформаційної безпеки

1.3 Абстрактні моделі захисту інформації

1.4 Огляд найбільш поширених методів "злому"

1.4.1 Комплексний пошук можливих методів доступу

1.4.2 Термінали захищеної інформаційної системи

1.4.3 Отримання пароля на основі помилок адміністратора і користувачів

1.4.4 Отримання пароля на основі помилок в реалізації

1.4.5 Соціальна психологія і інші способи отримання паролів

Розділ 2. Поняття інформаційних загроз та їх види

2.1 Інформаційні загрози

2.2 Інформаційні загрози можуть бути обумовлені:

2.2 Витік конфіденційної інформації

2.3 Канали витоку конфіденційної інформації

2.4 Несанкціонований доступ (НСД)

2.5 Шляхи несанкціонованого доступу

2.6 Поширені технічні загрози і причини

2.7. Способи впливу загроз на інформацію

2.8 Взаємодія автоматизованої інформаційної системи

2.9. Класифікація загроз інформації безпеки

Розділ 3. Шкідливі програми

3.1 Середовища існування

3.2 Способи зараження

3.3 Особливості алгоритму

3.4 Антивірусні програми

3.5. Найпоширеніші види шкідливих програм

Розділ 4. Підходи, принципи, методи і засоби забезпечення безпеки інформаційних систем

4.1 Два принципових підходи до забезпечення інформаційної безпеки

4.2 Принципи забезпечення безпеки інформації

4.3 Комплексний підхід до побудови системи

4.4 Методи і засоби забезпечення безпеки

Розділ 5. Потокові крипто алгоритми.

5.1 Сучасні потокові шифри, переваги та недоліки

5.2 Регістри зсуву зі зворотним зв’язком

Теми практичних занять

Тема 1. Принципи кодування інформації для захисту під час передавання.

Тема 2. Дослідження традиційних крипто алгоритми.

Тема 3. Дослідження принципу роботи генераторів псевдо випадкових величин.

Тема 4. Дослідження принципу роботи алгоритмів потокового шифрування інформації.

Тема 5. Дослідження принципу роботи алгоритмів блочного шифрування інформації.

Тема 6. Принцип формування пари ключів для асиметричних крипто алгоритмів.

Тема 7. Математичні моделі систем та процесів захисту інформації.

Тема 8. Модель реалізації загроз інформаційній безпеці.

Лабораторна робота № 1

Знайомство та початок роботи із відкритою платформою Arduino. Дослідження основних принципів кодування сигналів.

Мета роботи: здійснити передачу бінарного сигналу в лінію зв’язку за допомогою апаратно-програмної платформи Arduino та реалізувати синхронізацію даного сигналу.

1. Основні теоретичні відомості

Arduino — вільна та відкрита апаратно-програмна платформа для створення електронних прототипів. Ця платформа основана на гнучкому, готовому для використання апаратному та програмному забезпеченні. Arduino, завдяки сенсорам, здатна сприймати сигнали із оточення та впливати на оточення, таке як: контроль освітлення, двигуни тощо. Мікроконтролер на платі програмується мовою програмування Arduino, яка основана на мові Wiring [1], та середовище розробки Arduino основано на мові Processing. Проекти Arduino можуть бути як у вигляді окремих одиниць, так і пов'язаних із ПЗ, які виконуються на ПК.

Плата може бути вільно зроблена користувачем, або замовлена у готовому для використання вигляді, програмне забезпечення може бути вільно завантажене із Мережі. Опис апаратної частини (для CAD) розповсюджується із вільною ліцензією і може без обмежень модифіковуватись для власних потреб.

1.1 Інтегроване середовище розробки - Arduino IDE.

IDE (від англ. Integrated Development Environment - інтегроване середовище розробки) - це додаток або група додатків (середовище), призначене для створення, налаштування, тестування і обслуговування програмного забезпечення.

Інтегроване середовище розробки характеризується наявністю складної функціональності, включаючи редагування і компіляція вихідного коду, створення програмних ресурсів, створення баз даних і т.д.

В рамках проекту Arduino було створено програмне забезпечення, що відповідає основним вимогам типового середовища IDE [2]. Це не потужне програмне забезпечення, як наприклад Eclipse або NetBeans, а проста, функціональна програма, яка дозволяє нам писати, компілювати і завантажувати програму в мікроконтролер.

Рисунок 1.1 – Середовище розробки Arduino IDE

Проста структура Arduino IDE є перевагою, так як забезпечує швидке освоєння програми і перехід до розробки додатків для Arduino. Незважаючи на свою простоту і інтуїтивно зрозуміле управління, варто звернути увагу на найбільш важливі елементи програми [3].

Після запуску програми ви можете знайти чотири головних функціональних елементи (див. рисунок 1.1):

меню програми;

панель швидкого доступу до найбільш важливих функцій;

редактор (для розміщення коду програми);

панель повідомлень і статусу програми.

Меню програми дозволяє здійснювати управління проектом, наприклад, створення нового проекту, збереження поточного, роздрукувати на принтері вихідний код.

Цікавою особливістю програми є вбудований набір прикладів програм. Це дуже зручно, так як приклади програм можна відразу перевірити, завантаживши їх в мікроконтролер. При необхідності ви можете зберегти приклад і змінити його відповідно до ваших потреб.

Меню «Файл» і «Правка» містять стандартні параметри.

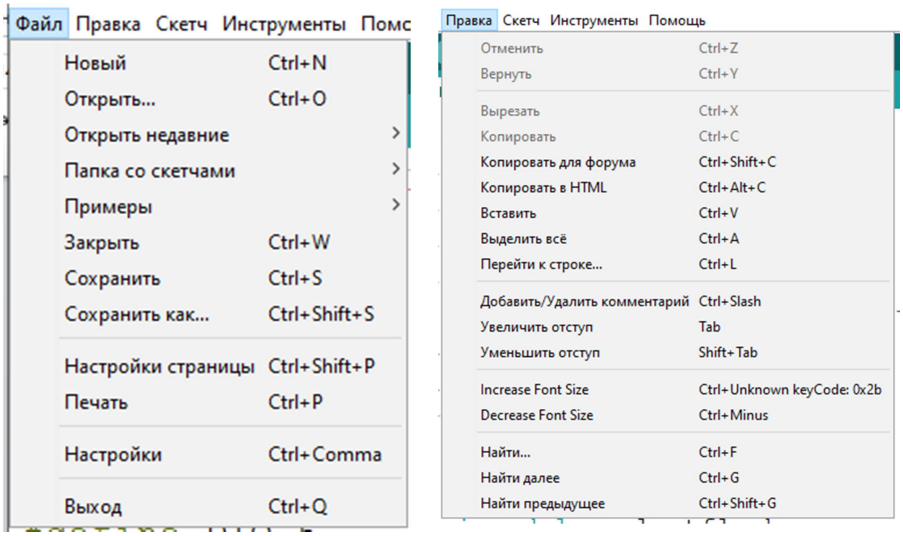


Рисунок 1.2 – Меню «Файл» і «Правка» Arduino IDE

Меню «Скетч» містить параметри для компіляції проекту і імпорту необхідних бібліотек (рис. 1.3).

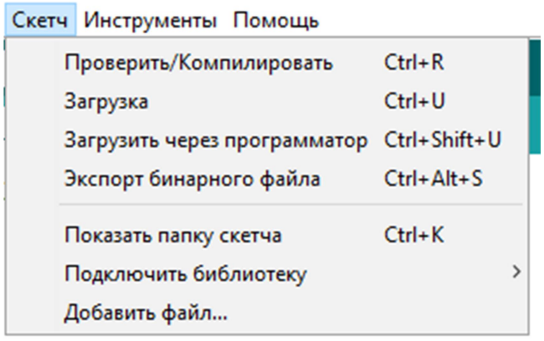


Рисунок 1.3 – Меню «Скетч» Arduino IDE

Цікавим і корисним елементом IDE є меню «Інструменти», яке включає в себе функції автоматичного форматування коду, архівування проекту, включення монітора послідовного порту (USB в Arduino розглядається як звичайний послідовний порт) (рис. 1.4).

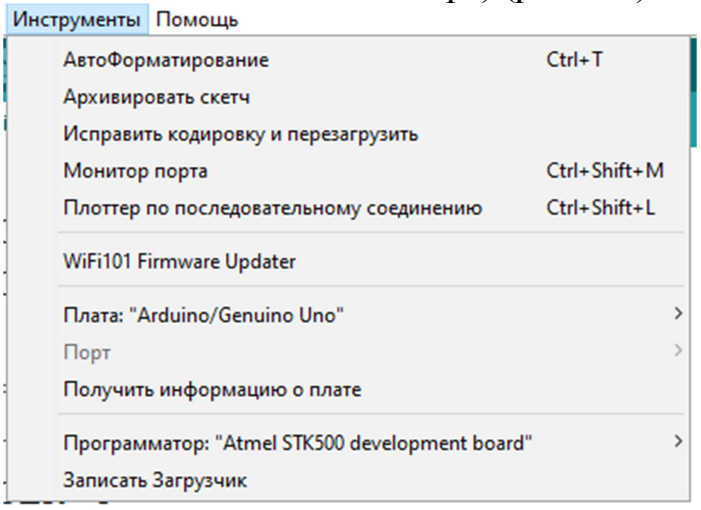


Рисунок 1.4 – Меню «Інструменти» Arduino IDE

Найбільш важливим елементом меню «Інструменти» є можливість вибору відповідної плати, тобто вашої системи Arduino підключеної до комп'ютера. У списку знаходяться всі офіційні версії Arduino. Якщо ваш тип плати відсутній в списку, то ви можемо додати його, змінивши один з файлів програми.

У меню «Інструменти» ви також можете встановити порт, до якого підключена плата Arduino. Пакет Arduino IDE сам визначає порт, але іноді потрібно вручну встановити номер порту в налаштуваннях.

За допомогою Arduino IDE можна також завантажити, тобто запрограмувати Bootloader (завантажувач) для нового, чистого мікроконтролера Atmega, що дозволяє клонувати чіпи або просто замінити несправний мікроконтролер в Arduino.

Для зручної роботи з Arduino IDE використовується панель швидкого доступу (рис. 1.5), яка оснащена найбільш важливими кнопками. Це рішення, що полегшує роботу з пакетом IDE, дає нам прямий доступ до практично всім необхідним параметрам при написанні і тестуванні програми.

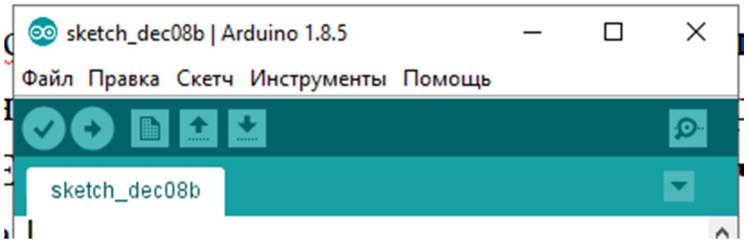


Рисунок 1.5 – Панель швидкого доступу Arduino IDE

Вони дозволяють:

- скомпілювати програму;
- завантажити програму в мікроконтролер (перед прошивкою код програми компілюється);
- почати роботу над новим проектом;
- відкрити існуючий проект;
- зберегти проект на диск;
- включити монітор послідовного порту.

Всі опції, розташовані на панелі швидкого доступу, продубльовані в меню програми.

Додатковим корисним елементом, що знаходяться під кнопкою включення монітора послідовного порту - це меню для управління вкладками. Виберіть в Arduino IDE спрощують написання складних проектів, а так само дозволяють працювати з декількома проектами одночасно.

Найбільша частина вікна програми призначена для написання безпосередньо самого коду програми. Редактор в Arduino IDE не дуже просунутий, але має найважливіші елементи, що дозволяють полегшити написання простих програм. До таких елементів можна віднести

підсвічування синтаксису і блоків (дужки). Це не багато, але достатньо для простих проектів.

Останнім елементом програми є вікно повідомлень і статусу. Інформація, що там виводиться, дозволяє користувачеві знайти помилки в програмному коді і отримати підтвердження про завершення компіляції і завантаження програми в мікроконтролер.

1.2 ArduinoUnoR3. Загальні відомості [4].

ArduinoUno - це пристрій на основі мікроконтролера ATmega328. У його склад входить все необхідне для зручної роботи з мікроконтролером: 14 цифрових входів / виходів (з них 6 можуть використовуватися в якості ШІМ-виходів), 6 аналогових входів, кварцовий резонатор на 16 МГц, роз'єм USB, роз'єм живлення, роз'єм для внутрішньосхемного програмування (ICSP) і кнопка Reset. Для початку роботи з пристроєм достатньо лише подати живлення від AC/DC-адаптера або елемента живлення, або підключити його до комп'ютера за допомогою USB-кабелю див. рисунок 1.6.

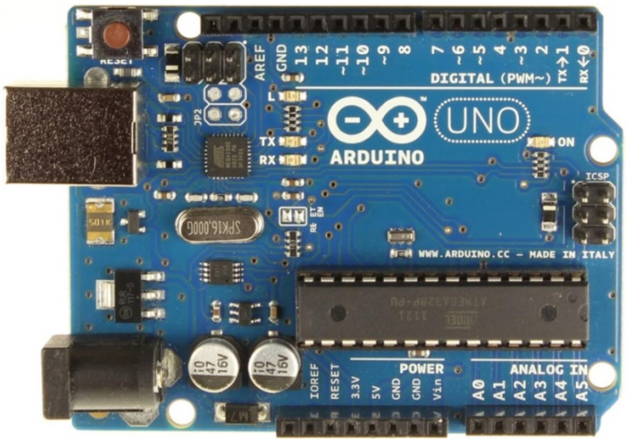


Рисунок 1.6 – Зовнішній вигляд Arduino Uno R3

На відміну від всіх попередніх плат Ардуіно, Uno в якості перетворювача інтерфейсів USB-UART використовує мікроконтролер ATmega16U2 (ATmega8U2 до версії R2) замість мікросхеми FTDI.

Таблиця 1.1 – Характеристики ArduinoUnoR3

мікроконтролер	ATmega328
робоча напруга	5В
Напруга живлення (рекомендована)	7-12В
Напруга живлення (гранична)	6-20В
Цифрові входи / виходи	14 (з них 6 можуть використовуватися в якості ШІМ-виходів)
аналогові входи	6
Максимальний струм одного виводу	40 мА
Максимальний вихідний струм виводу 3.3V	50 мА

Продовження таблиця 1.1 – Характеристики ArduinoUnoR3

Flash-пам'ять	32 КБ (ATmega328) з яких 0.5 КБ використовуються загрузчиком
SRAM	2 КБ (ATmega328)
EEPROM	1 КБ (ATmega328)
Тактова частота	16 МГц

Живлення

ArduinoUno може живиться від USB або від зовнішнього джерела живлення - тип джерела вибирається автоматично.

В якості зовнішнього джерела живлення (не USB) може використовуватися мережевий AC/DC-адаптер або акумулятор/ батарея. Штекер адаптера (діаметр - 2.1мм, центральний контакт - плюс) необхідно вставити у відповідний роз'єм живлення на платі. У разі живлення від акумулятора/батареї, її виводи необхідно під'єднати до виводів Gnd і Vin роз'єму POWER.

Напруга зовнішнього джерела живлення може бути в межах від 6 до 20 В. Однак, зменшення напруги живлення нижче 7В призводить до зменшення напруги на виводі 5V, що може стати причиною нестабільної роботи пристрою. Використання напруги більше 12В може призводити до перегріву стабілізатора напруги і виходу плати з ладу. З огляду на це, рекомендується використовувати джерело живлення з напругою в діапазоні від 7 до 12В.

Нижче перераховані виводи живлення, розташовані на платі:

VIN. Напруга, що надходить в Arduino безпосередньо від зовнішнього джерела живлення (не пов'язане з 5В від USB або іншим стабілізованою напругою). Через цей вивід можна як подавати зовнішнє живлення, так і споживати струм, коли пристрій живиться від зовнішнього адаптера.

5V. На вивід надходить напруга 5В. від стабілізатора напруги на платі, незалежно від того, як живиться пристрій: від адаптера (7 - 12В), від USB (5В) або через вивід VIN (7 - 12В). Живити пристрій через вивід 5V або 3V3 не рекомендується, оскільки в цьому випадку не використовується стабілізатор напруги, що може привести до виходу плати з ладу.

3V3. 3.3В, що надходять від стабілізатора напруги на платі. Максимальний струм, споживаний від цього виводу, становить 50 мА.

GND. Вивід землі.

IOREF. Цей вивід надає платам розширення інформацію про робочу напругу мікроконтролера Ардуіно. Залежно від напруги, знятої з виводу IOREF, плата розширення може переключитися на відповідне джерело живлення або задіяти перетворювачі рівнів, що дозволить їй працювати як з 5В, так і з 3.3В-пристроями.

Пам'ять. Обсяг флеш-пам'яті ATmega328 становить 32 КБ (з яких 0.5 КБ використовуються зантажувачем). Мікроконтролер також має 2 КБ пам'яті SRAM і 1 КБ EEPROM (з якої можна зчитувати або записувати інформацію за допомогою бібліотеки EEPROM).

Входи і виходи. З використанням функцій pinMode (), digitalWrite () і digitalRead () кожен з 14 цифрових вводів може працювати в якості входу або виходу. Рівень напруги на виводах обмежений 5В. Максимальний струм, який може віддавати або споживати один вивід, становить 40 мА. Всі виводи пов'язані з внутрішніми підтягуються резисторами (за замовчуванням відключеними) номіналом 20-50 кОм. Крім цього, деякі виводи Ардуіно можуть виконувати додаткові функції:

Послідовний інтерфейс: вивід 0 (RX) і 1 (TX). Використовуються для отримання (RX) і передачі (TX) даних по послідовному інтерфейсу. Ці виводи з'єднані з відповідними виводами мікросхеми ATmega8U2, яка виконує роль перетворювача USB-UART.

Зовнішні переривання: виводи 2 і 3. Можуть служити джерелами переривань, що виникають при фронті, спаді або при низькому рівні сигналу на цих виводах. Для отримання додаткової інформації див. функцію attachInterrupt ().

ШІМ: виводи 3, 5, 6, 9, 10 і 11. За допомогою функції analogWrite () можуть виводити 8-бітові аналогові значення в вигляді ШІМ-сигналу.

Інтерфейс SPI: виводи 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Із застосуванням бібліотеки SPI дані виводи можуть здійснювати зв'язок по інтерфейсу SPI.

Світлодіод: **13**. Вбудований світлодіод, приєднаний до піна 13. При відправці значення HIGH світло діод вмикається, при відправці LOW - вимикається.

В ArduinoUno є **6 аналогових портів** (A0 - A5), кожен з яких може представляти аналогову напругу у вигляді 10-бітного числа (1024 різних значення). За замовчуванням, вимір напруги здійснюється в діапазоні від 0 до 5 В. Проте, верхню межу цього діапазону можна змінити, використовуючи вивід AREF і функцію analogReference (). Крім цього, деякі з аналогових входів мають додаткові функції:

TWI: вивід A4 або SDA і вивід A5 або SCL. З використанням бібліотеки Wire дані виводи можуть здійснювати зв'язок по інтерфейсу TWI.

Крім перерахованих на платі існує ще кілька виводів:

AREF. Опорна напруга для аналогових входів. Може бути задіяні функцією analogReference ().

Reset. Формування низького рівня (LOW) на цьому виводі призведе до перезавантаження мікроконтролера. Зазвичай цей вивід служить для функціонування кнопки скидання на платах розширення.

Зв'язок. Arduino Uno надає ряд можливостей для здійснення зв'язку з комп'ютером, ще одним Ардуіно або іншими микроконтролерами. У ATmega 328 є приймач UART, що дозволяє здійснювати послідовний зв'язок за допомогою цифрових виводів 0 (RX) і 1 (TX). Мікроконтролер ATmega16U2 на платі забезпечує зв'язок цього приймача з USB-портом комп'ютера, і при підключенні до ПК дозволяє Ардуіно визначатися як віртуальний COM-порт. Прошивка мікросхеми 16U2 використовує стандартні драйвера USB-COM, тому установка зовнішніх драйверів не потрібно. На платформі Windows необхідний тільки відповідний .inf-файл. У пакет програмного забезпечення Ардуіно входить спеціальна програма, що дозволяє зчитувати і відправляти на Ардуіно прості текстові дані. При передачі даних через мікросхему-перетворювач USB-UART під час USB-з'єднання з комп'ютером, на платі будуть мигати світлодіоди RX і TX. (При послідовній передачі даних за допомогою виводів 0 і 1, без використання USB-перетворювача).

Бібліотека SoftwareSerial дозволяє реалізувати послідовний зв'язок на будь-яких цифрових піном Arduino Uno.

У мікроконтролері ATmega328 також реалізована підтримка послідовних інтерфейсів I2C (TWI) і SPI. У програмне забезпечення Ардуіно входить бібліотека Wire, що дозволяє спростити роботу з шиною I2C . Для роботи з інтерфейсом SPI використовуйте бібліотеку SPI .

Програмування

ArduinoUno програмується за допомогою програмного забезпечення Ардуіно. Для цього з меню Tools>Board необхідно вибрати "ArduinoUno" з мікро контролером, відповідним вашій платі.

ATmega328 в ArduinoUno випускається з прошитим загрузчиком, що дозволяє завантажувати в мікроконтролер нові програми без необхідності використання зовнішнього програматора. Взаємодія з ним здійснюється за оригінальним протоколом STK500.

Проте, мікроконтролер можна прошити і через роз'єм для внутрисхемного програмування ICSP (In-Circuit SerialProgramming), не звертаючи уваги на завантажувач.

Вихідний код прошивки мікроконтролера ATmega16U2 (або 8U2 на платах версії R1 і R2) знаходиться у вільному доступі. Прошивка ATmega16U2/8U2 включає в себе DFU-завантажувач (Device Firmware Update), що дозволяє оновлювати прошивку мікроконтролера. Для активації режиму DFU необхідно: після переходу в DFU-режим для завантаження нової прошивки можна використовувати програмне

забезпечення Atmel's FLIP (для Windows) або DFUprogrammer (для Mac OS X і Linux). Альтернативний варіант - прошити мікроконтролера через роз'єм для внутрішньосхемного програмування ISP за допомогою зовнішнього програматора, проте в цьому випадку DFU-завантажувач затреться.

Захист USB від перевантажень. В ArduinoUno є відновлювані запобіжники, що захищають USB-порт комп'ютера від коротких замикань і перевантажень. Незважаючи на те, що більшість комп'ютерів мають власний захист, такі запобіжники забезпечують додатковий рівень захисту. Якщо від USB-порту споживається струм більше 500 мА, запобіжник автоматично роз'єднає з'єднання до усунення причин короткого замикання або перевантаження.

Фізичні характеристики. Максимальна довжина і ширина друкованої плати Uno становить 6.9 см і 5.4 см відповідно, з урахуванням роз'єму USB і роз'єму живлення, які виступають за межі плати. Чотири монтажних отвори дозволяють прикріплювати плату до поверхні або корпусу. Зверніть увагу, що відстань між цифровими виводами 7 і 8 не кратне традиційним 2.54 мм і становить 4 мм.

1.3 Порядок виконання роботи

1. Ознайомимося із компоновкою плати “Arduino Uno R3”.

2. Завантажити інтегроване середовище розробки Arduino– IDE пройшовши за посиланням: посилання наArduino IDE [2] та встановити на ПК.

4. Підключити плату “ArduinoUno R3” до ПК через USB-порт.

5. Запустіть середовище Arduino IDE.

6. В середовищі програмування Arduino відкрийте заготовку, а саме: у меню програми Файл>приклади>Basics>Blink. Відкриється заготовка програми, яка змушує світлодіод на цифровому контакті 13 циклічно умикатися на 1 секунду та вимикатися на 1 секунду.

7. Вибираємо послідовний порт, через який ми будемо працювати із платою “ArduinoUnoR3”. Для цього визначаємо USB порт на який підключена плата у меню програми Інструпент>порт.

8. Завантажити програму у плату, для цього натискаємо кнопку “Upload” у середовищі розробки. Через декілька секунд світлодіоди RX та TX почнуть блимати, вказуючи про обмін інформацією із платою.

Якщо вивантаження пройшло успішно, то у рядку стану буде виведено надпис “Doneuploading.”. Через декілька секунд почне миготіти вбудований на платі світлодіод. Це і є успішне завантаження та виконання програми.

9. Ввести та проаналізувати код наведений у додатку А.

10. Перевести заданий варіант лабораторної роботи у двійковий код, використовуючи ASCII таблицю, або програмні засоби. Запрограмувати виведення отриманого двійкового числа. Виведення здійснити на цифровий порт у відповідності із завданням.

11. Доповнити програму операторами, які будуть показувати візуальні сигнали синхронізації, наприклад, через 1 сек., 5 сек. або будь-яке інше значення. Довжину сигналів синхронізації обрати самостійно, виходячи із інерційності людського зору.

12. Реалізувати програмний код з використанням операторів «delay» та «millis».

1.4 Зміст звіту

1. Звіт з лабораторної роботи повинен бути виконаний на листах формату А4.
2. Звіт повинен містити: назву лабораторної роботи, її мету і короткі теоретичні відомості.
3. В розділі «Результати виконання лабораторної роботи» студент повинен представити процедуру отримання бінарної послідовності, алгоритм реалізації, а також навести код програмної реалізації.
4. Написати висновки до даної лабораторної роботи.
5. Оформити звіт та представити його викладачу на захист.

Контрольні запитання

1. Для чого встановлюються пристрої: “USB SerialConverter” та “USB SerialPort”?
2. Опишіть принцип підключення потужних виконавчих пристрої до порту мікроконтролера.
3. Призначення методу voidsetup() ?
4. Призначення методу voidloop()?
5. Що таке внутрішньо схемне програмування мікроконтролера?
6. Що таке внутрішньо схемне підтягування порту до шини живлення «pull UP»?
7. Наведіть основні характеристики плати Arduino Uno R3?
8. Які дії потрібно виконати, якщо випливає надпис про помилку протокола завантаження програми в макет?
9. Як завантажити готову програму у макет?

Додаток А

```
intledPin = 8;   // Вказівка про під'єднання LED до
цифрового контакту 8.
voidsetup()   // Метод виконується разово, під час
завантаження програми.
{
  pinMode(ledPin, OUTPUT);  // Установити цифрового
портаledPin на вивід.
}
voidloop()   // Метод, який виконується нескінченно
{
  digitalWrite(ledPin, HIGH);   // Установлення LED у стан
HIGH.
  delay(5000);  // Затримка на вказану довжину мілісекунд.
  digitalWrite(ledPin, LOW);  // Установлення LED у стан
LOW.
  delay(1000);  // Затримка на вказану довжину мілісекунд.
}
```

Лабораторна робота № 2

Дослідження процедури біт-стафінга при передачі сигналів в лінії зв’язку.

Мета роботи: дослідити процес додавання додаткових бітів інформації в кард при передачі інформації в каналі зв’язку.

2.1 Основні теоретичні відомості

У дуплексній системі передачі даних (рис. 2.1, а) пристрої передачі інформації А і В одночасно посилають один одному по каналу зв'язку безперервні потоки бітів [5]. Канал зв'язку в даному прикладі виконаний у вигляді витої пари проводів кабелю, в якості пристроїв А і В використані модеми.

Система передачі може бути симетричною або асиметричною. В будь-якому випадку призначені для користувача дані містяться в кадри. Кожен кадр починається з прапора F – заздалегідь заданої комбінації службових бітів, що дорівнює, наприклад, 01111110. Закінчення кадру також може позначатися прапорцем, але тут цей варіант не розглядається. Якщо даних немає, то по каналу зв'язку передаються "порожні" кадри. В якості таких вироджених кадрів може передаватися безперервна послідовність прапорців, що не перекриваються або перекриваються: ...0111111001111110011111110011... або ...0111111011111101111101111... . Далі для визначеності передбачається, що дані передаються з пристрою А (передавача) в пристрій В (приймач).

Прапорцеві біти з точки зору користувача системи не приносять користі і навіть шкідливі, так як для їх передачі необхідно витрачати частину пропускної здатності каналу зв'язку. Тому в деяких системах [6] довжину прапорців зменшують до одного біта і навіть повністю виключають прапорці в тому вигляді, в якому він показаний на рис. 2.1, б. Але в таких системах час початкового встановлення (або відновлення втраченої) кадрової синхронізації між пристроями А і В відносно великий. Для зменшення цього часу довжину прапорця збільшують або роблять його "унікальним".

Як уже зазначалося, унікальність прапорця гарантує відсутність в тілі кадру його випадкових копій. Ці копії можуть спочатку існувати, так як дані, що передаються, як призначені для користувача, так і службові, в загальному випадку довільні. Для усунення цих копій вони навмисно і зворотно спотворюються за допомогою біт-стафінга (рис. 2.1, б).

Рисунок 2.1 – Передача даних між пристроями А і В: а – схема;
б – структура потоку бітів.

Широке поширення набув варіант біт-стафінга, показаний на рис. 2.2.

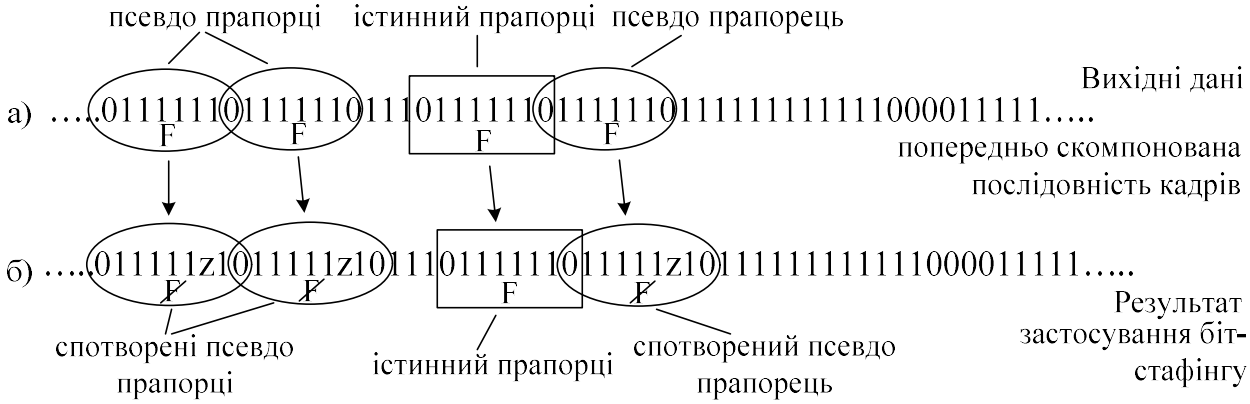


Рисунок 2.2 – Використаня біт-стаффінгу

При формуванні кадру в його початок поміщається прапорець, який вибирається рівним 01111110 і надалі залишається незмінним. Далі кадр або група кадрів аналізується з метою виявлення "особливих" кодів. Особливий код (для стислості - код Q) являє собою комбінацію бітів, частково збігається з прапорцем: Q = 011111. До знайдених кодів Q передавач приписує справа "зайвий" нуль (позначається тут і далі символом z) незалежно від значення подальшого біта. При цьому утворюються послідовності бітів 011111z0 або 011111z1. До теперішнього прапорця z-біт не додають, так як прапорець не піддається біт-стафінг і тому стає унікальним - тепер він однозначно розпізнається на тлі отриманого "прозорого" масиву бітів.

Приймач виявляє коди (01111110) і розцінює їх як ознаки початку кадрів. Далі приймач проводить операцію пошуку і подальшого виключення z-бітів. Виявивши коди Q і наступні за ними z-біти (вони за визначенням є "зайвими"), приймач викреслює їх і тим самим відновлює вихідну послідовність бітів.

У кодовій ситуації, показаній на рис. 2.2, (потік дани а), крім істинного прапорця в початкових даних легко проглядаються його помилкові копії. Передавач навмисно спотворює їх введенням z-бітів рис. 2.2, (потік дани б), тому копії не сприймаються приймачем як прапорці.

Біти z* = 0 можуть вводитися або не вводиться в послідовність одиниць в залежності від прийнятого протоколу обміну. У першому випадку мається на увазі, що в якості нуля в коді Q може бути присутнім як "звичайний" логічний 0, що належить вихідними даними, такі z-біт. У другому випадку приймається домовленість про те, що z-біт не може входити в код Q. Можливі й інші домовленості, передбачені конкретним протоколом обміну. Наприклад, якщо слідом за кодом Q приймач виявляє дві логічні 1, (комбінація 01111111),то він аналізує такі біти, в яких можуть кодуватися будь-які команди або ознаки (наприклад, команда аварійного завершення роботи).

Однак такий спосіб кодування вимагає певної кількості надлишкових бітів. Це пов'язано, по-перше, з тим, що код Q має розрядність, рівну шести і зустрічається в випадкових даних досить часто, по-друге, з'являється необхідність поділу ланцюжків з логічних 1 в призначених для користувача даних бітами z* на п’яти розрядні групи. Якщо вихідні дані в тілі кадру розглядати як випадкові, то ймовірність виявлення шести розрядна коду Q в будь-який випадково обраної групі з шести бітів дорівнює $2^{-6} = 1/64$. Іншими словами, при послідовному перегляді вихідних даних код Q буде виявлятися передавачем в середньому один раз в кожному ланцюжку з 64 бітів.

2.2 Порядок виконання роботи

1. Розглянути основні теоретичні відомостями поняття біт-стафінг.
2. Ввести та проаналізувати код наведений у додатку Б.
3. У відповідності з варіантом лабораторної роботи проаналізувати код наведений в додатку Б та написати такий код, окремо для клієнта, окремо для сервера, який буде виконувати функцію біт-стаффінгу та зворотне перетворення у відповідності з заданим варіантом. Для успішного виконання слід скористатися програмами із попередньої лабораторної роботи.
4. Зробити висновки до даної лабораторної роботи.
5. Оформити звіт та представити його викладачу на захист.

2.3 Зміст звіту

1. Звіт з лабораторної роботи повинен бути виконаний на аркушах формату А4.

2. Звіт повинен містити: назву лабораторної роботи, її мету і короткі теоретичні відомості.

3. В розділі «Результати виконання лабораторної роботи» студент повинен представити процедуру біт-стафінгу та де біт-стафінгу, а також навести код програмної реалізації.

4. Написати висновки до даної лабораторної роботи.

5. Оформити звіт та представити його викладачу на захист.

Таблиця 2.1 - Варіанти завдань

Варіант	прапорець	Довжина кадру, біт	При відсутності даних в лінії зв'язку передається послідовність прапорців
1	10000001	256	що перекриваються
2	01111110	128	що не перекриваються
3	1000001	64	що перекриваються
4	011111110	32	що не перекриваються
5	01111110	256	що перекриваються
6	1000001	128	що не перекриваються
7	011111110	64	що перекриваються
8	0111110	32	що не перекриваються
9	10000001	256	що перекриваються
10	01111110	128	що не перекриваються
11	1000001	64	що перекриваються
12	011111110	32	що не перекриваються
13	01111110	256	що перекриваються
14	1000001	128	що не перекриваються
15	011111110	64	що перекриваються
16	011111110	32	що не перекриваються
17	10000001	256	що перекриваються
18	01111110	128	що не перекриваються
19	1000001	64	що перекриваються
20	0111110	32	що не перекриваються
21	01111110	256	що перекриваються
22	1000001	128	що не перекриваються
23	01111110	64	що перекриваються
24	0111110	32	що не перекриваються
25	10000001	256	що перекриваються
26	01111110	128	що не перекриваються
27	1000001	64	що перекриваються
28	011110	32	що не перекриваються
29	01111110	256	що перекриваються
30	1000001	128	що не перекриваються

Контрольні запитання

1. Поясніть принцип роботи алгоритму біт-стафінга.
2. Яке призначення прапорців при передачі кадру у каналі зв’язку?
3. Що таке «вироджені» кадри?
4. Що таке послідовність прапорців що перекриваються?
5. У якому випадку в каналі зв’язку безперервно передається послідовність прапорців?
6. Що розуміється під словом «кадр»?
7. Поясніть принцип утворення додаткових розрядів в результуючому коді.

Додаток Б

```
// Виконує біт-стафінг відповідно до будь-якого прапорця та вхідної комбінації (універсальна)
// Вхідна комбінація записана в коді (hardcoded)
#define ledPin 9
#define syncPin 8
#define period 500

#define LEN(x) (sizeof(x) / sizeof((x)[0]))

int variant[] = {1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1};
int flag[] = {0, 1, 1, 1, 1, 0};

const int dataSize = LEN(variant) + (LEN(variant) + LEN(flag) - 2) / (LEN(flag) - 1) + LEN(flag) * 2;
int data[dataSize] = {};

int iterator = 0;
int counter = 0;
int previous = 0;
int sync;

void setup()
{
  pinMode(ledPin, OUTPUT);
  pinMode(syncPin, OUTPUT);
  generateData();
}

void bitStuff()
{
  int flagIterator = 0;
  for (int i = 0; i < LEN(variant); i++, counter++)
```

```
  {
    if (variant[i] == flag[flagIterator])
    {
      flagIterator++;
    }
    else
    {
      flagIterator = (variant[i] == flag[0]) ? 1 : 0;
    }
    if (flagIterator > LEN(flag) - 2)
    {
      data[counter] = !flag[LEN(flag) - 1];
      counter++;
      flagIterator = (variant[i] == flag[0]) ? 1 : 0;
    }
    data[counter] = variant[i];
  }
}

void addFlag()
{
  for (int i = 0; i < LEN(flag); i++, counter++)
    data[counter] = flag[i];
}

void generateData()
{
  addFlag();
  bitStuff();
  addFlag();
}

void loop()
{
  sync = digitalRead(syncPin);
  if (sync != previous)
    digitalWrite(ledPin, data[iterator]);
  digitalWrite(syncPin, !sync);
  previous = sync;
  delay(period);
  iterator++;
  if (iterator >= counter) exit(0);
}
```

Лабораторна робота № 3
Дослідження основних методів кодування сигналів.

Мета роботи: Дослідити один із методів кодування сигналів та реалізувати у програмному коді за допомогою апаратних засобів.

4.1 Основні теоретичні відомості

При цифровому кодуванні дискретної інформації застосовують потенційні й імпульсні коди.

У потенційних кодах для представлення логічних одиниць і нулів використовується тільки значення потенціалу сигналу, а його перепади, що формують закінчені імпульси, до уваги не приймаються. Імпульсні коди дозволяють представити двійкові дані або імпульсами визначеної полярності, або частиною імпульсу — перепадом потенціалу визначеного напрямку.

Вимоги до методів цифрового кодування

В традиційній термінології формування імпульсних сигналів називають фізичним кодуванням. Те саме значення має термін імпульсно-кодова модуляція. Але ми будемо використовувати саме перше визначення. Далі розглянемо найбільш поширені форми імпульсних сигналів [8].

При використанні прямокутних імпульсів для передачі дискретної інформації необхідно вибрати такий спосіб кодування, що одночасно досягав би декількох цілей:

1. Мав при одній і тій же бітовій швидкості найменшу ширину спектра результуючого сигналу;
2. Забезпечував синхронізацію між передавачем і приймачем;
3. Мав здатність розпізнавати помилки;
4. Мав низьку вартість реалізації.

4.2 Системи з дворівневим кодуванням

Прості дворівневі потенційні сигнали (NRZ)

Найбільш простим і водночас достатньо популярним є так званий сигнал NRZ (not return to zero – без повернення до нуля). Такий сигнал не змінює значення всередині такту і його інформаційним параметром є потенціал [8].

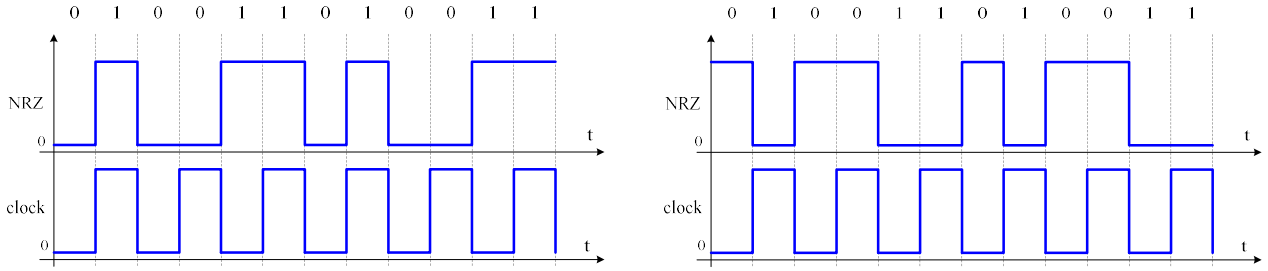
NRZ – потенційний код, стан якого прямо або інверсно відображає значення біта даних. NRZ – дворівневий код, нулю відповідає нижній рівень, одиниці – верхній рівень. Інформаційні переходи відбуваються на межі значущих інтервалів.

NRZ (прямий):

- біти 0 подаються нульовою напругою 0 (В);
- біти 1 подаються значенням U (В).

NRZ (інверсний):

- біти 1 подаються нульовою напругою 0 (В);
- біти 0 подаються значенням U (В).



NRZ (прямий) NRZ (інверсний)

Рисунок 3.1 – Потенційний код NRZ

Переваги. До переваг сигналу NRZ належить простота реалізації та надійність (розпізнаються два альтернативні значення), а також відносно вузький спектр (сигнал заповнює весь такт передачі). Висока швидкість передачі при заданій смузі пропускання (для забезпечення пропускної здатності в 10 Мбіт/сек смуга пропускання складе 5 МГц, так як одне коливання дорівнює 2 бітам).

Недоліки. До недоліків можна віднести можливість між сигнальної інтерференції та можливість порушення синхронізації при передачі довгих однорідних послідовностей 1 або 0 (ці недоліки частково усуваються зокрема за рахунок логічного кодування). Навіть при наявності високоточного тактового генератора приймач може помилитися з моментом зчитування даних, тому що частоти двох генераторів ніколи не бувають цілком ідентичними. Тому при високих швидкостях обміну даними і довгими послідовностями одиниць чи нулів невелика неузгодженість тактових частот може привести до помилки в цілий такт і, відповідно, зчитуванню некоректного значення біту.

Потенційний код з інверсією при одиниці NRZI (1)

При передачі нуля він передає потенціал, що був встановлений у попередньому такті (тобто не змінює його), а при передачі одиниці потенціал інвертується на протилежний. Цей код називається потенційним кодом з інверсією при одиниці.

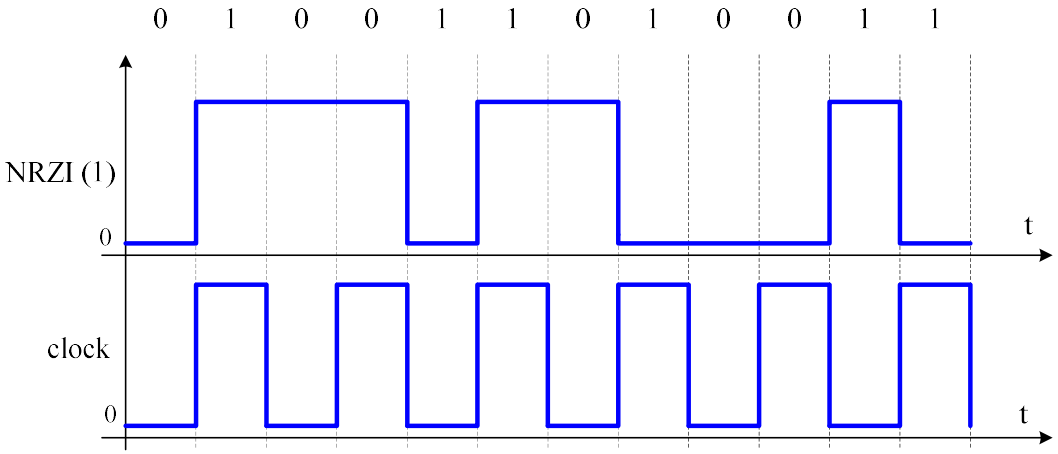


Рисунок 3.2 – Потенційний код NRZI (1)

Переваги методу NRZI:

Простота реалізації.

Метод має гарне розпізнання помилок (завдяки наявності двох потенціалів які різко відрізняються).

Недоліки методу NRZI:

Метод не має властивість самосинхронізації. Навіть при наявності високоточного тактового генератора приймач може помилитися з вибором моменту знімання даних, так як частоти двох генераторів ніколи не бувають повністю ідентичними. Тому при високих швидкостях обміну даними і довгих послідовностях одиниць або нулів невелика неузгодженість тактових частот може призвести до помилки в цілий такт і, відповідно, зчитуванню некоректного значення біта

Другим серйозним недоліком методу, є наявність низькочастотної складової, яка наближається до постійного сигналу при передачі довгих послідовностей одиниць і нулів. Через це багато ліній зв'язку, що не забезпечують прямого гальванічного з'єднання між приймачем і джерелом, цей вид кодування не підтримують. Тому в мережах код NRZ в основному використовується у вигляді різних його модифікацій, в яких усунуті як погана самосинхронізація коду, так і проблеми постійної складової.

Потенційний код з інверсією при нулеві NRZI (0)

При передачі одиниці він передає потенціал, що був встановлений у попередньому такті (тобто не змінює його), а при передачі нуля потенціал інвертується на протилежний. Цей код називається потенційним кодом з інверсією при нулеві.

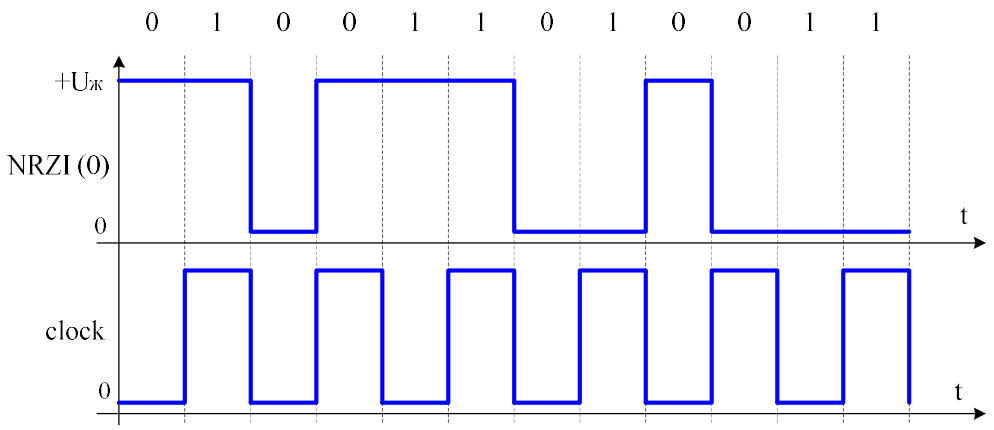


Рисунок 3.3 – Потенційний код NRZI (0)

Манчестерське кодування

У манчестерському коді для кодування одиниць і нулів використовується перепад потенціалу, тобто фронт імпульсу. При манчестерському кодуванні кожен такт поділяється на дві частин. Інформація кодується перепадами потенціалу, що відбуваються в середині кожного такту. Одиниця кодується перепадом від низького рівня сигналу до високого, а нуль — зворотнім перепадом. На початку кожного такту може відбуватися службовий перепад сигналу, якщо потрібно представити кілька одиниць чи нулів підряд. Тому сигнал змінюється принаймні один раз за такт передачі одного біта даних, манчестерський код володіє гарними самосинхронізувальними властивостями [9].

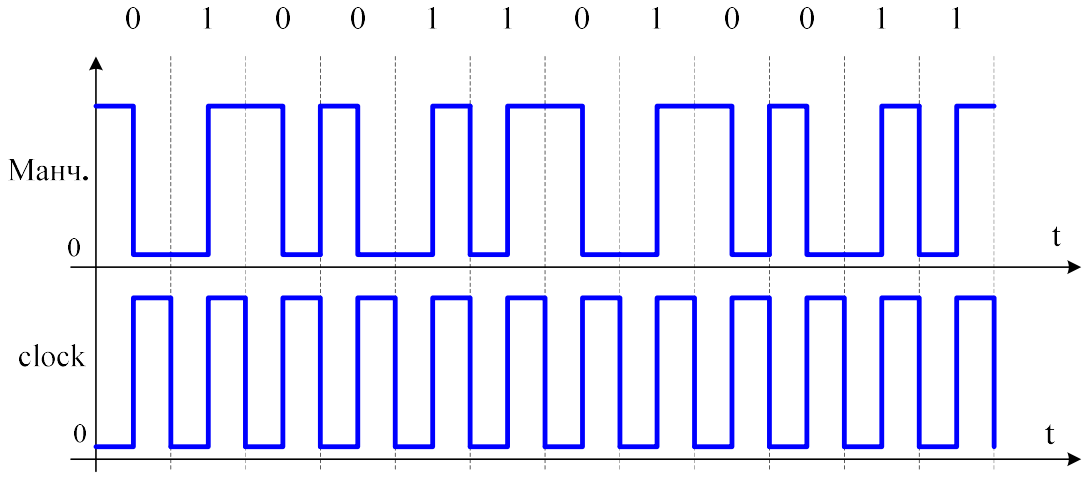


Рисунок 3.4 – Манчестерське кодування

В середньому ширина спектра при манчестерському кодуванні в два рази ширше ніж при NRZ кодуванні.

Метод є самосинхронізувальним, тобто не вимагає спеціального кодування синхроімпульсу, який би займав смугу даних і тому є самим щільним кодом на одиницю частоти.

4.3 Системи з трирівневим кодуванням

Біполярний код AMI

Однією з модифікацій методу NRZ є метод *біполярного кодування з альтернативною інверсією* (*Bipolar Alternate Mark Inversion, AMI*) [10]. У цьому методі використовуються три рівні потенціалу — негативний, нульовий і позитивний. Для кодування логічного нуля використовується нульовий потенціал, а логічна одиниця кодується або позитивним потенціалом, або негативним, при цьому потенціал кожної нової одиниці протилежний потенціалу попередньої.

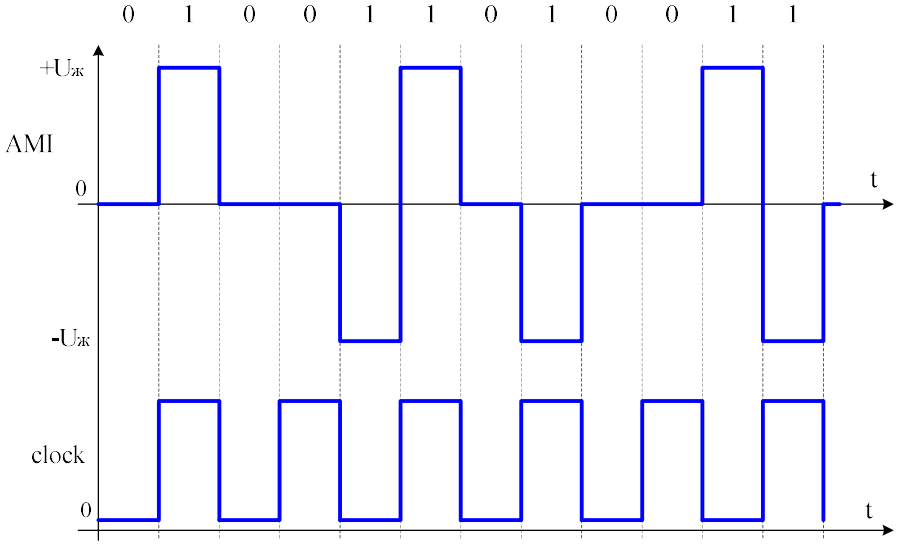


Рисунок 3.5 – Біполярний код AMI

AMI-код використовує такі подання бітів:

- біти 0 подаються нульовою напругою 0 (В);
- біти1 подаються почерговими значеннями -Uж або +Uж (В).

AMI-код володіє хорошими синхронізуючими властивостями при передачі серій одиниць і порівняно простий в реалізації.

Недоліком коду є обмеження на щільність нулів в потоці даних, оскільки довгі послідовності нулів ведуть до втрати синхронізації.

Біполярний імпульсний код

Крім потенційних кодів у мережах використовуються й імпульсні коди, коли дані представлені повним імпульсом чи його частиною — фронтом. Найбільш простим випадком такого підходу є біполярний імпульсний код, у якому одиниця представлена імпульсом однієї полярності, а нуль — іншої. Кожен імпульс триває половину такту. Такий код володіє гарними самосинхронізувальними властивостями, але постійна складова може бути присутня, наприклад, при передачі довгої послідовності одиниць чи нулів. Крім того, спектр у нього ширше, ніж у потенційних кодів. Через занадто широкий спектр біполярний імпульсний код використовується рідко.

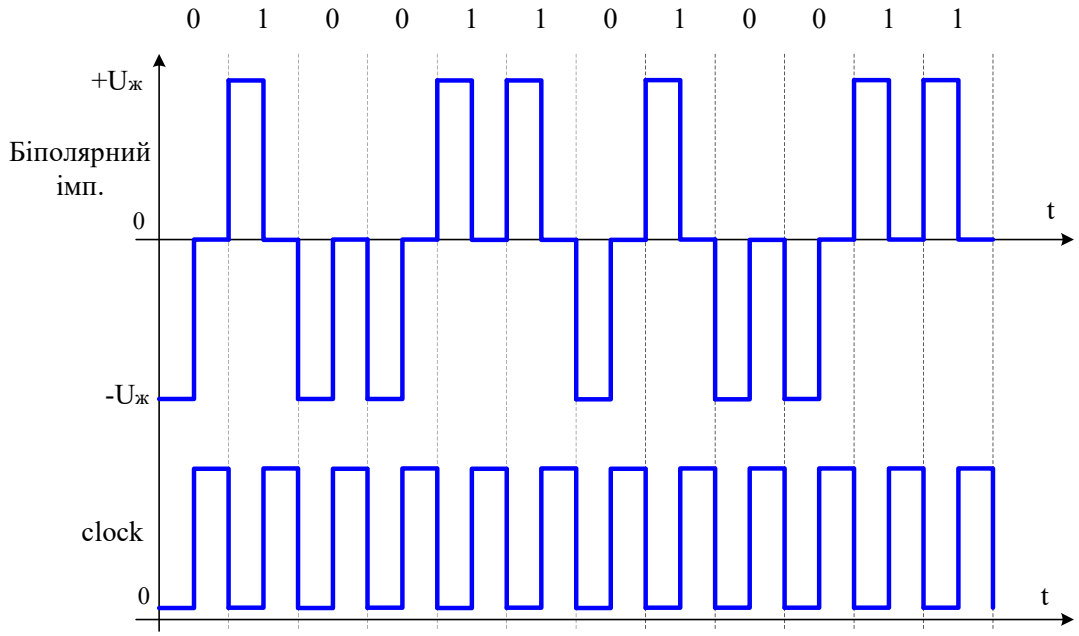


Рисунок 3.6 – Біполярний імпульсний код

MLT-3

Код трирівневої передачі даних MLT-3 (Multi Level Transmission-3) має багато спільного з кодом AMI [11]. Одиниці відповідає послідовний перехід на границі бітового інтервалу з одного рівня сигналу на інший. При передачі нулів сигнал не змінюється (рисунок 3.7). Максимальна частота сигналу досягається при передачі довгою послідовності одиниць. У цьому випадку зміна рівня сигналу відбувається послідовно з одного рівня на інший з урахуванням попереднього переходу.

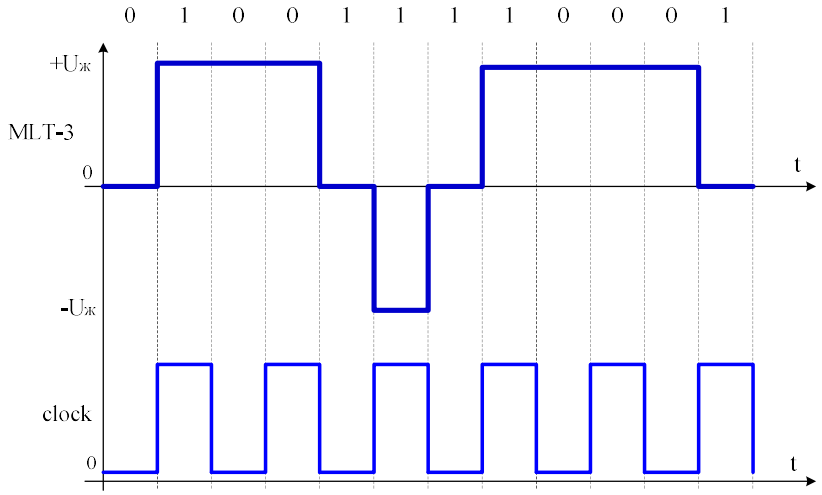


Рисунок 3.7 – MLT-3 кодування сигналу

MLT-3 -код використовує такі подання бітів:

- біти 0 подаються попереднім значенням сигналу (В);
- біти 1 подаються почерговими значеннями +Uж, 0, -Uж. (В).

Для сигналів MLT-3 значення «1» циклічно переключають рівні +Uж → 0→ -Uж→ 0→ +Uж. тоді як «0» відображаються паузами (сигнал залишається не зміннним). Такий спосіб дозволяє додатково звузити спектр. Як і у випадку NRZ для виключення довгих послідовностей нулів, для яких може порушуватись синхронізація, тут використовується попереднє логічне кодування.

Переваги. У порівняння з кодуванням NRZ синхронізацію приймача і передавача можна виконати при послідовній передачі логічних «нулів».

Недоліки. Самосинхронізація гірше в порівнянні з іншими трьохрівневими методами кодування, оскільки синхронізація не провадиться в період часу, коли передається послідовність представлена логічними нулями. Проблему синхронізації доводиться вирішувати за допомогою перетворення даних виключаючи довгі послідовності з логічних «нулів». Реалізація трьох логічних рівнів складніше, ніж реалізація двох рівнів.

4.4 Системи з чотирьох рівневим кодуванням (зі збільшенням бітової швидкості)

2B1Q

Алгоритм 2B1Q (2 Binary 1 Quaternary) [12] є одним з варіантів реалізації алгоритму амплітудно-імпульсної модуляції з чотирма рівнями вихідної напруги без повернення до нульового рівня (NRZ). Для формування лінійного коду вхідний інформаційний потік ділиться на кодові групи по два біта (2B) в кожній. Залежно від комбінації значень бітів кодової групи їй ставиться у відповідність один з чотирьох кодових символів, кожному з яких у свою чергу ставиться у відповідність один з рівнів кодової напруги (чотири стани 1Q). Таким чином, закодований відповідно до правил алгоритму 2B1Q,сигнал є послідовністю стрибкоподібних змін напруги.

Таблиця 3.1 – Таблиця відповідності коду 2B1Q

Кодова група	Кодова напруга
00	–2,5 В
01	–0,83В
10	2,5 В
11	0,83 В

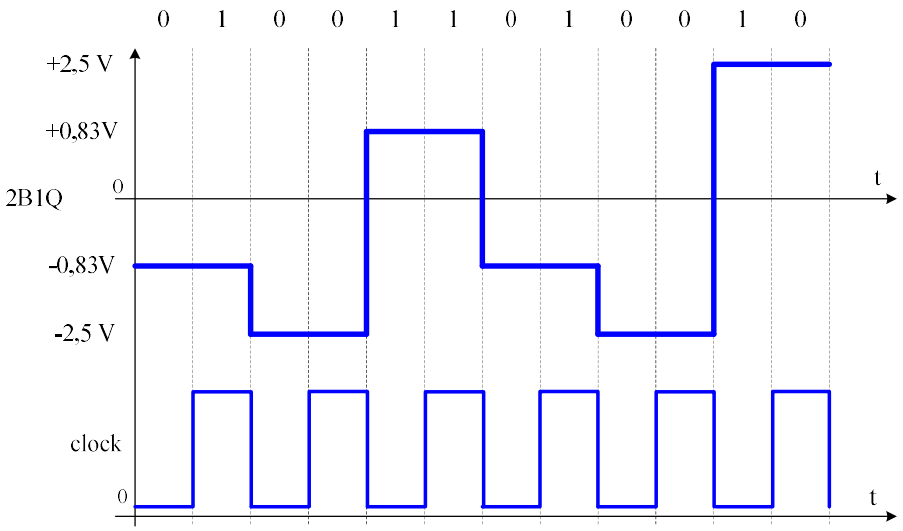


Рисунок 3.8 – Кодування 2B1Q

При цьому способі кодування вимагаються додаткові заходи по боротьбі з довгими послідовностями однакових пар біт, тому що при цьому сигнал перетворюється в постійну складову.

Переваги. При випадковому чергуванні біт спектр сигналу в два рази вужче, ніж у коду NRZ, тому що при тій же бітовій швидкості тривалість такту збільшується в два рази. Таким чином, за допомогою коду 2B1Q можна по одній і тій же лінії передавати дані в два рази швидше, ніж за допомогою коду AMI чи NRZI.

Недоліки. Для реалізації 2B1Q потужність передавача повинна бути вище, щоб чотири рівні чітко розрізнялися приймачем на тлі перешкод.

PAM-5

PAM-5 (Pulse Amplitude Modulation) є певним удосконаленням 2B1Q. Він передбачає додаткове використання нульового рівню сигналу для збільшення надійності передачі під впливом шуму. Отже тут умовні значення сигналу -2, -1, 0, +1, +2.

Таблиця 3.2 – Таблиця відповідності коду PAM-5

Кодова група	Кодова напруга
00	–2,5 В
01	–0,83В
Сигнал відсутній	0 В
11	0,83 В
10	2,5 В

При фізичних значеннях -1,0В / -0,5В / 0,0В / +0,5В / +1,0В). Для збільшення надійності розпізнавання "сусідні" значення сигналів після попередньої обробки передають різними фізичиними лініями.

3.3 Порядок виконання роботи

1. Розглянути основні теоретичні відомості систем кодування інформації.
2. У відповідності з варіантом лабораторної роботи реалізувати алгоритм кодування інформації для клієнта та сервера у відповідності до заданого варіанту.
3. Реалізувати розроблений в пункті 2 алгоритм з урахуванням прикладу реалізації методу кодування інформації (див. додаток В), за допомого інтегрованого середовища розробки Arduino IDE для плати Arduino UNO R3.

3.4 Зміст звіту

1. Звіт з лабораторної роботи повинен бути виконаний на аркушах формату А4.
2. Звіт повинен містити: назву лабораторної роботи, її мету і короткі теоретичні відомості.
3. В розділі «Результати виконання лабораторної роботи» студент повинен представити процедуру кодування інформації у вигляді часових діаграм, у відповідності з варіантом, а також навести код програмної реалізації процесу кодування декодування для клієнта і для сервера.
4. Написати висновки до даної лабораторної роботи.
5. Оформити звіт та представити його викладачу на захист.

Таблиця 3.1 - Варіанти завдань

Варіант	Тип кодування
1	AMI
2	NRZI (0)
3	Біполярний імпульсний
4	Манчестерський
5	2B1Q
6	AMI
7	NRZI (1)
8	Біполярний імпульсний
9	Манчестерський
10	2B1Q
11	AMI
12	NRZI (0)
13	Біполярний імпульсний
14	Манчестерський
15	2B1Q

Варіант	Тип кодування
16	2B1Q
17	NRZI (1)
18	Біполярний імпульсний
19	Манчестерський
20	Манчестерський
21	2B1Q
22	AMI
23	2B1Q
24	NRZI (0)
25	Біполярний імпульсний
26	Манчестерський
27	2B1Q
28	AMI
29	NRZI (0)
30	Манчестерський

Контрольні запитання

1. Наведіть приклади дворівневого кодування інформації?
2. Наведіть приклади трьохрівневого кодування інформації?
3. Наведіть приклади чотирьохрівневого кодування інформації?
4. Які є основні типи алгоритмів кодування інформації?
5. Порівняти швидкість кодування інформації за допомогою 2B1Q та Манчестерського кодування?
6. Що таке цифрове кодування?
7. Пояснити, яким чином цифрове кодування впливає на криптостійкість системи?
8. Який з методів цифрового кодування найшвидший? За рахунок чого?
9. Якщо методи цифрового кодування в назві відрізняються тільки значенням рівня, чи можна назвати ці методи інверсними один відносно одного?
10. Порівняти швидкість кодування інформації за допомогою NRZI (1) та NRZ?

ДОДАТОК В

```
//Виконує кодування вхідного сигналу в двійковій системі кодом AMI
//Вхідна комбінація записана в коді

#define PLUS 8
#define MINUS 9
#define ZERO 13
#define DELAY_TIME 500

intvariant[] = {1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0};
boolpolus = true;
intiterator = 0;


voidsetup()
{
pinMode(PLUS, OUTPUT);
pinMode(MINUS, OUTPUT);
pinMode(ZERO, OUTPUT);
```

```
}

voidloop()
{
intpin = ZERO;
if (variant[iterator] == 1)
    {
    pin = (polus)?PLUS:MINUS;
polus = !polus;
    }
digitalWrite(pin,  HIGH);
iterator++;
delay(DELAY_TIME);
digitalWrite(pin, LOW);
delay(DELAY_TIME);
if(iterator>= 24) exit(0);
}
```

Лабораторна робота № 4
Дослідження традиційних симетричних криптосистем шифрування інформації.

Мета роботи: реалізувати один із традиційних симетричних крипто алгоритмів шифрування інформації.

4.1 Основні теоретичні відомості

Більшість засобів захисту інформації базується на використанні криптографічних шифрів і процедур шифрування – розшифрування.

Під шифром розуміють сукупність зворотних перетворень множини відкритих даних на множину зашифрованих даних, що задаються ключем і алгоритмом криптографічного перетворення.

Ключ – це конкретний секретний стан деяких параметрів алгоритму криптографічного перетворення даних, що забезпечує вибір лише одного варіанту із усіх можливих для даного алгоритму [13].

Основною характеристикою шифру є криптостійкість, яка визначається його стійкістю до розкриття методами крипто аналізу. Як правило ця характеристика визначається інтервалом часу, необхідним для розкриття шифру.

До шифрів, які використовуються для криптографічного захисту інформації, висувається ряд вимог:

- Достатня криптостійкість (надійність закриття даних);
- Простота процедури шифрування та розшифрування;
- Незначна надлишковість інформації за рахунок шифрування;
- Нечутливість до незначних помилок шифрування и т.д.

В тій чи іншій мірі цим вимогам відповідають:

- Шифри перестановки;
- Шифри заміни;
- Шифри гамування;
- Шифри, основані на аналітичних перетвореннях шифрованих даних.

Шифрування перестановкою. Полягає в тому, щоб символи шифрованого тексту переставляються по певному правилу в межах певного блоку цього тексту. При достатній довжині блоку в межах чого здійснюється перестановка, і складному порядку перестановки, що не повторюється можна досягнути прийнятних для простих практичних використань стійкості шифру.

Шифрування заміною (підстановкою). Полягає в тому, що символи шифрованого тексту замінюються символами тог ж чи іншого алфавіту у відповідності з завідома обумовленою схемою заміни.

Шифрування гамуванням. Полягає в тому, що символи шифрованого тексту складаються з символами деякої випадкової послідовності, що зветься гаммою шифру. Стійкість шифрування визначається в основному довжиною (періодом) неповторюваної частини гамми шифру. Оскільки за допомогою ПК можна генерувати практично нескінченну гаму шифру то даний спосіб є одним із основних для шифрування інформації в автоматизованих системах.

Шифрування аналітичним перетворенням полягає в тому, що шифрований текст перетворюється по деякому аналітичному правила (формулі).

Наприклад можна використовувати правило множення вектора на матрицю, при цьому матриця множення є ключем шифрування.

Як відкритий текст так і шифр-текст складається з букв, що входить вскінченну множину символів, названих алфавітом.

Прикладом алфавітів є скінченна множина усіх заголовних букв, скінченна множина усіх заголовкових і стрічкових букв і цифр і т.д.

В загальному вигляді:

$$\sum\{a_0, a_1, a_2, a_3, a_4, \dots., a_{m-1}.\}$$

Об'єднуючи по певному правилу букви алфавіту $\sum$ можна створити нові алфавіти:

- алфавіт: $\sum^2$, що містить m^2 , биграм $a_0a_0, a_1a_1, a_2a_2, \dots., a_{m-1}a_{m-1}$.
- алфавіт: $\sum^3$, що містить m^3 , биграм $a_0a_0a_0, a_1a_1\,a_1, \dots., a_{m-1}a_{m-1}a_{m-1}$.

В загальному випадку, об'єднуючи по n букв, отримуємо алфавіти $\sum^n$ n-грам [14].

Наприклад, український алфавіт

$$\sum\{А, Б, В, Г, Ґ, \dots Ь, Ю, Я\}$$

Об'ємом m=33 букви дозволяє згенерувати шляхом операції конкатенації алфавіт із 33^2=1089 біграм.

АА, АВ, …., ЮЯ, ЯЯ,

Алфавіт із 33^3=35937триграм.

ААА, ААВ, …., ЮЮЯ, ЯЯЯ, і так далі.

При виконанні криптографічних перетворень корисно замінювати букви алфавіту цілими числами {0,1,2,3,4, ….32}.
Це дозволяє спростити виконання необхідними алгебраїчних операцій.

Таблиця відповідності між українським алфавітом і множиною цілих чисел.

буква	число	буква	число	буква	число	буква	число
А	0	**З**	9	**О**	18	**Ч**	27
Б	1	**И**	10	**П**	19	**Ш**	28
В	2	**І**	11	**Р**	20	**Щ**	29
Г	3	**Ї**	12	**С**	21	**Ь**	30
Ґ	4	**Й**	13	**Т**	22	**Ю**	31
Д	5	**К**	14	**У**	23	**Я**	32
Е	6	**Л**	15	**Ф**	24		
Є	7	**М**	16	**Х**	25		
Ж	8	**Н**	17	**Ц**	26		

4.2 Шифр перестановки

При шифруванні перестановкою символи шифруючого тексту переставляються по певному правилу межах блоку цього тексту. Шифри перестановки є самі прості.

В якості ключа в шифруючи таблицях використовуються:

- Розмір таблиці;
- Слово чи фраза, що задає перестановку;
- Особливості структури таблиці.

Проста перестановка.

Одним із самих примітивних табличних шифрів перестановки є проста перестановка, для якої ключем слугує розмір таблиці.

Наприклад: повідомлення «ПАРИ У ВІВТОРОК ПРОВОДИТИСЬ НЕ БУДУТЬ» записуємо в таблиці почергово по стовпчиках. Результат заповнення таблиці з 5 рядків і 7 стовпчиків приведено на рисунку 4.1.

П	В	Р	О	Т	Е	Т
А	І	О	В	И	Б	Ь
Р	В	К	О	С	У	О
И	Т	П	Д	Ь	Д	О
У	О	Р	И	Н	У	О

Рисунок 4.1 – Заповнена таблиця

Шифр текст матиме вигляд (групи по 5 букв)

ПВРОТ ЕТАІО ВИБЬР ВКОСУ ОИТПД ЬДОУО РИНУО

Звичайно, відправник та одержувачзаздалегіть повинні домовитись про ключ у вигляді розмірності таблиці.

Одиночною перестановкою по ключу

Дещо більшою криптостійкістю володіє метод шифрування що зветься одиночною перестановкою по ключу.

Цей метод відрізняється від попереднього тим, що стовбці таблиці представляються по ключовому слову, фразі чи набору чисел довжиною в рядок таблиці.

Застосуємо в якості ключа слово ГРУПА

Г	**Р**	**У**	**П**	**А**
2	4	5	3	1
П	**В**	**Р**	**И**	**Д**
А	**Т**	**О**	**С**	**У**
Р	**О**	**В**	**Ь**	**Т**
И	**Р**	**О**	**Н**	**Ь**
У	**О**	**Д**	**Е**	**А**
В	**К**	**И**	**Б**	**Б**
І	**П**	**Т**	**У**	**В**

До перестановки

А	**Г**	**П**	**Р**	**У**
1	2	3	4	5
У	**П**	**И**	**В**	**Р**
Т	**А**	**С**	**Т**	**О**
Ь	**Р**	**Ь**	**О**	**В**
А	**И**	**Н**	**Р**	**О**
Б	**У**	**Б**	**О**	**Д**
В	**В**	**У**	**К**	**И**
Г	**І**	**Д**	**П**	**Т**

Після перестановки

Рисунок 4.2 – Таблиці, заповнені ключовим словом і текстом повідомлення

В верхній стрічці лівої таблиці записаний ключ, а номера під буквами ключа визначені у відповідності з порядком в алфавіті. Якщо в ключі зустрічаються однакові букви, вони нумеруються з ліва на право. В правій таблиці стовбці представлені у відповідності з впорядкованими номерами букв ключа.

При зчитуванні вмісту правої таблиці по рядках і запису шрифт тексту групами по п’ять букв отримаємо шифр текст:

УПИВР ТАСТО ЬРЬОВ АИНРО БУБОД ВВУКИ ГІДПТ

Подвійна перестановка.

Для забезпечення додаткової криптостійкості можна повторно зашифрувати повідомлення, яке вже пройшло шифрування. Такий метод шифрування називається подвійною перестановкою.

У випадку подвійної перестановки стовпчиків і стрічок таблиці перестановки визначаються окремо для стовпчиків і окремо для стрічок.

Спочатку в таблицю записується текст повідомлення, а потім почергово переставляються стовпчики а потім рядки. При розшифруванні порядок перестановок повинен бути зворотнім.

Повідомлення: НАВЧАННЯ ВОСЬМОГО

	3	1	2	4
4	**Н**	А	В	Ч
1	А	Н	Н	Я
2	В	О	С	Ь
3	М	О	Г	О

вихідна таблиця

	1	2	3	4
4	А	В	Н	Ч
1	Н	Н	А	Я
2	О	С	В	Ь
3	О	Г	М	О

перестановка стовбців

	1	2	3	4
1	Н	Н	А	Я
2	О	С	В	Ь
3	О	Г	М	О
4	А	В	Н	Ч

перестановка стрічок

Рисунок 4.3 – Приклад виконання шифрування методом подвійної перестановки

Приклад виконання шифрування методом подвійної перестановки показаний на рисунку 4.3. Якщо зчитувати шифр текст з правої таблиці пострічково блоками по чотири букви, то отримаємо наступний шифр текст:

НННАЯ ОСВЬ ОГМО АВНЧ

Ключем до шифру подвійної перестановки є послідовність номерів стовбців і номерів стрічок вихідної таблиці (в нашому прикладі послідовність 3124 і 4123 відповідно).

Число варіантів подвійної перестановки швидко зростає при збільшенні розміру таблиці

- 3x3 - 36 вар.
- 4x4 - 576 вар.
- 5x5 - 14400 вар.

Застосування магічних квадратів

Магічним квадратом називаються квадратні таблиці з вписаними в їхні клітинки послідовні натуральні числа починаючи з 1, які дають в сумі по кожному стовбцю, кожній стрічці і кожній діагоналі одне й теж число.

Зашифрований текст вписується в квадрат у відповідності з нумерацією їх клітинок. Якщо потім виписати вміст такої таблиці по стрічкам, то отримаємо шифр текст, сформований завдяки перестановці букв вихідного повідомлення.

Приклад магічного квадрату і його заповнення повідомленням:

НАВЧАННЯ ВОСЬМОГО

Наведено на рисунку 4.4.

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

О	В	А	М
А	О	С	Я
В	Н	Н	Ь
Ч	Г	О	Н

Рисунок 4.4 – Приклад магічного квадрата 4х4 і його заповнення повідомленням.

Шифр текст.

ОВАМ АОСЯ ВННЬ ЧГОН

Число магічних квадратів швидко зростає з збільшенням розміру квадрату. Існує лише 1 магічний квадрат розміром 3х3 (якщо не враховувати його повтори). Кількість магічних квадратів 4х4 вже 880, а 5х5 близько 250000.

Магічні квадрати середніх і великих розмірів слугували чудовою базою для забезпечення потреб шифрування, оскільки практично не реально виконати вручну перебір усіх варіантів для такого шифру.

4.3 Шифри простої заміни

При шифруванні заміною (підстановкою) символи шифрованого тексту замінюються символами того ж чи іншого алфавіту з завідома встановленим правилом заміни.

В шифрі простої заміни кожний символ вихідного тексту замінюється символами того ж алфавіту однаково на усьому тексті. Часто їх називають шифрами одно алфавітної підстановки.

Система шифрування Цезаря

При шифруванні вихідного тексту кожна буква замінювалась на іншу букву того ж алфавіту за певним правилом (див. рисунок 4.5).

Шифр заміни при зміщенні **К=4.**

А	→	Ґ	**З**	→	Й	**О**	→	Т	**Ч**	→	Ю
Б	→	Д	**И**	→	К	**П**	→	У	**Ш**	→	Я
В	→	Е	**І**	→	Л	**Р**	→	Ф	**Щ**	→	А
Г	→	Є	**Ї**	→	М	**С**	→	Х	**Ь**	→	Б
Ґ	→	Ж	**Й**	→	Н	**Т**	→	Ц	**Ю**	→	В
Д	→	З	**К**	→	О	**У**	→	Ч	**Я**	→	Г
Е	→	И	**Л**	→	П	**Ф**	→	Ш			
Є	→	І	**М**	→	Р	**Х**	→	Щ			
Ж	→	Ї	**Н**	→	С	**Ц**	→	Ь			

Рисунок 4.5 – Зміна літер при зміщенні К=4

НАВЧАННЯ ВОСЬМОГО →СҐЄЮҐССГЕТХБРТЕТ

Афінська система підстановки Цезаря

В системі шифрування Цезаря використовувалось лише адитивні властивості множини цілих Z_m. Однак символи множини можна також помножити по модулю m.

Визначимо перетворення в такій системі:

$$\mathrm{E}_{a,b}(t) = at + b(mod\ m)$$

Де a, b - цілі числа, $0 \leq a, b < m$).

В данному перетворені букви, відповідають числу t, замінюються на букви, які відповідають числовому значенню $(at + b)$ по модулю m.

Наприклад: нехай $m = 32$, a=3, b=5 отримаємо 3t+5 $(mod\ 32)$.

t	0	1	2	3	4	5	6	7	8	9	10
3t+5	**5**	**8**	**11**	**14**	**17**	**20**	**23**	**26**	**29**	**0**	**3**
t	11	12	13	14	15	16	17	18	19	20	21
3t+5	**6**	**9**	**12**	**15**	**18**	**21**	**24**	**27**	**30**	**1**	**4**
t	22	23	24	25	26	27	28	29	30	31	32
3t+5	**7**	**10**	**13**	**16**	**19**	**22**	**25**	**28**	**31**	**2**	**5**

Перетворюємо числа в букви українського алфавіту, отримуємо наступну відповідність для букв відкритого тексту і шифр тексту:

А	**Б**	**В**	**Г**	**Ґ**	**Д**	**Е**	**Є**	**Ж**	**З**	**И**	**І**	**Ї**	**Й**	**К**	**Л**	**М**
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
Д	**Ж**	**І**	**К**	**Н**	**Р**	**У**	**Ц**	**Щ**	**А**	**Г**	**Е**	**З**	**Ї**	**Л**	**О**	**С**

Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
Ф	Ч	Ь	Б	Ґ	Є	И	Й	М	П	Т	Х	Ш	Ю	В	Д

Система Цезаря з ключовим словом є одно алфавітною системою підстановки. Особливістю цієї системи є використання ключового слова для зміщення і змінення порядку символів в алфавіті підстановки.

Виберемо деяке число $k0 \leq k < 32$ і слово чи коротку фразу в якості ключового слова. Бажано щоб букви в ключовому слові не повторювались.

Нехай вибране слово «МОДУЛЬ» в якості ключового слова і число к=5.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
А	Б	В	Г	Ґ	Д	Е	Є	Ж	З	И	І	Ї	Й	К	Л	М
					М	О	Д	У	Л	Ь						

17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я

Літери алфавіту, що залишились записуються після ключового слова в алфавітному порядку.

0					5					10						
А	Б	В	Г	Ґ	Д	Е	Є	Ж	З	И	І	Ї	Й	К	Л	М
Ч	Ш	Щ	Ю	Я	М	О	Д	У	Л	Ь	А	Б	В	Г	Ґ	Е

Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ь	Ю	Я
Є	Ж	З	И	І	Ї	Й	К	Н	П	Р	С	Т	Ф	Х	Ц

Тепер ми маємо підстановку для кожної літери повідомлення.

Повідомлення – **СЛОВА ЦЕ ГОЛОС БІЛЬШ НІЧОГО**

Шифр текст - **І ҐЖЩЧ ПО ЮЖҐЖІ ШФҐФС ЄФРЖЮЖ**

Шифр таблиці Трісемуса

Для отримання такого шифру заміни використовується таблиця для запису букв алфавіту і ключове слово (чи фраза). В таблицю спочатку вписується по стрічках ключове слово, при чому літери що повторюються відкидаються. Потім ця таблиця доповнюється літерами алфавіту, що не увійшли до неї по порядку. Оскільки ключове слово чи фразу легко

зберігати в пам’яті, то такий підхід спрощує процес шифрування та розшифрування.

Розглянемо цей процес шифрування на прикладі. Для українського алфавіту шифр таблиця може мати розмір 4×9. Для повного заповнення таблиці використовуються незначущі символи. Виберемо в якості ключа слово КЛЕМА. Шифр таблиця з таким ключем показана на рисунку 4.6.

К	**Л**	**Е**	**М**	**А**	Б	В	Г	Ґ
Д	Є	Ж	З	И	І	Ї	Й	Н
О	П	Р	С	Т	У	Ф	Х	Ц
Ч	Ш	Щ	Ь	Ю	Я	.	,	_

Рисунок 4.6 – Шифр таблиця з ключовим словом КЛЕМА

При шифруванні в таблиці находять наступну літеру відкритого тексту і записують в шифр текст букву, розташовану нижче неї в тому ж стовбці. Якщо літера знаходиться в нижній стрічці таблиці для шифр тексту беруть саму верхню літеру того ж стовпчика.

Наприклад, при шифруванні за допомогою цієї таблиці повідомлення:

НУ_ЩО_Б_ЗДАВАЛОСЯ_СЛОВА

Отримує шифр текст:

ЦЯ ҐЕЧ ҐСОИЇИЄЧЬМБҐЬЄЧЇИ

Такі табличні шифри називаються моногамними, так як шифрування здійснюється по одній літері.

4.4 Шифри складної заміни

Система шифрування Віжінера [15] подібна до тієї системи шифрування Цезаря, в якій ключ підстановки змінюється від букви до букви. Це шифр багато алфавітної заміни можна описати таблицею шифрування, що називається таблицею (квадратом) Віжінера. На рисунку 4.7. Наведено таблицю Віжінера для українського алфавіту.

Таблиця Віджінра використовується для шифрування та розшифрування. Таблиця має два входи:

- Верхню стрічку підкреслених символів, використовують для зчитування відповідної літери вихідного відкритого тексту;
- Крайній лівий стовпчик ключа.

Послідовність ключів як правило отримують із числових значень букв ключового слова.

При шифруванні вихідного повідомлення його виписують в стрічку, а під ним записують ключове слово (чи фразу). Якщо виявилось, що ключ коротший за повідомлення, то його циклічно повторюють. В процесі шифрування знаходять у верхній стрічці таблиці наступну літеру вихідного тексту і в лівому стовбці наступне значення ключа.

Розглянемо приклад отримання шифр тексту за допомогою таблиці Віжінера.

Нехай вибране слово ключ КАРАНТИН.

І нам необхідно зашифрувати повідомлення: РОЗПОЧАВСЯ КАРАНТИН.

ключ	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я
0	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я
1	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а
2	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б
3	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в
4	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г
5	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ
6	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д
7	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е
8	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є
9	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж
10	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з
11	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и
12	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і
13	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї
14	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й
15	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к
16	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л
17	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м
18	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н
19	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о
20	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п
21	с	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р
22	т	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с
23	у	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т
24	ф	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у
25	х	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф
26	ц	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х
27	ч	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц
28	ш	щ	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч
29	щ	ь	ю	Я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш
30	ь	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ
31	ю	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь
32	я	а	б	в	г	ґ	д	е	є	ж	з	и	і	ї	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ь	ю

Рисунок 4.7 – Квадрат (таблиця) Віжінера.

В результаті проведення шифрування отримаємо шифр текст – БОЩПВМИПБЯБАҐТЧДФН

Шифр «подвійний квадрат» Уітстона відкрив новий етап в історії розвитку криптографії. На відміну від полібіанського шифру «подвійний квадрат» використовує зразу дві таблиці, розміщені по одній горизонталі, а шифрування йде біграмами.

Ці не складні модифікації призвели до виникнення якісно нової криптографічної системи ручного шифрування. Шифр «подвійний квадрат» виявився дуже надійним і зручним.

Пояснимо процедуру шифрування цим шифром на прикладі. Нехай є дві таблиці з випадково розташованими в них українським алфавітом (рис. 4.8). Перед шифрування вихідне повідомлення розбивається на біграми. Кожна біграма шифрується окремо. Першу літеру біграми находять у лівій таблиці, а другу у правій. Потім умовно будують прямокутник так, щоб літери біграми розташовувались у його протилежних вершинах. Інші дві вершини цього прямокутника дають букви біграми шифр тексту.

-	Ч	Ф	_	Л	А
Б	У	Ж	Щ	М	И
І	З	Н	Ц	В	Р
С	О	Г	.	Ї	Ю
Я	Д	Й	П	Т	Г’
Є	К	Х	Е	Ь	,

Ж	Щ	М	И	Б	У
Ф	_	Л	А	-	Ч
Й	П	Т	Г’	Я	Д
Н	Ц	В	Р	І	З
Х	Е	Ь	,	Є	К
Г	.	Ї	Ю	С	О

Рисунок 4.8 – Дві таблиці з випадково вписаними символами українського алфавіту для шифру «подвійний квадрат»

Припустимо, що шифрується біграма вихідного тексту КР. Літера К знаходиться в стовпчику 2 і стрічці 6 лівої таблиці. Літера Р знаходиться в стовпчику 4 і стрічці 4 правої таблиці (див. рис. 9). Ще означає, що в біграму шифр тексту входить літера Ю правої таблиці та літера О лівої таблиці.

-	Ч	Ф	_	Л	А
Б	У	Ж	Щ	М	И
І	З	Н	Ц	В	Р
С	О	Г	.	Ї	Ю
Я	Д	Й	П	Т	Г’
Є	К	Х	Е	Ь	,

Ж	Щ	М	И	Б	У
Ф	_	Л	А	-	Ч
Й	П	Т	Г’	Я	Д
Н	Ц	В	Р	І	З
Х	Е	Ь	,	Є	К
Г	.	Ї	Ю	С	О

Рисунок 4.9 – «подвійний квадрат» Уітстона

Якщо обидві літери біргами повідомлення лежать на одній стрічці, то літери шифр тексту беруть з цієї ж стрічки. Першу літеру біграми шифртексту беруть з лівої таблиці в стовпчику, що відповідає другій літері біграми повідомлення. Друга літера біграми шифр тексту береться з першої таблиці в стовпчику що відповідає першій літер повідомлення.

Приклад біграма повідомлення ЖА буде відповідати біграмі ЛЩ шифр тексту.

Аналогічним чином шифруються усі біграми повідомлення:

Повідомлення – КРИПТОГРАФІЧНА СТІЙКІСТЬ

Шифр текст – ЮО_РКЬВ.ЖИДБҐЖБЄЄЇЄПЯЄЄҐ

Ширування методом «подвійного квадрату» дає доволі стійкий до взламу і простий у використанні шифр. Взлом шифр тексту вимагає значних зусиль.

Одноразова система шифрування (одноразовий блокнот).

Майже усі шифри які застосовуються на практиці характеризуються як умовно надійні, оскільки вони можуть бути зламані при наявності необмежених обчислюваних можливостей. Абсолютно надійні шифри не можливо розкрити навіть при використанні необмежених обчислювальних можливостей. Існує єдиний такий шифр, що застосовується на практиці – одноразова система шифрування. Характерною властивістю одноразової системи шифрування є одноразове використання ключової послідовності.

Одноразова система шифрування винайдена Дж. Моборном і Г. Вернамом [16]. Для реалізації цієї системи інколи використовують одноразовий блокнот. Цей блокнот складений із сторінок на кожній з яких надрукована таблиця з випадковими числами (ключами). Блокнот виконується в двох екземплярах: один використовується відправником інший одержувачем. Після того як таблиця буде використана вона повинна бути видалена. Кожне нове повідомлення починається з нової сторінки.

Цей шифр абсолютно надійний, якщо набір ключів дійсно випадковий та непередбачуваний.

В деяких варіантах одноразового блокнота застосовують більш простий метод керування ключовою послідовністю, але це призводить до деякого зниження надійності шифру. Наприклад ключ визначається вказаним місцем книги, відомої відправнику і отримувачу. Ключова послідовність починається з вказаного місця цієї книги.

Багатоалфавітна підстановка визначається ключем $\pi=(\pi_1, \pi_2, ...)$ що містить не менше двох різних підстановок. На початку розглянемо багатоалфавіті системи підстановок з нульовим початковим зміщенням.

Як інформація, що підлягає шифруванню і розшифруванню, будуть розглядатися тексти, побудовані на деякому алфавіті.

Під цими термінами розуміється наступне.

Алфавіт - кінцева множина використовуваних для кодування інформації знаків.

Текст - упорядкований набір з елементів алфавіту.

Як приклади алфавітів, що використовуються в сучасних ІС можна навести наступні:

алфавіт Z_{34} - 33 літери українського алфавіту і пробіл;

$$Z_{укр.} = \{А,Б,В,Г\ldots,Ь,Ю,Я, \_ \};$$

$$Z_{34} = \{0,1,2,3\ldots,31,32, 33\};$$

алфавіт Z_{256} - символи, що входять в стандартні коди ASCII і КОІ8;

бінарний алфавіт - $Z_2 = \{0,1\}$;

Величина ключів К системи одноразовою підстановки є вектором рангів (K_0, K_1, ..., K_{n-1}) і містить m^n точок.

Розглянемо невеликий приклад шифрування з нескінченним ключем. Як ключ приймемо текст "НЕСКІНЧЕННІЙ_КЛЮЧ ....".

Зашифруємо за його допомогою текст “ГРУПА_КН_ЗАЛІК”. Шифрування оформимо у вигляді таблиці:

ГРУПА_КН_ЗАЛІК	4	21	24	20	1	34	15	18	34	10	1	16	12	15
НЕСКІНЧЕННИЙ_КЛЮЧ ...	18	7	22	15	12	18	28	7	18	18	11	14	15	16
Σ	22	28	46	35	13	52	43	25	52	28	12	30	27	31
Mod 34	22	28	12	1	13	18	9	25	18	28	12	30	27	31

Отримаємо зашифроване повідомлення у вигляді «СЧІАЇНЖФНЧІЩЦЬ».

Оригінальний текст неможливо відновити без ключа.

Накладення білого шуму в вигляді нескінченного ключа на вихідний текст змінює статистичні характеристики мови джерела. Системи одноразового використання теоретично не розшифровуванні, так як не містять достатньої інформації для відновлення тексту.

4.3 Порядок виконання роботи

1. Розглянути основні теоретичні відомості традиційних симетричних криптосистем шифрування інформації.

2. У відповідності з варіантом лабораторної роботи розробити систему шифрування повідомлення з урахування заданих ключів (ключової послідовності).

3. У відповідності з варіантом лабораторної роботи розробити систему розшифрування шифр тексту з урахування заданих ключів (ключової послідовності).

4. Реалізувати дані алгоритми у вигляді програмного коду.

4.5 Зміст звіту

1. Звіт з лабораторної роботи повинен бути виконаний на листах формату А4.
2. Звіт повинен містити: назву лабораторної роботи, її мету і короткі теоретичні відомості.
3. В розділі «Результати виконання лабораторної роботи» студент повинен представити процедуру шифрування та розшифрування для заданої криптосистеми, а також навести код програмної реалізації процесу шифрування та розшифрування.
4. Написати висновки до даної лабораторної роботи.
5. Оформити звіт та представити його викладачу на захист.

Таблиця 4.1 - Варіанти завдань

Варіант	Криптосистема	Ключ / пояснення
1	Подвійна перестановка	Таблиця 9×12
2	Одиночною перестановкою по ключу	МАРКЕТИНГ
3	Магічний квадрат	Квадрат 5×5
4	Система шифрування Цезаря	К=8
5	Афінська системи підстановки Цезаря	a=5, b=3
6	Система Цезаря з ключовим словом	КАРТОН
7	Шифр таблиця Трісемуса	Таблиця 6x6, КРОПИВА
8	Система шифрування Віжінера	НУМЕРАЦІЯ
9	Подвійний квадрат	-
10	Одноразовий блокнот	[1] с 6, 3 абзац
11	Афінська системи підстановки Цезаря	a=2, b=5
12	Система шифрування Цезаря	К=6
13	Магічний квадрат	Квадрат 4×4
14	Подвійний квадрат	-
15	Подвійна перестановка	Таблиця 8×9
16	Система Цезаря з ключовим словом	РОЗВІДНИК
17	Система шифрування Віжінера	ЗАСТУПНИК
18	Афінська системи підстановки Цезаря	a=4, b=7
19	Одноразовий блокнот	[1] с 24, 6 абзац
20	Магічний квадрат	Квадрат 5×5
21	Одноразовий блокнот	[1] с 15, 1 абзац
22	Подвійний квадрат	-
23	Подвійна перестановка	Таблиця 6×10
24	Система Цезаря з ключовим словом	СКЕЙТБОРД
25	Система шифрування Віжінера	НОМЕР
26	Магічний квадрат	Квадрат 6×6
27	Система Цезаря з ключовим словом	РЕСПУБЛІКА
28	Подвійна перестановка	Таблиця 8×9
29	Система шифрування Цезаря	К=24
30	Система шифрування Віжінера	ГРАДУСНИК

Контрольні запитання

1. За яким правилом зчитуються біграми «подвійного квадрату»?
2. Що таке біграмма?
3. За яким правилом відбувається шифрування за допомогою подвійної перестановки?
4. Що розуміється у шифрування під словом «ключ»?
4. Що таке шифри багатоалфавітної заміни?
5. Які розміщуються літери у подвійному квадраті Уітстона?
6. Що таке шифр складної заміни?
7. Наведіть приклади алфавітів для одноразових систем шифрування?
8. Які перетворення необхідно виконати з ключем «карандаш» при шифруванні повідомлення за допомогою таблиці Трісемуса?

Додаток Г

```
// подвійна перестановка таблиця 8×9
#include <iostream>
#include <fstream>
#include <string>
#include "windows.h"
using namespace std;
void read(char** sm) {
  ifstream enFile; string inTxt;
  enFile.open("some.txt"); int pc = 0;
  int pg = 0; int ls;
  while (!enFile.eof()) {
    enFile >> inTxt;
    ls = inTxt.length();
    for (int i = 0; i < ls; i++) {
      sm[pg][pc] = inTxt[i];
      pc++;
      if (pc == 8) {
        pg++;
        pc = 0;
      }
    }
    break;
  }
  enFile.close();
}
void change(char** mainT, int* fKey, int* sKey, int* pf,
int* ps) {
  char buff;
  int k = 1;
```

```
  char* startMesVer = new char[8]; for (int i = 0; i < 8;
i++) {
    startMesVer[i] = mainT[0][i];
  }
  char* startMesHor = new char[9]; for (int i = 0; i < 9;
i++) {
    startMesHor[i] = mainT[i][0];
  }
  char* sortMesHor = new char[9]; for (int i = 0; i < 9;
i++) {
    sortMesHor[i] = mainT[i][0];
  }
  for (int j = 0; j < 9; j++) {
    for (int c = 0; c < 8; c++) {
      if (sortMesHor[c] > sortMesHor[c + 1]) {
        for (int i = 0; i < 8; i++) {
          buff = sortMesHor[c]; sortMesHor[c] =
sortMesHor[c + 1];
          sortMesHor[c + 1] = buff;
        }
      }
    }
  }
  int* MesIndex = new int[9]; for (int i = 0; i < 8; i++)
{
    if (i == 0) {
      MesIndex[i] = k; k++;
      continue;
    }
    if (sortMesHor[i] == sortMesHor[i - 1]) {
      MesIndex[i] = MesIndex[i - 1]; continue;
    }
    MesIndex[i] = k; k++;
  }
  for (int i = 0; i < 9; i++) {
    for (int j = 0; j < 9; j++) {
      if (startMesHor[i] == sortMesHor[j]) {
        sKey[i] = MesIndex[j];
      }
    }
  }
  for (int i = 0; i < 9; i++) {
    ps[i] = MesIndex[i];
  }
  for (int j = 0; j < 8; j++) {
    for (int c = 0; c < 7; c++) {
      if (mainT[0][c] > mainT[0][c + 1]) {
        for (int i = 0; i < 9; i++) {
```

```
          buff = mainT[i][c]; mainT[i][c] = mainT[i][c +
1];
          mainT[i][c + 1] = buff;
        }
      }
    }
  }
  delete[]MesIndex; k = 1;
  MesIndex = new int[8];
  for (int i = 0; i < 8; i++) {
    if (mainT[0][i] == mainT[0][i - 1]) {
      MesIndex[i] = MesIndex[i - 1]; continue;
    }
    MesIndex[i] = k; k++;
  }
  for (int i = 0; i < 8; i++) {
    for (int j = 0; j < 8; j++) {
      if (startMesVer[i] == mainT[0][j]) {
        fKey[i] = MesIndex[j];
      }
    }
  }
  for (int i = 0; i < 8; i++) {
    pf[i] = MesIndex[i];
  }
  char** buffT = new char*[9]; for (int i = 0; i < 9; i++)
{
    buffT[i] = new char[8];
  }
  for (int i = 0; i < 9; i++) {
    for (int j = 0; j < 9; j++) {
      if (sortMesHor[i] == startMesHor[j]) {
        for (int c = 0; c < 8; c++) {
          buffT[i][c] = mainT[j][c];
        }
        startMesHor[j] = '1'; break;
      }
    }
  }
  for (int i = 0; i < 9; i++) {
    for (int j = 0; j < 8; j++) {
      mainT[i][j] = buffT[i][j];
    }
  }
}
void ds(char** mainT, int* fKey, int* sKey, int* pf, int*
ps) {
  char** unMes = new char*[9];
```

```
  for (int i = 0; i < 9; i++) unMes[i] = new char[8];
  for (int i = 0; i < 9; i++) {
    for (int j = 0; j < 9; j++) {
      if (sKey[i] == ps[j]) {
        for (int c = 0; c < 8; c++) {
          unMes[i][c] = mainT[j][c];
        }
        ps[j] = 0; break;
      }
    }
  }
  for (int i = 0; i < 9; i++) {
    for (int j = 0; j < 8; j++) {
      mainT[i][j] = unMes[i][j];
    }
  }
  for (int i = 0; i < 8; i++) {
    for (int j = 0; j < 8; j++) {
      if (fKey[i] == pf[j]) {
        for (int c = 0; c < 9; c++) {
          unMes[c][i] = mainT[c][j];
        }
        pf[j] = 0;
        break;
      }
    }
  }
  for (int i = 0; i < 9; i++) {
    for (int j = 0; j < 8; j++) {
      mainT[i][j] = unMes[i][j];
    }
  }
}
int main() {
  SetConsoleCP(1251); SetConsoleOutputCP(1251); int
fKey[8];
  int sKey[9]; int pf[8]; int ps[9];
  char** mainT = new char*[9]; for (int i = 0; i < 9; i++)
{
    mainT[i] = new char[8];
  }
  read(mainT);
  cout << "Отриманий текст" << endl; for (int i = 0; i <
9; i++) {
    for (int j = 0; j < 8; j++) {
      cout << mainT[i][j] << " ";
    }
    cout << endl;
```

```
  }
  cout << endl;
  change(mainT, fKey, sKey, pf, ps); int los = 0;
  cout << "Зашифрований текст" << endl;
  for (int i = 0; i < 9; i++) {
    for (int j = 0; j < 8; j++) {
      cout << mainT[i][j]; los++;
      if (los == 4) {
        cout << " "; los = 0;
      }
    }
  }
  cout << endl;
  cout << "Перший ключ: "; for (int i = 0; i < 9; i++)
    cout << sKey[i] << " "; cout << endl << "Другий ключ:
"; for (int i = 0; i < 8;
      i++)
    cout << fKey[i] << " "; cout << endl << endl;
  ds(mainT, fKey, sKey, pf, ps);
  cout << "Розшифрований текст" << endl;
  for (int i = 0; i < 9; i++) {
    for (int j = 0; j < 8; j++) {
      cout << mainT[i][j] << " ";
    }
    cout << endl;
  }
  return 0;
}
```

Лабораторна робота № 5

Дослідження потокових алгоритмів шифрування інформації.

Мета роботи: дослідити симетричне потокове шифрування інформації на прикладі алгоритму RC-4, A5, скремблера та реалізувати один із наведених алгоритмів шифрування.

5. Основні теоретичні відомості

Потокові шифри - Одиницею кодування є один біт. Результат кодування не залежить від попереднього 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.

Потокові шифри – по бітна обробка інформації. 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.

На даний час створено велику кількість алгоритмів потокового шифрування. Представниками потокових шифрів є: A3, A5, A8, MUGI, PIKE, RC4, SEAL.

5.1 Потоковий шифр RC4

Потоковий шифр RC4 був створений Рональдом Ривестом, співробітником компанії RSA Security, в 1987 році . Скорочення «RC4» офіційно позначає «Rivestcipher 4» або «шифр Ривеста » («4» - номер версії; див. RC2, RC5, RC6). Алгоритм шифрування RC4 застосовується в деяких широко поширених стандартах і протоколах шифрування (наприклад, WEP, WPA, SSL і TLS) [17].

RC4 став популярний завдяки:

- простоті його апаратної і програмної реалізації;
- високій швидкості роботи алгоритму в обох випадках.

У США довжина ключа, рекомендована для використання всередині країни, дорівнює 128 бітам.

Ядро алгоритму поточних шифрів складається з функції - генератора псевдовипадкових бітів (гами), який видає потік бітів ключа (ключовий потік, гаму, послідовність псевдовипадкових бітів).

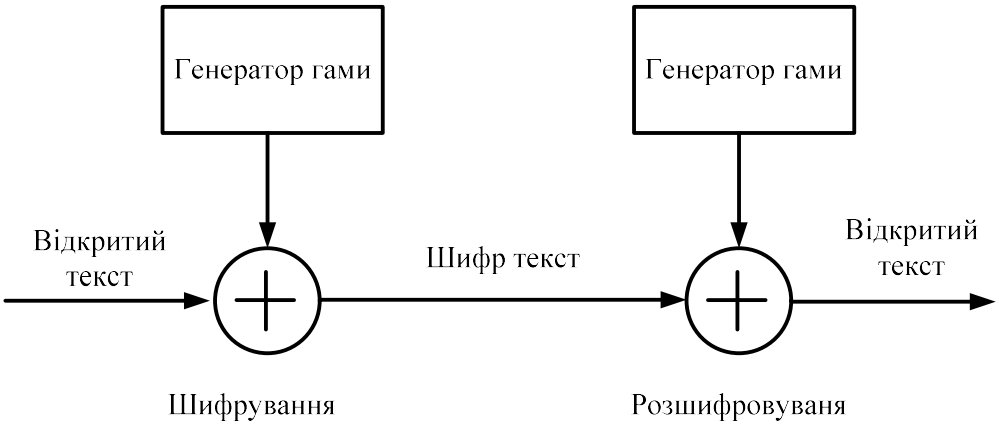


Рисунок 5.1 – Режим гамування для потокових шифрів

Алгоритм шифрування.

Функція генерує послідовність бітів (K_i).

Потім послідовність бітів за допомогою операції «добуток по модулю два» (XOR) об'єднується з відкритим текстом (m_i). В результаті виходить шифрограма (C_i):

$$C_i = m_i \oplus k_i.$$

Алгоритм розшифровки.

Повторно створюється (регенерується) потік бітів ключа (ключовий потік) (K_i).

Потік бітів ключа складається з шифрограмою (C_i) Операцією « XOR ». В силу властивостей операції « XOR » на виході отримуємо вихідний (незашифрований) текст (m_i):

$$m_i = C_i \oplus k_i = (m_i \oplus k_i) \oplus k_i.$$

RC4 - фактично клас алгоритмів, що визначаються розміром блоку (надалі S-блоку). Параметр n є розміром слова для алгоритму і визначає довжину S-блоку. Зазвичай, n=8, але в цілях аналізу можна зменшити його. Однак для підвищення безпеки необхідно збільшити цю величину. В алгоритмі немає протиріч на збільшення розміру S-блоку. При збільшенні n, припустимо, до 16 біт, елементів в S-блоці стає 65 536 і відповідно час початкової ітерації буде збільшено.

Внутрішній стан RC4 представляється у вигляді масиву розміром 2^n і двох лічильників. Масив відомий як S-блок, і далі буде позначатися як S. Він завжди містить перестановку 2 n можливих значень слова. Два лічильника позначені через i та j.

Ініціалізація RC4 складається з двох частин:

- ініціалізація S-блоку;
- генерація псевдовипадкового слова K.

ІНІЦІАЛІЗАЦІЯ S-БЛОКУ

Алгоритм також відомий як «KSA». Цей алгоритм використовує ключ, що подається на вхід користувачем, збережений в *KEY* і має довжину *L* байт. Ініціалізація починається з заповнення масиву *S*, далі цей масив перемішується шляхом перестановок, які визначаються ключем. Так як тільки одну дію виконується над *S*, то повинно виконуватися твердження, що *S* завжди містить один набір значень, який був даний при початкової ініціалізації (S [i]: = i).

ГЕНЕРАЦІЯ ПСЕВДОВИПАДКОВОГО СЛОВА K

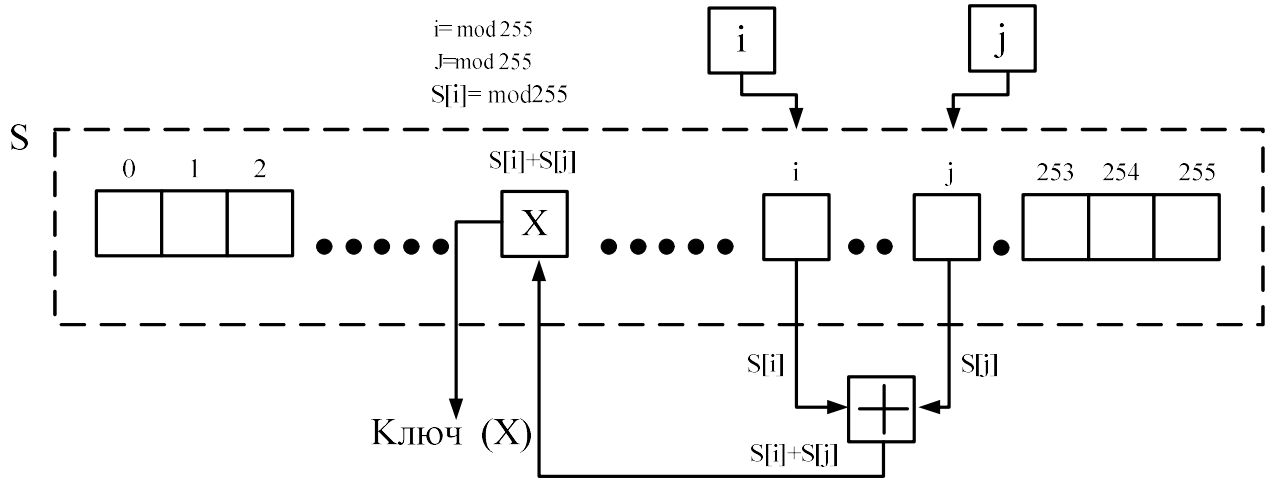


Рисунок 5.2

Ця частина алгоритму називається генератором псевдовипадкової послідовності. Генератор ключового потоку RC4 переставляє значення, що зберігаються в S. В одному циклі RC4 визначається одне n -бітне слово K з ключового потоку. Надалі ключове слово буде складено по модулю два з вихідним текстом, яке користувач хоче зашифрувати, і отриманий зашифрований текст.

Клас алгоритму, визначаються розміром його блоку або слова параметром *n*.

За звичай n=8 , але можна використовувати і інші значення. Для спрощення аналізу алгоритму приймемо n=4 . Внутрішній станRC4 складається з масиву розміром 2^n слів і двох лічильників, кожен розміром в однеслово. Два лічильника, обидва при n = 4 4-бітові, назвемо *i* і *j*.

Подивимось,як ініціалізується таблиця S .

Спочатку вона заповнюється послідовно числами від 0 до 2^n.

Ініціалізуємо таблицю S .

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**

Ключ представляється у вигляді послідовності n-бітових слів, якими заповнюється інший масив K , такого ж розміру, як S. Якщо ключ виявився коротшим, ніж нам необхідно, він повторюється необхідну кількість разів.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
K_i	**К**	**Л**	**Ю**	**Ч**	**К**	**Л**	**Ю**	**Ч**	**К**	**Л**	**Ю**	**Ч**	**К**	**Л**	**Ю**	**Ч**

Потім виконуються наступні дії (алгоритм 1):

1. j = 0; i = 0;
2. $j = (j + S_i + K_i) \bmod 2^n$;
3. поміняти місцями S_i і S_j ;
4. i = i +1;
5. якщо $i < 2^n$, то перейти на п.2

В результаті виконання цього алгоритму проводиться початкове заповнення таблиці замін S, при чому це початкова перемішування значень проводиться в залежності від секретного ключа.

Після того як таблиця S підготовлена, можна починати генерацію випадкових *n*-бітових слів. Для цього лічильникам *i* та *j* присвоюється початкове значення 0. Потім для отримання кожного нового значення z_i виконуються наступні дії (алгоритм 2):

$i = (i + 1) \bmod 2^n$;

$j = (j + S_i) \bmod 2^n$;

поміняти місцями S_i і S_j ;

$a = (S_i + S_j) \bmod 2^n$;

$z_i = S_a$.

Отримане 4-бітове значення z_i можевикористовуватися в якості ключа для шифрування чергового 4-бітового блоку вхідного потоку даних.

Приклад

Розглянемо на прикладі 4-бітних значень. Але для зручності обчислень дані представляємо в десятковій системі числення.

Всі обчислення проводяться по модулю 2^n. В нашому випадку mod 2^4=16.

Ініціалізуємо таблицю S .

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**

Ключ представляється у вигляді послідовності 4-бітових слів, якими заповнюється інший масив K, такого ж розміру, як S. Якщо ключ виявився коротшим, ніж треба, він повторюється необхідну кількість разів.

Ключ – **840** (Для зручності обчислень ключ приймаємо в десятковій системі числення)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
K_i	**8**	**4**	**0**	**8**	**4**	**0**	**8**	**4**	**0**	**8**	**4**	**0**	**8**	**4**	**0**	**8**

1-ша ітерація;
i =0; j=0.

$j = (j + S_i + K_i) \bmod 16$; $j = (0 + 0 + 8)\ [\text{mod}16] = 8$
поміняти місцями S_0 и S_8;

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_i	**8**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**0**	**9**	**10**	**11**	**12**	**13**	**14**	**15**

2-га ітерація;
i =1; j=8.
$j = (8 + 1 + 4)\ [\text{mod}16] = 13$
поміняти місцями S_1 и S_{13};

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_i	**8**	**13**	**2**	**3**	**4**	**5**	**6**	**7**	**0**	**9**	**10**	**11**	**12**	**1**	**14**	**15**

3-тя ітерація;
i =2; j=13.
$j = (13 + 1 + 0)\ [\text{mod}16] = 14$
поміняти місцями S_2 и S_{14};

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_i	**8**	**13**	**14**	**3**	**4**	**5**	**6**	**7**	**0**	**9**	**10**	**11**	**12**	**1**	**2**	**15**

4-та ітерація;
i =3; j=14.
$j = (14 + 3 + 8)\ [\text{mod}16] = 12$
поміняти місцями S_3 и S_{12};

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_i	**8**	**13**	**14**	**12**	**4**	**5**	**6**	**7**	**0**	**9**	**10**	**11**	**3**	**1**	**2**	**15**

5-та ітерація;
i =4; j=14.
$j = (14 + 4 + 4)\ [\text{mod}16] = 6$
поміняти місцями S_4 и S_6;

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_i	**8**	**13**	**14**	**12**	**6**	**5**	**4**	**7**	**0**	**9**	**10**	**11**	**3**	**1**	**2**	**15**

6-та ітерація;

i =5; j=6.

j = (6 + 5 + 0) [mod16] =11

поміняти місцями S_5 и S_{11};

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_i	**8**	**13**	**14**	**12**	**6**	**11**	**4**	**7**	**0**	**9**	**10**	**5**	**3**	**1**	**2**	**15**

7-ма ітерація;

i =6; j=11.

j = (j + S_i + K_i) mod16; j = (11 +7 + 8) [mod16] =10

поміняти місцями S_6 и S_{10};

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_i	**8**	**13**	**14**	**12**	**6**	**11**	**10**	**7**	**0**	**9**	**4**	**5**	**3**	**1**	**2**	**15**

8-ма ітерація;

i =7; j=10.

j = (j + S_i + K_i) mod16; j = (10 +7 + 4) [mod16] =5

поміняти місцями S_7 и S_5;

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_i	**8**	**13**	**14**	**12**	**6**	**7**	**10**	**11**	**0**	**9**	**4**	**5**	**3**	**1**	**2**	**15**

9-та ітерація;

i =8; j=5.

j = (j + S_i + K_i) mod16; j = (5 +0 + 0) [mod16] =5

поміняти місцями S_8 и S_5;

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_i	**8**	**13**	**14**	**12**	**6**	**0**	**10**	**11**	**7**	**9**	**4**	**5**	**3**	**1**	**2**	**15**

10-та ітерація;

i =9; j=5.

j = (j + S_i + K_i) mod16; j = (5 +9 + 8) [mod16] =6

поміняти місцями S_9 и S_6;

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_i	**8**	**13**	**14**	**12**	**6**	**0**	**9**	**11**	**7**	**10**	**4**	**5**	**3**	**1**	**2**	**15**

11-та ітерація;

i =10; j=6.

j = (j + S_i + K_i) mod16; j = (6 +4 + 4) [mod16] =14

поміняти місцями S_{10} и S_{14};

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_i	**8**	**13**	**14**	**12**	**6**	**0**	**9**	**11**	**7**	**2**	**4**	**5**	**3**	**1**	**10**	**15**

12-та ітерація;

i =11; j=14.

$j = (j + S_i + K_i) \bmod 16$; $j = (14 + 5 + 0) [\bmod 16] = 3$

поміняти місцями S_{11} и S_3;

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_i	**8**	**13**	**14**	**5**	**6**	**0**	**9**	**11**	**7**	**2**	**4**	**12**	**3**	**1**	**10**	**15**

13-та ітерація;

i =12; j=3.

$j = (j + S_i + K_i) \bmod 16$; $j = (3 + 3 + 8) [\bmod 16] = 14$

поміняти місцями S_{12} и S_{14};

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_i	**8**	**13**	**14**	**5**	**6**	**0**	**9**	**11**	**7**	**2**	**4**	**12**	**10**	**1**	**3**	**15**

14-та ітерація;

i =13; j=14.

$j = (j + S_i + K_i) \bmod 16$; $j = (14 + 1 + 4) [\bmod 16] = 3$

поміняти місцями S_{13} и S_3;

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_i	**8**	**13**	**14**	**1**	**6**	**0**	**9**	**11**	**7**	**2**	**4**	**12**	**10**	**5**	**3**	**15**

15-та ітерація;

i =14; j=3.

$j = (j + S_i + K_i) \bmod 16$; $j = (3 + 3 + 0) [\bmod 16] = 6$

поміняти місцями S_{14} и S_6;

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_i	**8**	**13**	**14**	**1**	**6**	**0**	**3**	**11**	**7**	**2**	**4**	**12**	**10**	**5**	**9**	**15**

16-та ітерація;

i =15; j=6.

$j = (j + S_i + K_i) \bmod 16$; $j = (6 + 15 + 8) [\bmod 16] = 13$

поміняти місцями S_{15} и S_{13};

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_i	**8**	**13**	**14**	**1**	**6**	**0**	**3**	**11**	**7**	**2**	**4**	**12**	**10**	**15**	**9**	**5**

Отримали ініціалізовану таблицю:

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_i	**8**	**13**	**14**	**1**	**6**	**0**	**3**	**11**	**7**	**2**	**4**	**12**	**10**	**15**	**9**	**5**

Після того як таблиця S підготовлена, можна починати генерацію випадкових *n*-бітових слів. Для цього лічильникам *i* та *j* присвоюється початкове значення 0. Потім для отримання кожного нового значення z_i виконуються наступні дії:

i =0; j=0.
i = (i + 1) mod 16;
j = (j + S_i) mod 16;
поміняти місцями S_i і S_j ;
a = (S_i + S_j) mod 16;
$z_i = S_a$.

Отримане 4-бітове значення z_i можевикористовуватися в якості ключа для шифрування чергового 4-бітового блоку вхідного потоку даних.

1-ша ітерація;
i =0; j=0.

i = (i + 1) mod 16; i = (0 + 1) mod 16; i =1
j = (j + S_i) mod 16; j = (0 + 13) mod 16; j =13

поміняти місцями S_1 і S_{13} ;

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_i	**8**	**15**	**14**	**1**	**6**	**0**	**3**	**11**	**7**	**2**	**4**	**12**	**10**	**13**	**9**	**5**

a=(S_i + S_j) mod 16; a=(15 + 13) mod 16; a =12;
$z_0 = S_{12}$; z_0 =10.

2-га ітерація;
i =1; j=13.

i = (i + 1) mod 16; i = (1 + 1) mod 16; i =2
j = (j + S_i) mod 16; j = (13 + 15) mod 16; j =12

поміняти місцями S_2 і S_{12} ;

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_i	**8**	**15**	**10**	**1**	**6**	**0**	**3**	**11**	**7**	**2**	**4**	**12**	**14**	**13**	**9**	**5**

$a = (S_i + S_j) \bmod 16;$ $a = (10 + 14) \bmod 16;$ $a = 12$

$z_0 = S_{12};$ $z_0 = 14.$

3-тя ітерація;

i =2; j=12.

$i = (i + 1) \bmod 16;$ $i = (2+ 1) \bmod 16;$ $i = 3$

$j = (j + S_3) \bmod 16;$ $j = (12 + 1) \bmod 16;$ $j = 13$

поміняти місцями S_3 і S_{13} ;

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_i	**8**	**15**	**10**	**13**	**6**	**0**	**3**	**11**	**7**	**2**	**4**	**12**	**14**	**1**	**9**	**5**

$a = (S_3 + S_{13}) \bmod 16;$ $a = (1 + 13) \bmod 16;$ $a = 14$

$z_0 = S_{12};$ $z_0 = 14.$

4-та ітерація;

i =3; j=14.

$i = (i + 1) \bmod 16;$ $i = (3+ 1) \bmod 16;$ $i = 4$

$j = (j + S_4) \bmod 16;$ $j = (14 + 6) \bmod 16;$ $j = 4$

поміняти місцями S_4 і S_4 ;

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_i	**8**	**15**	**10**	**13**	**6**	**0**	**3**	**11**	**7**	**2**	**4**	**12**	**14**	**1**	**9**	**5**

$a = (S_4 + S_4) \bmod 16;$ $a = (6 + 6) \bmod 16;$ $a = 12$

$z_0 = S_{12};$ $z_0 = 14.$

5-та ітерація;

i =4; j=4.

$i = (i + 1) \bmod 16;$ $i = (4+ 1) \bmod 16;$ $i = 5$

$j = (j + S_5) \bmod 16;$ $j = (4 + 0) \bmod 16;$ $j = 4$

поміняти місцями S_5 і S_4 ;

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_i	**8**	**15**	**10**	**13**	**0**	**6**	**3**	**11**	**7**	**2**	**4**	**12**	**14**	**1**	**9**	**5**

$a = (S_4 + S_5) \bmod 16;$ $a = (0 + 6) \bmod 16;$ $a = 6$

$z_0 = S_6;$ $z_0 = 3.$

6-та ітерація;

i =5; j=4.

$i = (i + 1) \bmod 16;\ \ i = (5+ 1) \bmod 16;\ \ \ i = 6$

$j = (j + S_6) \bmod 16;\ \ j = (4 + 3) \bmod 16;\ \ j = 7$

поміняти місцями S_6 і S_7 ;

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_i	**8**	**15**	**10**	**13**	**4**	**0**	**11**	**3**	**7**	**2**	**6**	**12**	**14**	**1**	**9**	**5**

$a = (S_6 + S_7) \bmod 16;\ \ a = (11 + 3) \bmod 16;\ \ \ a = 14$

$z_0 = S_{14}$; $\ \ z_0 = 9.$

І так далі, поки не буде сформовано псевдовипадкову послідовність довжиною, що рівна вхідній послідовності.

5.2 Потоковий алгоритм шифрування А5

А5 - це потоковий алгоритм шифрування, що використовується для забезпечення конфіденційності даних між мобільним телефоном і базовою станцією в європейській системі мобільного цифрового зв'язку GSM [17].

Шифр заснований на побітовому добутку по модулю два (булева операція «виключне або (XOR)») генерованої псевдовипадкової послідовності і вихідної інформації. У А5псевдовипадкова послідовність реалізується на основі трьох лінійних регістрів зсуву зі зворотним зв'язком. Регістри мають довжини 19, 22 і 23 біти відповідно. Зсувом управляє спеціальна схема, яка організовує на кожному кроці зміщення як мінімум двох регістрів, що призводить до їх нерівномірного руху. Послідовність формується шляхом операції «виключне АБО» над вихідними бітами регістрів.

Після того як виникла необхідність розширення застосування GSM стандарту А5 перейменували в А5/1 і стали поширювати як в Європі, так і в США. Для інших країн (в тому числі України і Росії) алгоритм модифікували, значно знизивши крипостійкість шифру. А5/2 був спеціально розроблений як експортний варіант для країн, що не входили до Євросоюзу. Крипостійкість А5/2 була знижена додаванням ще одного регістра (17 біт), який керує зсувом інших. А5/0 шифрування відсутня зовсім. В даний час розроблений також алгоритм А5/3, заснований на алгоритмі Касумі і затверджений для використання в мережах 3G. Ці модифікації позначають як А5 / х.

Алгоритм А5 в даний час - це ціле сімейство шифрів. Для прикладу візьмемо А5/1, як основоположника.

Рисунок 5.3 – Схема поточного шифру: додавання по модулю два відкритого тексту і послідовності біт дає шифр текст

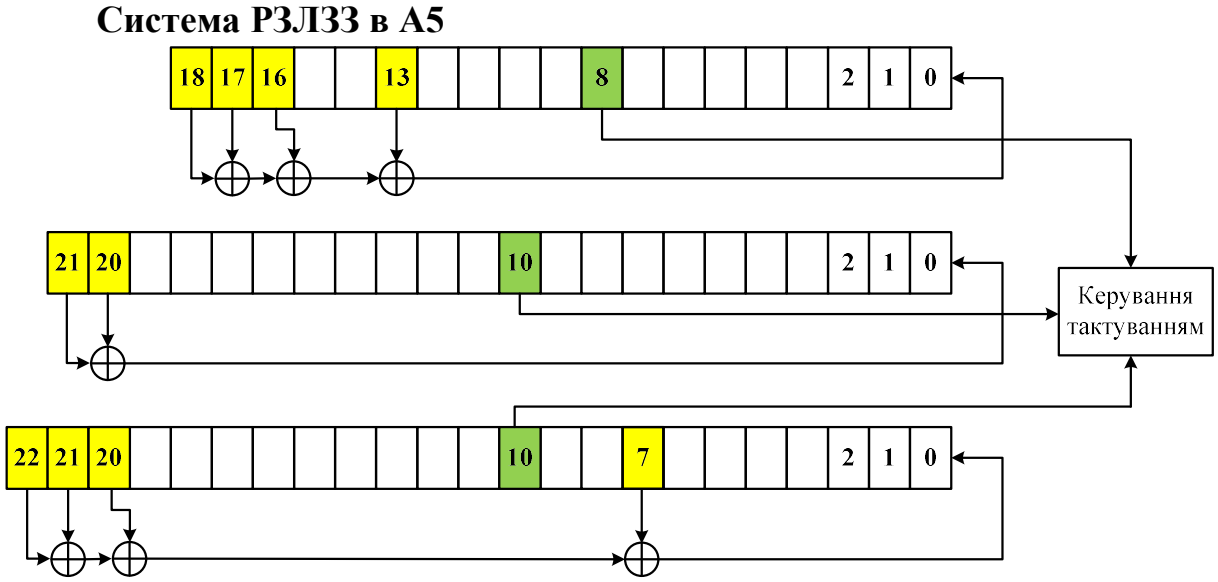


Рисунок 5.4 – Система регістрів в алгоритмі А5/1

Сам по собі регістр зсуву з лінійним зворотним зв'язком (РЗЛЗЗ) легко піддається крипто аналізу і не є достатньо надійним для використання в шифруванні. Практичне застосування мають системи регістрів змінного тактування з різними довжинами і функціями зворотного зв'язку.

Структура алгоритму А5 виглядає наступним чином:

• Три регістра (R1, R2, R3) мають довжини 19, 22 і 23 біта,

• Багаточлени зворотних зв'язків:

X19 + X18 + X17 + X14 + 1 для R1,

X22 + X21 + 1 для R2,

X23 + X22 + X21 + X8 + 1 для R3,

• Керування тактами здійснюється спеціальним механізмом:

- в кожному регістрі є біти синхронізації: 8 (R1), 10 (R2), 10 (R3),
- обчислюється функція:

$$F = x\&y\ OR\ x\&z\ OR\ y\&z,$$

де & - логічне AND «і», OR - булеве «АБО», а x, y і z - біти синхронізації R1, R2 і R3 відповідно,

- зсуву піддаються лише ті регістри, у яких біт синхронізації дорівнює *F*,
- фактично, зсуваються регістри, синхробіти яких належать до більшості,

• Вихідний біт системи - результат операції XOR над вихідними бітами регістрів.

Функціонування алгоритму А5

Розглянемо особливості функціонування алгоритму на основі відомої схеми. Передача даних здійснюється в структурованому вигляді - з розбивкою на кадри (114 біт). Перед ініціалізацією регістри обнуляються, на вхід алгоритму надходять сеансовий ключ (K - 64 біта), сформований А8, і номер кадру (F_n - 22 біта). Далі послідовно виконуються наступні дії:

• Ініціалізація:
- 64 такти, при яких черговий біт ключа XOR-иться з молодшим бітом кожного регістру, регістри при цьому зсуваються на кожному такті;
- аналогічні 22 такту, тільки операція XOR проводиться з номером кадру;
- 100 тактів з керуванням регістрів зсуву, але без генерації послідовності;

• 228 (114+114) тактів робочі, відбувається шифрування переданого кадру (перші 114 біт) і дешифрування (останні 114 біт) прийнятого;

• далі ініціалізація проводиться заново, використовується новий номер кадру.

5.3 Скремблер

Скремблер – це набір біт, які змінюються покроково по визначеному алгоритму. Після виконання кожного наступного кроку на його виході з'являється шифруючий біт (0 або 1), який накладається на поточний біт інформаційного потоку операцією XOR [18].

Скремблювання - це зворотне перетворення цифрового потоку без зміни швидкості передачі з метою отримання властивостей, близьких до властивостей випадкової послідовності. Оригінал тексту можна відновити, застосувавши зворотний алгоритм.

Для синхронної передачі двійковий сигнал повинен відповідати двом основним вимогам:

1. частота зміни символів (1, 0) повинна забезпечувати надійне виділення тактової частоти безпосередньо з прийнятого сигналу;

2. спектральна щільність потужності переданого сигналу повинна бути, по можливості, постійною і зосередженою в заданій області частот з метою зниження взаємного впливу каналів.

Одним із способів обробки двійкових посилок, що задовольняє даним вимогам є скремблювання (*scramble* - перемішування).

Після скремблювання поява «1» і «0» в вихідний послідовності приблизно рівно ймовірна. Скремблювання також може використовуватися для певного захисту переданої інформації, а також для ідентифікації абонентів.

Основною частиною скремблера є генератор псевдовипадкової послідовності (ПВП) у вигляді лінійного n-каскадного регістра з зворотними зв'язками, що формує послідовність максимальної довжини 2n-1.

Розрізняють два основних типи скремблерів і дескремблерів – само синхронізуючих (СС) і з установкою (адитивні). В літературі також можна зустріти інші назви - скремблери з неізольованим і ізольованим від лінії зв'язку генераторами псевдовипадкових послідовностей.

Особливістю само синхронізуючого скремблера (СС скремблера) (рис. 5.5) є те, що він керується скрембльованою послідовністю, тобто тією, яка передається в канал.

Тому при даному виді скремблювання не потрібно спеціальної установки станів скремблера і дескремблера; Скрембльована послідовність записується в регістри зсуву скремблера і дескремблера, встановлюючи їх в ідентичне стан. при втраті синхронізації між скремблером і дескремблером час відновлення синхронізації не перевищує числа тактів, рівного числу елементів регістра скремблера.

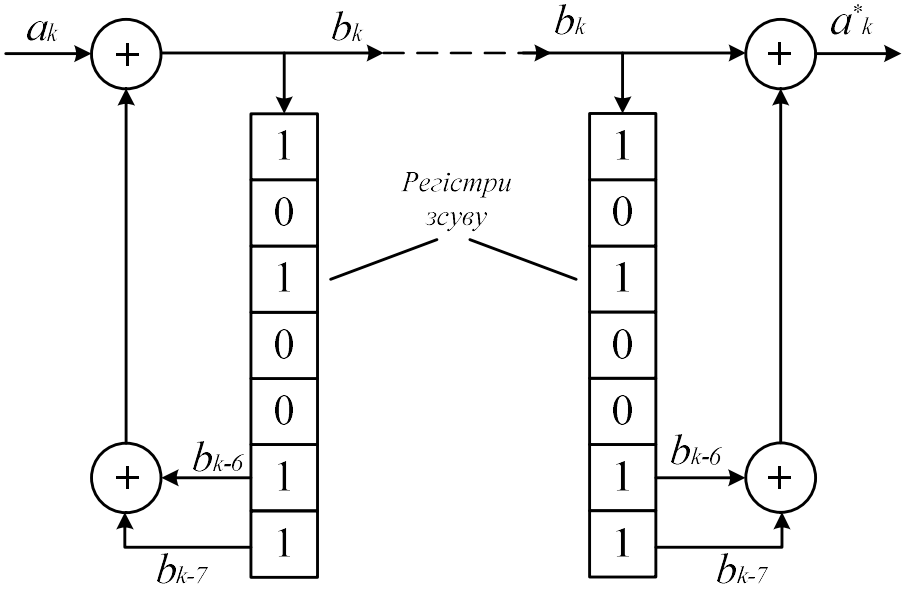


Рисунок 5.5 – Самосинхронізуючий скремблер і дескремблер.

На приймальній стороні виділення вихідної послідовності відбувається шляхом додавання за модулем два прийнятої скрембльованої послідовності з послідовністю на виході зсувного регістру. Наприклад, для

схеми рис. 5.5 вхідна послідовність a_k за допомогою скремблера відповідно до співвідношення $b_k = a_k \oplus (b_{k-6} \oplus b_{k-7})$ перетворюється в відправну двійкову послідовність b_k. У приймчі з цієї послідовності таким же регістром зсуву, як на прийомі, формується послідовність $a^*{}_k = b_k \oplus (b_{k-6} \oplus b_{k-7})$ ця послідовність на виході дескремблера ідентична первісної послідовності a_k.

Як випливає з принципу дії схеми, при одній помилці в послідовності b_k помилковими виходять також наступні шостий і сьомий символи (в даному прикладі). В даному прикладі - число зворотних α раз, де α число зворотних зв'язків. Таким чином, СС скремблер – дескремблер має властивість розмноження помилок.

Даний недолік СС скремблера - дескремблера обмежує число зворотних зв'язків в регістрі зсуву; практично це число не перевищує $\alpha = 2$.

Другий недолік СС скремблера пов'язаний з можливістю появи на його виході при визначених умовах так званих «критичних ситуацій», коли вихідна послідовність набуває періодичний характер з періодом, меншим довжини ПСП.

Щоб запобігти цьому, в скремблер і дескремблері передбачаються спеціальні додаткові схеми контролю, які виявляють наявність періодичності елементів на вході і порушують її.

Недоліки, властиві СС скремблер-дескремблер, практично відсутні при адитивному Скремблюванні (рис. 5.6) проте, цей тип скремблерів-дескремблерів вимагає попередньої ідентичної установки станів регістрів скремблера і дескремблера. В скремблері з установкою (АД-скремблер), як і в СС скремблері, проводиться підсумовування вхідного сигналу і ПСП, але результуючий сигнал не надходить на вхід регістра. В дескремблер і скрембльований сигнал також не проходить через регістр зсуву, тому розмноження помилок не відбувається.

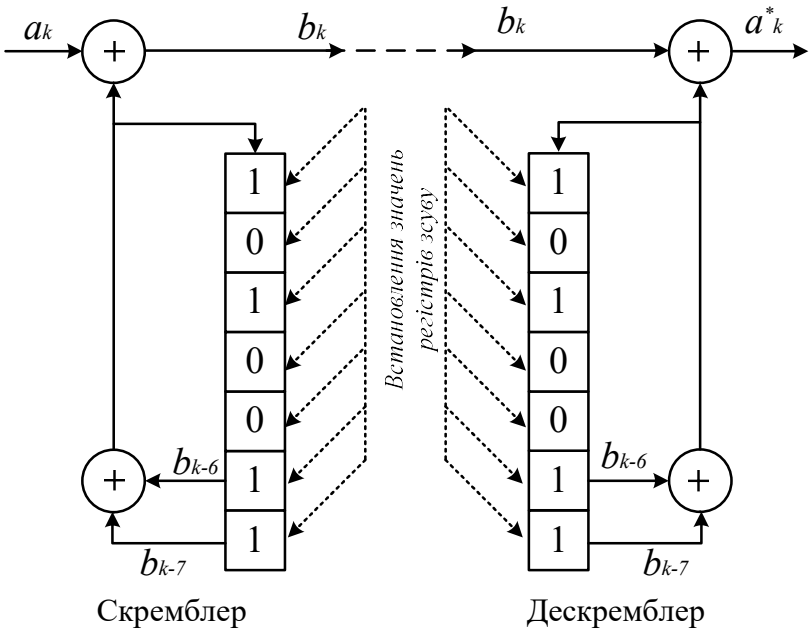


Рисунок. 5.6 – Адитивні скремблер і дескремблер.

Підсумовані в скремблері послідовності незалежні, тому їх період завжди дорівнює найменшому спільному короткому величин періодів вхідної послідовності і ПВП і критичний стан відсутній. Відсутність ефекту розмноження помилок і необхідності в спеціальній логіці захисту від небажаних ситуацій роблять спосіб адитивного скремблювання краще, якщо не враховувати витрат на вирішення завдання фазування скремблера і дескремблера. Як сигнал установки в ЦСП використовують сигнал циклової синхронізації.

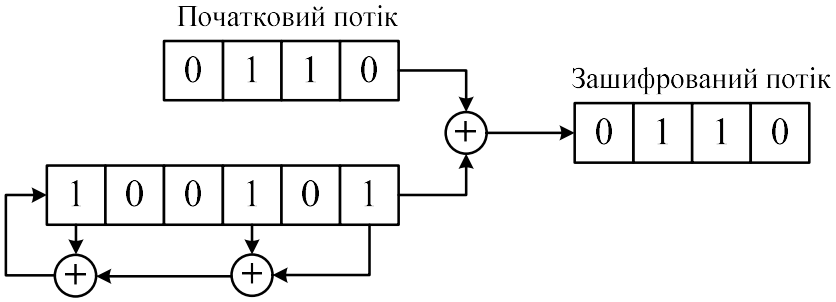


Рисунок 5.7 – Структура шести розрядного скремблера

З теорії передачі даних криптографія запозичила для запису подібних схем двійкову систему запису. По ній зображений на рисунку 5.7 скремблер записується комбінацією «100101_2» - одиниці відповідають розрядам, з яких знімаються біти для формування зворотного зв'язку.

5.3 Порядок виконання роботи

1. Розглянути основні теоретичні відомості потокових симетричних криптосистем шифрування інформації.

2. Проаналізувати наведені в додатку Д одного з приклади програмного коду побудови потокових шифрів.

3. На підставі проведеного аналізу у відповідності з варіантом лабораторної роботи розробити систему шифрування повідомлення з урахування властивостей певного алгоритму.

4. На підставі проведеного аналізу у відповідності з варіантом лабораторної роботи розробити систему розшифрування шифр тексту з урахування властивостей певного алгоритму.

5. Реалізувати дані алгоритми у вигляді програмного коду.

5.4 Зміст звіту

1. Звіт з лабораторної роботи повинен бути виконаний на листах формату А4.

2. Звіт повинен містити: назву лабораторної роботи, її мету і короткі теоретичні відомості.

3. В розділі «Результати виконання лабораторної роботи» студент повинен представити процедуру шифрування або розшифрування, в залежності від варіанту, для заданої криптосистеми, а також навести код програмної реалізації.

4. Написати висновки до даної лабораторної роботи.

5. Оформити звіт та представити його викладачу на захист.

Таблиця 5.1 - Варіанти завдань

Варіант	Вид потокового шифру	Додаткові дані необхідні для заданого варіанту
1	RC-4	n=4; ключ 742113
2	Адитивний скремблер	Схема «1001010100_2»; стан регістру зсуву 1010101010
3	Самосинхронізуючийскремблер	Схема «1001010100_2»; стан регістру зсуву 1110101010
4	A5	R1 – 0001110101010101010 R2 –10101011101010101010 R3 – 10101011101010101010101
5	Ізольований від лінії зв’язку скремблер	Схема «1010100100_2»; стан регістру зсуву 1011001010
6	Неізольований від лінії зв’язку скремблер	Схема «1101000100_2»; стан регістру зсуву 1011101110
7	RC-4	n=4; ключ 742113
8	Адитивний скремблер	Схема «1001010100_2»; стан регістру зсуву 1010101010
9	Самосинхронізуючийскремблер	Схема «1010010100_2»; стан регістру зсуву 1110101010
10	A5	R1 – 0011111101010101010 R2 –10000011101110111101010 R3 – 10110101110001101010101

Продовження таблиця 5.1 - Варіанти завдань

Варіант	Вид потокового шифру	Додаткові дані необхідні для заданого варіанту
11	Ізольований від лінії зв'язку скремблер	Схема «1010100100_2»; стан регістру зсуву 1011001010
12	Неізольований від лінії зв'язку скремблер	Схема «0010100100_2»; стан регістру зсуву 1110101100
13	A5	R1 –1010101010100011111 R2 –11101110000001011101010 R3 –00110101010110110101110
14	Самосинхронізуючий скремблер	Схема «0101000100_2»; стан регістру зсуву 1011001010
15	RC-4	n=4; ключ 352448
16	Адитивний скремблер	Схема «0100010100_2»; стан регістру зсуву 1010100101
17	Самосинхронізуючий скремблер	Схема «0001010001_2»; стан регістру зсуву 1010101010
18	A5	R1 – 0001111111010101010 R2 –10101011100000010101010 R3 – 10101011010101110101010
19	Ізольований від лінії зв'язку скремблер	Схема «1010100100_2»; стан регістру зсуву 1010101010
20	Неізольований від лінії зв'язку скремблер	Схема «0101000101_2»; стан регістру зсуву 0101101010
21	RC-4	n=4; ключ 608123
22	Адитивний скремблер	Схема «1110000100_2»; стан регістру зсуву 0101010100
23	Самосинхронізуючий скремблер	Схема «0001110100_2»; стан регістру зсуву 0010011010
24	A5	R1 – 0001110101010101010 R2 –00001011000000101000000 R3 –10101101010110101010101
25	RC-4	n=4; ключ 883071
26	Адитивний скремблер	Схема «1100010001_2»; стан регістру зсуву 1111101010
27	Самосинхронізуючийскремблер	Схема «1010010101_2»; стан регістру зсуву 0011001100
28	A5	R1 –10000111010101010010 R2 –10101010111011010010010 R3 –10101101010111010010101
29	Ізольований від лінії зв'язку скремблер	Схема «0010110110_2»; стан регістру зсуву 0000111100
30	Неізольований від лінії зв'язку скремблер	Схема «1100010011_2»; стан регістру зсуву 0111111111

Контрольні запитання

1. Перелічіть основні переваги та недоліки адитивного скремблера?
2. Перелічіть основні переваги та недоліки само синхронізуючого скремблера?
3. Яка основна відмінність алгоритму шифрування A5/0 від інших модифікацій алгоритму A5/x?
4. За яким правилом здійснюється зміщення регістрів R1, R2,R3РЗЛЗЗ?
5. Вкажіть який із регістрів R1, R2,R3 буде здійснювати зсув при наступних значеннях x, y, z:

№	x	y	z
1	1	1	0
2	1	0	1
3	0	1	0
4	0	0	1
5	1	1	0
6	1	0	1

6. Опишіть у чому полягає принцип ініціалізації S таблиці алгоритму RC-4?
7. Опишіть у чому полягає принцип ініціалізації K таблиці алгоритму RC-4?
8. Що таке генератор псевдо випадкової послідовності біт?
9. Як працює генератор псевдо випадкової послідовності біт на основі регістру зсуву?

Додаток Д

```
#include <fstream>
#include <iostream>
#include <bitset>
using namespace std;

string BUFF;

void Read(string in){
    ifstream readFile(in);

    if (!readFile.is_open())
        cout << "Не знайдено файл для читання.
Скремблювання/дискремблювання неможливе"<< endl;
    else{
    getline(readFile,BUFF);
```

```
    readFile.close();
    cout << "Файл зчитаний" << endl;
    }
}

void Scrembler (){
     char key[] = "101010111";
     char start[] = "010110101";
     for(int i,t=0; i<=BUFF.size();i++){
          BUFF[i] = start[t] ^ BUFF[i];
          start[t]=key[t];
          t++;
          if(t>sizeof(key))t=0;
     }
     cout << "Скремблювання завершено. Запис файлу"<< endl;
}

void Descrembler(){
     char  key[] = "101010111";
     char start[] = "010110101";
     for(int i,t=0; i<=BUFF.size();i++){
          BUFF[i] = start[t] ^ BUFF[i];
          start[t]=key[t];
          t++;
          if(t>sizeof(key))t=0;
     }
     cout << "Дескремблювання завершено. Запис файлу"<<
endl;
}

void ConvertToBits(){
     bitset <9> bitbuff[BUFF.size()];

     for(int i =0;i< BUFF.size();i++){
          bitbuff[i]= BUFF[i];
     }

     int size = BUFF.size();
     BUFF.clear();

     for(int i =0; i < size; i++){
          BUFF += bitbuff[i].to_string<char,
char_traits<char>, allocator<char> >();
     }
}

void ConvertToStr(){
     string copy;
```

```
     char mas[10];
     int k;
     long int temp;
     char *pEnd;
     for(int i = 0; i < BUFF.size();){
          k = 0;
          do{
               mas[k] = BUFF[i];
               i++;
               k++;
          }while(i % 9 != 0);
          temp = strtol(mas,&pEnd,2);
          copy += temp;
     }
     BUFF.clear();
     BUFF+=copy;
}

void WriteS(string in, string out){
     Read(in);

     ConvertToBits();

     Scrembler();

     ofstream wrouteFile(out);
     if(!wrouteFile.is_open())
        cout << "Не можливо відкрити/створити файл для
запису"<< endl;
    else{
          wrouteFile << BUFF;
         wrouteFile.close();
         cout << "Запис завершено"<< endl;
     }

}

void WriteD(string in, string out){
     Read(in);

     Descrembler();

     ConvertToStr();

     ofstream wrouteFile(out);
     if(!wrouteFile.is_open())
        cout << "Не можливо відкрити/створити файл для
запису"<< endl;
```

```
    else{
         wrouteFile << BUFF;
        wrouteFile.close();
        cout << "Запис завершено"<< endl;
     }
}

void Interface(){
     string curent_str;
    string name_in;
    string name_out;

     cout << "Виберіть дію" <<endl;
    cout << "1-Скремблювання\n2-Дескремблювання" <<endl;
    cin >> curent_str;

    cout << "Введіть повну назву файлу для зчитування " <<
endl;
     cin >> name_in;
     cout << "Введіть повну назву файлу для запису " <<
endl;
     cin >> name_out;

     if(curent_str == "1"){
          WriteS(name_in,name_out);
     }else if(curent_str == "2"){
          WriteD(name_in,name_out);
     }else {
          cout << "Невірний вибір дії" <<endl;
     }
     curent_str.clear();
     cout << "Бажаєте продовжити?(Y/N)"<<endl;
     cin >> curent_str;
     if(curent_str == "Y"){
          BUFF.clear();
          Interface();
     }else{
          cout << "Дякую за використання шифратора."<<endl;
     }
}

int main()
{
    setlocale(LC_ALL, "rus");
     Interface();
    system("pause");
    return 0;
}
```

Лабораторна робота № 6

Дослідження симетричних блокових алгоритмів шифрування інформації на основі мережі Фейстеля

Мета роботи:дослідити симетричні блокові алгоритми шифрування інформації на основі мережі Х. Фейстеля, реалізувати один із наведених варіантів шифрування інформації у відповідності з завданням.

6. Основні теоретичні відомості

Як зазначалось раніше, головний недолік абсолютно крипто стійкого шифру одноразових блокнотів – велика довжина ключа, яка має бути не меншою від довжини повідомлення. Через це використовувати його на практиці складно [19].

Для побудови стійкого шифру, зручного для практичного використання, можна запропонувати такі підходи:

1. Блоковість.

Вибираємо довжину ключа меншою за довжину повідомлення, розбиваємо повідомлення на окремі блоки і шифруємо кожний з них (крім, можливо, останнього) шляхом сумування з ключем по модулю два.

2. Режими шифрування.

Для збільшення стійкості блокового шифрування можна забезпечити використання при шифруванні кожного наступного блоку результатів шифрування попереднього блоку (наприклад, в якості нового ключа шифрування для блоку). Тоді зловмисник не зможе розшифрувати блок криптотексту, доки не розшифрує всі попередні блоки.

3. Багатораундовість.

Додатково збільшити стійкість блокового шифрування можна з рахунок багаторазового виконання шифрування кожного блоку за умови, що функція шифрування є нелінійним перетворенням.

В блокових алгоритмах вхідна послідовність розбивається на блоки – ділянки певної довжини (найчастіше, по 64 біти). Якщо довжина відкритого тексту виявляється некратною довжині блоку, застосовується операція доповнення (padding) останнього блоку до необхідної довжини, яка полягає у дописуванні необхідної кількості нулів або випадкового набору символів (Рис.6.1).

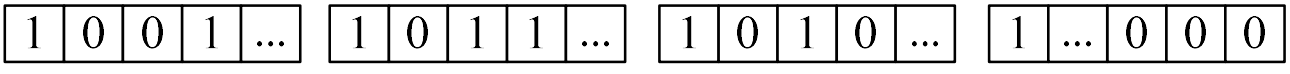


Рисунок 6.1 – Представлення даних в блоковому алгоритмі

Криптографічне перетворення в блокових алгоритмах шифрування здійснюється над кожним блоком окремо (Рис.6.2). Його сутність полягає у застосуванні до блока багаторазово математичного перетворення. Внаслідок цього результуюче перетворення виявляється криптографічно більш сильним, ніж перетворення над окремо взятим блоком. Метою таких перетворень є створення залежності кожного біту блоку шифротексту від кожного біту ключа і кожного біту відкритого тексту:

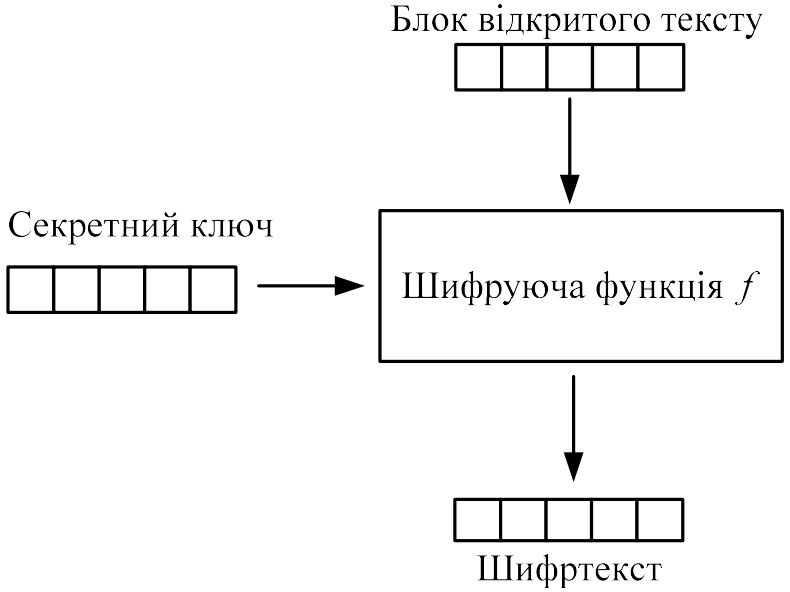


Рисунок 6.2 – Схема шифрування блоку даних в блоковому алгоритмі

Зашифрований блок може використовуватися для шифрування наступного, в результаті чого кожний блок отримає контекст, що властивий всьому повідомленню.

6.1 Мережа Фейстеля

Блоковий алгоритм перетворює *n*-бітний блок незашифрованого тексту в *n*-бітний блок зашифрованого тексту. Кількість блоків довжини *n* рівна 2n. Для того, щоб перетворення було зворотним, кожний з таких блоків повинен перетворюватися у свій унікальний блок зашифрованого тексту [20, 21].

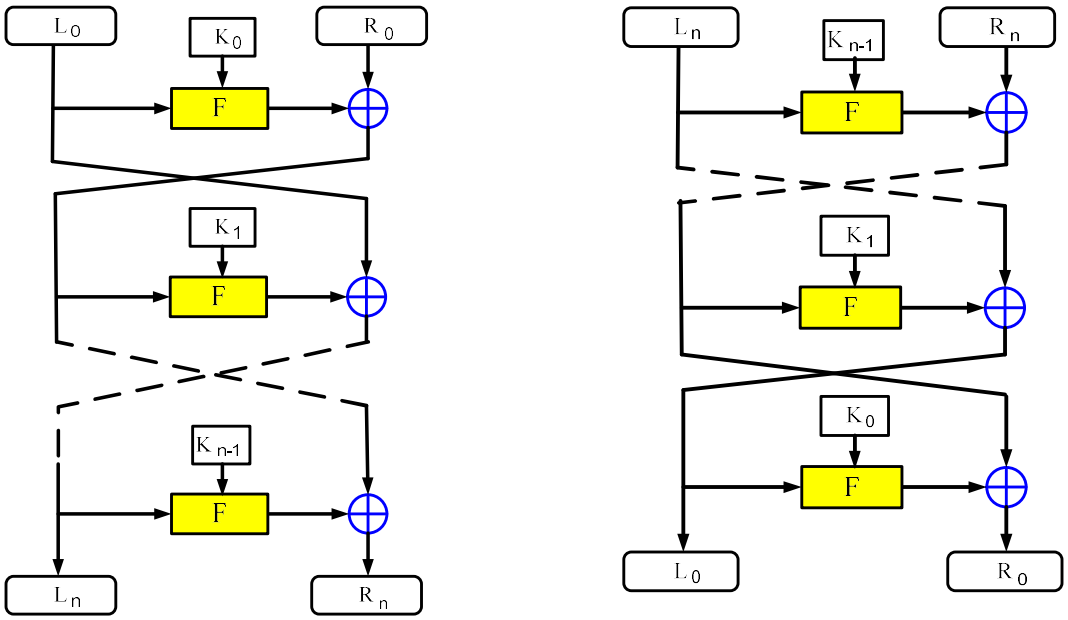
- блок відкритого тексту ділиться на 2 рівні частини (L_0 і R_0)
- в кожному раунді вираховується (***i=1….n*** — номер раунду)

$$L_i = R_{i-1} \oplus f(L_{i-1},\ K_{i-1});$$
$$R_i = L_{i-1},$$

де f – деякафункція, а K_{i-1} – ключ *i*-го раунду. Результатом виконання n раундів є $L_n;\ R_n$. Але зазвичай в n-му раунді перестановка $L_n;\ R_n$ і не виконуються, що дозволяє використовувати ту ж процедуру і для розшифрування, просто інвертувавши порядок використання раундової ключової інформації:

$$L_{i-1} = R_i \oplus f(L_i,\ K_{i-1});$$
$$R_{i-1} = L_i,$$

Невеликі зміни дозволяють досягнути повної ідентичності процедур шифрування та розшифрування. Одною із переваг такої моделі є застосовність алгоритму незалежно від функції f, і вона може бути довільної складності.



Шифрування Розшифрування

Рисунок 6.3 – Мережа Фейстеля

При великому розмірі блоків шифрування (128 біт і більше) реалізація такої мережі Фейстеля на 32-розрядних архітектурах може викликати складнощі, тому використовуються модифіковані варіанти цієї конструкції. У звичайних ситуаціях використовуються мережі з 4 гілками. На малюнку показано найбільш розповсюджені модифікації. Також існують схеми, в яких довжини половинок L_0; R_0 не збігаються, вони називаються незбалансованими.

Модифікації мережі Фейстеля

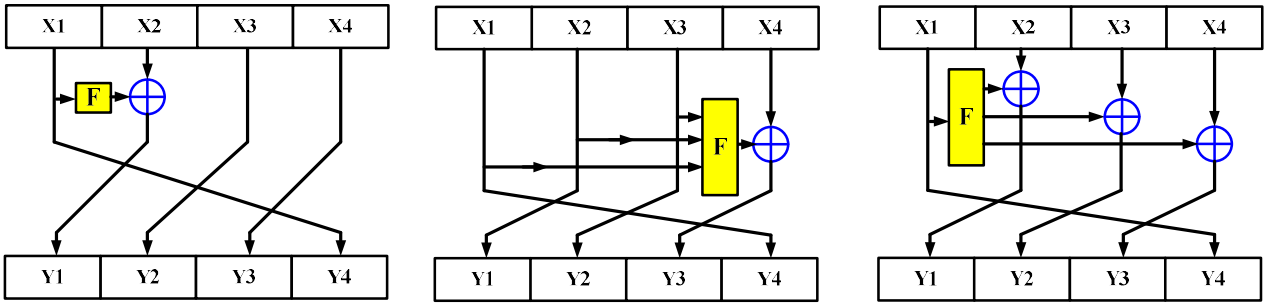


Рисунок 6.4 – Модифікована мережа Фейстеля

Порівняльний список алгоритмів на основі мережі Фейстеля

Наступні блокові шифри використовують класичну або модифіковану мережу Фейстеля в якості своєї основи [21]:

Алгоритм	Рік	Число раундів	Довжина ключа	Розмір блоку	Кількість підблоків
Blowfish	1993	16	від 32 до 448	64	2
Camellia	2000	18/24	128/192/256	128	2
CAST-128	1996	12/16	40-128	64	2
CAST-256	1998	12 4 = 48	128/192/256	128	2
CIPHERUNICORN-A	2000	16	128/192/256	128	2
CIPHERUNICORN-E	1998	16	128	64	2
CLEFIA	2007	16	128/192/256	128	16
DEAL	1998	6 (8)	(128/192) 256	128	2
DES	1977	16	56	64	2
DFC	1998	8	128/192/256	128	?
FEAL	1987	4-32	64	64	2
ГОСТ 28147-89	1989	32/16	256	64	2
IDEA	1991	8 +1	128	64	4
KASUMI	1999	8	128	64	2
Khufu	1990	16-32/64	512	64	2
LOKI97	1997	16	128/192/256	128	2
Lucifer	1971	16	48/64/128	48/32/128	2
MacGuffin	1994	32	128	64	4
MAGENTA	1998	6/8	128/192/256	128	2
MARS	1998	32	128-1248	128	2
Mercy	2000	6	128	4096	?
MISTY1	1995	4 n (8)	128	64	4
Raiden	2006	16	128	64	2
RC2	1987	16 +2	8-128	64	4
RC5	1994	1-255 (12)	0-2040 (128)	32/64/128	2
RC6	1998	20	128/192/256	128	4
RTEA	2007	48/64	128/256	64	2
SEED	1998	16	128	128	2
Sinople	2003	64	128	128	4
Skipjack	1998	23	80	64	4
TEA	1994	64	128	64	2
Triple DES	1978	32/48	112/168	64	2
Twofish	1998	16	128/192/256	128	4
XTEA	1997	64	128	64	2
XTEA-3	1999	64	256	128	4
XXTEA	1998	12-64	128	64	2

Структура, розроблена Х. Фейстелем, має низку переваг, а саме:

- процедури шифрування та розшифрування однакові, лише ключова інформація при розшифруванні подається в оберненому порядку;

- для розшифрування можна використовувати ті ж апаратні або програмні блоки, що й для шифрування, що значно зменшує вартість реалізації та робить її значно зручнішою у використанні.

Недолікоммережі Фейстеля вважають те, що в кожному циклі змінюється лише половина блоку вхідного тексту. Це призводить до збільшення кількості циклів для досягнення бажаної стійкості. Стосовно вибору *F*-функції, то чітких рекомендацій немає, проте найчастіше ця функція, яка повністю залежить від ключа, складається з нелінійних замін, перестановок і зсувів.

6.2 Шифр TEA

Шифр TEA (TinyEncryptionAlgorithm) – один із самих простих в реалізації, але стійких криптоалгоритмів [22]. Параметри алгоритму:

- Розмір блоку – 64 біта.
- Довжина ключа – 128 біт.
- В алгоритмі використана мережа Фейштеля з двома вітками в 32 біта кожна.
- Функція F оборотна.

Опис алгоритму

Оригінальний текст розбивається на блоки по 64 біта кожен. 128-бітний ключ *K* ділиться на чотири 32-бітних підключа K [0], K [1], K [2] і K [3]. На цьому підготовчий процес закінчується, після чого кожен 64-бітний блок шифрується протягом 32 циклів (64 раундів) по наведених нижче алгоритму [23].

Припустимо, що на вхід n-го раунду (для $1 \le n \le 64$) надходять права і ліва частина (L_n, R_n), тоді на виході *n*-го раунду будуть ліва і права частини (L_{n+1}, R_{n+1}), які обчислюються за такими правилами: $L_{n+1} = R_n$.

Якщо непарні раунди то:

$$R_{n+1} = L_n \boxplus (\{[R_n \ll 4] \boxplus K[0]\} \oplus \{R_n \boxplus i * \delta\} \oplus \{[R_n \gg 5] \boxplus K[1]\})$$

Якщо парні раунди то:

$$R_{n+1} = L_n \boxplus (\{[R_n \ll 4] \boxplus K[2]\} \oplus \{R_n \boxplus i * \delta\} \oplus \{[R_n \gg 5] \boxplus K[3]\})$$

- $X \boxplus Y$ - операція додавання чисел X і Y по модулю 2^{32} .
- $X \oplus Y$ - побітовое виключне «АБО» (XOR) чисел X і Y.

- X<< Y і X>>Y - операції побітового зсуву числа X на Y біт вліво і вправо відповідно.
- Константа δ була виведена з Золотого перетину $\delta = (\sqrt{5} - 1) * 2^{31} =$ 2654435769. У кожному раунді константа множиться *на номер циклу* ***i***. Це зроблено для запобігання простих атак, заснованих на симетрії раундів.
- Також очевидно, що в алгоритмі шифрування TEA немає як такого алгоритму розкладу ключів. Замість цього в непарних раундах використовуються підключи K [0] і K [1], в парних - K [2] і K [3].
- Так як це блоковий шифроалгоритм, де довжина блоку 64-біт, а довжина даних може бути не кратна 64-бітам, значення всіх байтів доповнюють блок до кратності в 64-біт встановлюється в 0x01 (Форма запису числа 1 в шіснадцятковій системі).

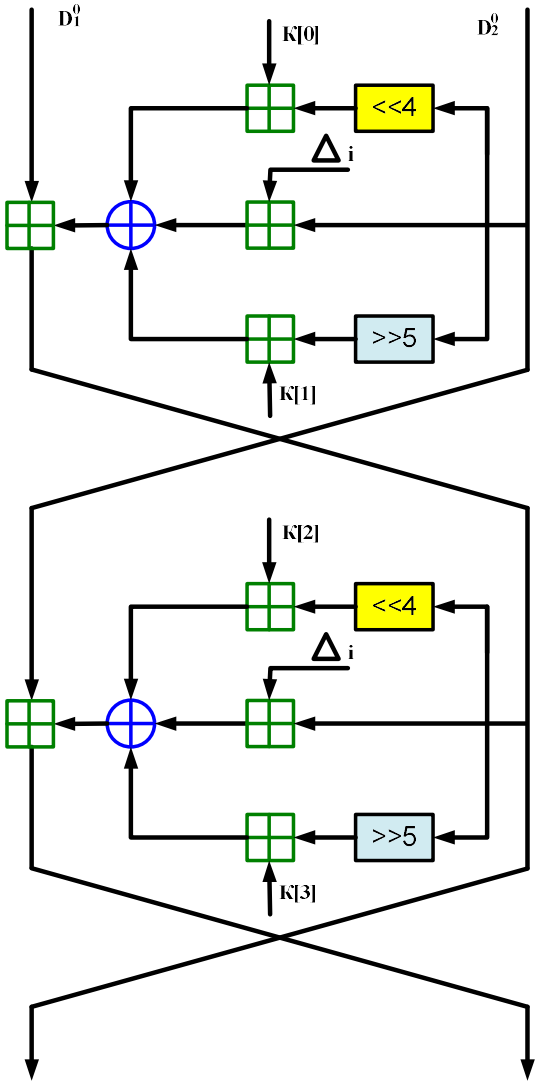


Рисунок 6.5 – Схема роботи алгоритму TEA

6.3Алгоритм шифрування IDEA

IDEA (International Data Encryption Algorithm - міжнародний алгоритм шифрування даних). (професор Джеймс Мессі) [24].

Основні характеристики і структура

Фундаментальним нововведенням в алгоритмі є використання операцій з різних алгебраїчних груп, а саме:

- Додавання по модулю 2^{16}
- Множення по модулю $2^{16}+1$
- Побітова виключна диз'юнкція (XOR).

Позначення операції:

⊞ - Додавання по модулю 2^{16}

⊙ - Множення по модулю $2^{16}+1$

⊕ - Побітова виключна дизюнуція (XOR)

Шифрування

Структура алгоритму IDEA показана на малюнку. Процес шифрування складається з восьми однакових раундів шифрування і одного вихідного перетворення. Вихідний незашифрований текст ділиться на блоки по 64 біта. Кожен такий блок ділиться на чотири підблока по 16 біт кожен. На малюнку ці підблоки позначені $D_1;D_2;D_3;D_4$.У кожному раунді використовуються свої підключі згідно з таблицею підключів. Над 16-бітними підключами і підблоками незашифрованого тексту проводяться наступні операції:

- Додавання по модулю ;
- Множення по модулю ;
- Побітова виключна диз'юнкція (XOR).

В кінці кожного раунду шифрування є чотири 16-бітних підблоки, які потім використовуються як вхідні підблоки для наступного раунду шифрування. Вихідна перетворення являє собою скорочений раунд, а саме, чотири 16-бітних підблоки на виході восьмого раунду і чотири відповідних підключа піддаються операціям:

- Додавання по модулю 2^{16}
- Множення по модулю $2^{16}+1$

Після виконання вихідного перетворення конкатенація під блоків $D_1'; D_2'; D_{3;}'D_4';$ і являє собою зашифрований текст. Потім береться наступний 64-бітний блок незашифрованого тексту і алгоритм шифрування повторюється. Так продовжується до тих пір, поки не зашифрують всі 64-бітові блоки вихідного тексту.

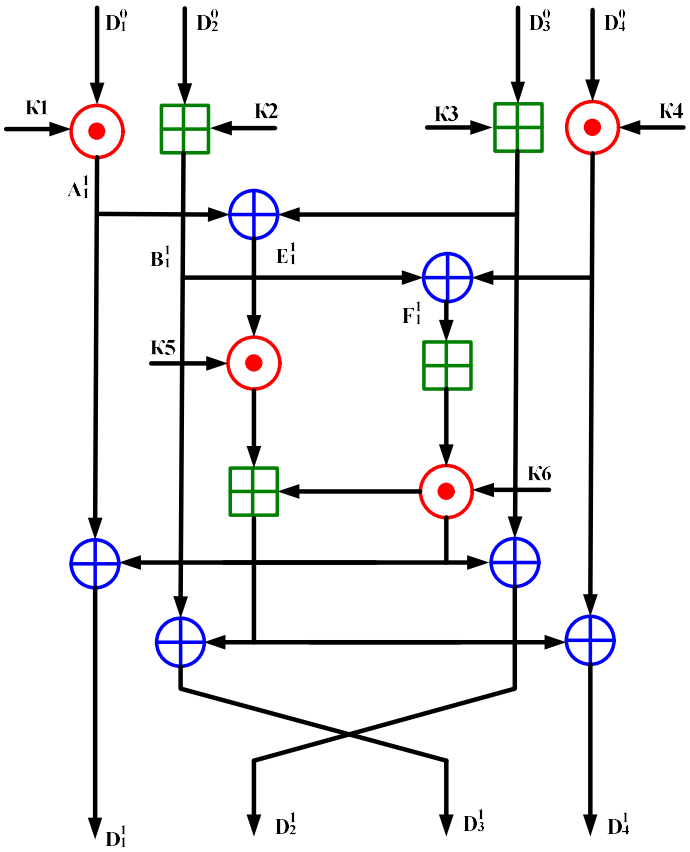


Рисунок 6.6 – Структура алгоритму IDEA.

Генерація ключів

З 128-бітного ключа для кожного з восьми раундів шифрування генерується по шість 16-бітних підключів, а для вихідного перетворення генерується чотири 16-бітних підключа. Всього буде потрібно 52 = 8 ×6 + 4 різних підключів по 16 біт кожен. Процес генерації п'ятдесяти двох 16-бітних ключів полягає в наступному:

- Насамперед, 128-бітний ключ розбивається на вісім 16-бітних блоків. Це будуть перші вісім підключів по 16 біт кожен — $K_1^{(1)}; K_2^{(1)}; K_3^{(1)}; K_4^{(1)}; K_5^{(1)}; K_6^{(1)}; K_1^{(2)}; K_2^{(2)};$
- Потім цей 128-бітний ключ циклічно зсувається вліво на 25 позицій, після чого новий 128-бітний блок знову розбивається на вісім 16-бітних блоків. Це вже наступні вісім подключів по 16 біт кожен — $K_3^{(2)}; K_4^{(2)}; K_5^{(2)}; K_6^{(2)}; K_1^{(3)}; K_2^{(3)}; K_3^{(3)}; K_4^{(3)};$
- Процедура циклічного зсуву і розбивки на блоки триває до тих пір, поки не будуть згенеровані всі 52 16-бітних підключа.

Таблиця 6.1 Підключі для кожного раунда шифрування

Номер раунда	Підключ
1	$K_1^{(1)}; K_2^{(1)}; K_3^{(1)}; K_4^{(1)}; K_5^{(1)}; K_6^{(1)};$
2	$K_1^{(2)}; K_2^{(2)}; K_3^{(2)}; K_4^{(2)}; K_5^{(2)}; K_6^{(2)};$
3	$K_1^{(3)}; K_2^{(3)}; K_3^{(3)}; K_4^{(3)}; K_5^{(3)}; K_6^{(3)};$
4	$K_1^{(4)}; K_2^{(4)}; K_3^{(4)}; K_4^{(4)}; K_5^{(4)}; K_6^{(4)};$
5	$K_1^{(5)}; K_2^{(5)}; K_3^{(5)}; K_4^{(5)}; K_5^{(5)}; K_6^{(5)};$
6	$K_1^{(6)}; K_2^{(6)}; K_3^{(6)}; K_4^{(6)}; K_5^{(6)}; K_6^{(6)};$
7	$K_1^{(7)}; K_2^{(7)}; K_3^{(7)}; K_4^{(7)}; K_5^{(7)}; K_6^{(7)};$
8	$K_1^{(8)}; K_2^{(8)}; K_3^{(8)}; K_4^{(8)}; K_5^{(8)}; K_6^{(8)};$
Вихідне перетворення	$K_1^{(9)}; K_2^{(9)}; K_3^{(9)}; K_4^{(9)};$

Математичний опис

- Блок відкритого тексту розміром 64 біт ділиться на чотири рівні підблока розміром по 16 біт $D_1^0;\ D_2^0;\ D_3^0;\ D_4^0$
- Для кожного раунду i=1….8 обчислюються:

$$A^{(i)} = D_1^{(i-1)} * K_1^{(i)}$$
$$B^{(i)} = D_2^{(i-1)} + K_2^{(i)}$$
$$C^{(i)} = D_3^{(i-1)} + K_3^{(i)}$$
$$D^{(i)} = D_4^{(i-1)} * K_4^{(i)}$$
$$E^{(i)} = A^{(i)} \oplus C^{(i)}$$
$$F^{(i)} = B^{(i)} \oplus D^{(i)}$$

$$D_1^{(i)} = A^{(i)} \oplus ((F^{(i)} + E^{(i)} * K_5^{(i)}) * K_6^{(i)})$$
$$D_2^{(i)} = C^{(i)} \oplus ((F^{(i)} + E^{(i)} * K_5^{(i)}) * K_6^{(i)})$$
$$D_3^{(i)} = B^{(i)} \oplus (E^{(i)} * K_5^{(i)} + (F^{(i)} + E^{(i)} * K_5^{(i)}) * K_6^{(i)})$$
$$D_4^{(i)} = D^{(i)} \oplus (E^{(i)} * K_5^{(i)} + (F^{(i)} + E^{(i)} * K_5^{(i)}) * K_6^{(i)})$$

- Результатом виконання восьми раундів будуть наступні чотири під блока $D_1^8;\ D_2^8;\ D_3^8; D_4^8$
- Виконується вихідне перетворення i=9:

$$D_1^{(9)} = D_1^{(8)} * K_1^{(9)}$$
$$D_2^{(9)} = D_3^{(8)} + K_2^{(9)}$$
$$D_3^{(9)} = D_2^{(8)} + K_3^{(9)}$$
$$D_4^{(9)} = D_4^{(8)} * K_4^{(9)}$$

Результатом виконання вихідного перетворення є зашифрований текст $D_1^9;\ D_2^9;\ D_3^9; D_4^9$

Розшифрування

Метод обчислення, що використовується для розшифровування тексту по суті такий же, як і при його шифруванні. Єдина відмінність полягає в тому, що для розшифровування використовуються інші підключі. У процесі розшифровування підключі повинні використовуватися у зворотному порядку. Перший і четвертий підключі і-го раунду розшифровування виходять з першого і четвертого підключа (10-і)-го раунду шифрування мультиплікативною інверсією. Для 1-го та 9-го раундів другий і третій підключи розшифровування виходять з другого і третього підключів 9-го і 1-го раундів шифрування адитивною інверсією. Для раундів з 2-го по 8-й другий і третій підключі розшифровування виходять з третього і другого підключів з 8-го по 2-й раундів шифрування адитивною інверсією. Останні два підключі *i*-го раунду розшифровки рівні останніх двох підключів (9-і)-го раунду шифрування. Мультиплікативна інверсія підключа K позначається $1/K_i$ і $(1/K)^* K \equiv 1 \bmod (2^{16}+1)$. Так як $2^{16}+1$ — простe число, кожне ціле не дорівнює нулю K має унікальну мультипликативну інверсію по модулю $2^{16}+1$. Адитивна інверсія підключа K позначається – - K і $-K+K \equiv 0 \bmod (2^{16}+1)$.

Таблиця 6.2 Підключі для кожного раунду розшифровування

Номер раунда	Підключ
1	$1/K_1^{(9)};\ -K_2^{(9)};\ -K_3^{(9)};\ 1/K_4^{(9)};\ K_5^{(8)};\ K_6^{(8)};$
2	$1/K_1^{(8)};\ -K_2^{(8)};\ -K_3^{(8)};\ 1/K_4^{(8)};\ K_5^{(7)};\ K_6^{(7)};$
3	$1/K_1^{(7)};\ -K_2^{(7)};\ -K_3^{(7)};\ 1/K_4^{(7)};\ K_5^{(6)};\ K_6^{(6)};$
4	$1/K_1^{(6)};\ -K_2^{(6)};\ -K_3^{(6)};\ 1/K_4^{(6)};\ K_5^{(5)};\ K_6^{(5)};$
5	$1/K_1^{(5)};\ -K_2^{(5)};\ -K_3^{(5)};\ 1/K_4^{(5)};\ K_5^{(4)};\ K_6^{(4)};$
6	$1/K_1^{(4)};\ -K_2^{(4)};\ -K_3^{(4)};\ 1/K_4^{(4)};\ K_5^{(3)};\ K_6^{(3)};$
7	$1/K_1^{(3)};\ -K_2^{(3)};\ -K_3^{(3)};\ 1/K_4^{(3)};\ K_5^{(2)};\ K_6^{(2)};$
8	$1/K_1^{(2)};\ -K_2^{(2)};\ -K_3^{(2)};\ 1/K_4^{(2)};\ K_5^{(1)};\ K_6^{(1)};$
Вихідне перетворення	$1/K_1^{(1)};\ -K_2^{(1)};\ -K_3^{(1)};\ 1/K_4^{(1)};$

6.4 Порядок виконання роботи

1. Розглянути основні теоретичні відомості блочних алгоритмів шифрування інформації основаних на мережі Фейстеля.

2. Проаналізувати наведені в додатку Д приклади програмного коду побудови алгоритму на основі мережі Фейстеля.

3. На підставі проведеного аналізу у відповідності з варіантом лабораторної роботи реалізувати систему шифрування повідомлення з урахування властивостей певного алгоритму.

4. На підставі проведеного аналізу у відповідності з варіантом лабораторної роботи розробити систему розшифрування шифр тексту з урахування властивостей певного алгоритму.

5. Реалізувати дані алгоритми у вигляді програмного коду.

6.5 Зміст звіту

1. Звіт з лабораторної роботи повинен бути виконаний на листах формату А4.

2. Звіт повинен містити: назву лабораторної роботи, її мету і короткі теоретичні відомості.

3. В розділі «Результати виконання лабораторної роботи» студент повинен представити у вигляду блок-схеми алгоритм шифрування та розшифрування блокового шифру у відповідності до заданого у завданні варіанту для заданої криптосистеми, а також навести код програмної реалізації.

4. Написати висновки до даної лабораторної роботи.

5. Оформити звіт та представити його викладачу на захист.

Таблиця 6.3 - Варіанти завдань

№ вар.n	Функція $f(K_i)$	ключ (K_i) i – номер раунду	Розмір блоку S, біт
1.	Сума по модулю K_i	$2n+imodS$	16
2.	>> Зсув вправо на K_i	$2n+imodS$	32
3.	<<Зсув вліво на K_i	$2n+imodS$	64
4.	Добуток по модулю два	$2n+imodS$	16
5.	Сума по модулю K_i	$2n+imodS$	32
6.	>> Зсув вправо на K_i	$2n+imodS$	64
7.	<<Зсув вліво на K_i	$2n+imodS$	16
8.	Добуток по модулю два	$2n+imodS$	32
9.	Сума по модулю K_i	$2n+imodS$	64
10.	>> Зсув вправо на K_i	$2n+imodS$	16
11.	<<Зсув вліво на K_i	$2n+imodS$	32
12.	Добуток по модулю два	$2n+imodS$	64
13.	Сума по модулю K_i	$2n+imodS$	16
14.	>> Зсув вправо на K_i	$2n+imodS$	32
15.	<<Зсув вліво на K_i	$2n+imodS$	64

Продовження таблиці 6.3

16.	Добуток по модулю два	$2n+imodS$	16
17.	Сума по модулю K_i	$2n+imodS$	32
18.	>> Зсув вправо на K_i	$2n+imodS$	64
19.	<<Зсув вліво на K_i	$2n+imodS$	16
20.	Добуток по модулю два	$2n+imodS$	32
21.	Сума по модулю K_i	$2n+imodS$	64
22.	>> Зсув вправо на K_i	$2n+imodS$	16
23.	<<Зсув вліво на K_i	$2n+imodS$	32
24.	Добуток по модулю два	$2n+imodS$	64
25.	Сума по модулю K_i	$2n+imodS$	16
26.	>> Зсув вправо на K_i	$2n+imodS$	32
27.	<<Зсув вліво на K_i	$2n+imodS$	64
28.	Добуток по модулю два	$2n+imodS$	16
29.	Сума по модулю K_i	$2n+imodS$	32
30.	>> Зсув вправо на K_i	$2n+imodS$	64

Контрольні запитання

1. Які переваги мають блокові шифри?
2. Яке призначення режимів шифрування?
3. Що таке раунд?
4. Якими перевагами володіють шифри основані на мережі Фейтстеля?
5. Які недоліки притаманні шифрам основаним на мережі Фейтстеля?
6. Які недоліки мають блокові шифри?
7. Що таке операція «padding» у блокових крипто алгоритмах?

Додаток Е

```
#include<iostream>
#include<string>
#include<iomanip>
#include<vector>
#include<climits>

usingnamespacestd;

voidprintBytes(vector<string>v_s)
{
    for(int i =0; i <v_s.size(); i++)
    {
        constchar* c = v_s.at(i).c_str();
```

```
        int* left =(int*)c;
        int* right = left+1;

        cout  <<*left <<" "<<*right <<" ";
    }
    cout<<endl;
}

int f(intsubblock, char key)
{
    returnsubblock^key;
}

int* decrypt(int*left, int*right, longlong*p_key)
{
    int rounds =8;
    for(int i = rounds -1; i >=0; i--)
    {
        int temp =*left ^ f(*right, *(char*)p_key+i);
        if(i !=7)
        {
            *left =*right;
            *right = temp;
        }
        else*right = temp;

        }

    return left;
}

int* encrypt(int*left, int*right, longlong*p_key)
{
    int rounds =8;
    for(int i =0; i < rounds; i++)
    {
        int temp =*right ^ f(*left, *(char*)p_key+i);

        if(i !=7)
        {
            *right =*left;
            *left = temp;
        }
        else*right = temp;
```

```
        //cout<< i+1 << "th iteration:" <<endl<< *left << " "
<< *right <<endl<<endl;
    }

    return left;
}

int main()
{
    string message;
    vector<string>v_s;

    cout<<"Please enter message to encrypt:"<<endl<<endl;
    getline(cin, message);

    int length =message.length();
    int quotient = length/8;
    if(length%8!=0) quotient++;

    for(int i =0; i < quotient; i++)
    {
        string new_message=message.substr(0, 8);
        v_s.push_back(new_message);

        for(int i =0; i <8; i++)
        {
            message.erase(0, 1);
        }
    }

    string last_string= v_s.at(v_s.size()-1);
    if(last_string.length()<8)last_string.resize(8, '0');

    longlong key =INT_MAX/0x100-0x1000;
    longlong*p_key=&key;

    cout<<endl<<"This message by
bytes:"<<endl<<endl<<setbase(16);
    printBytes(v_s);

    for(int i =0; i <v_s.size(); i++)
    {
        constchar* c = v_s.at(i).c_str();

        int* left =(int*)c;
        int* right = left+1;
```

```
        c =(char*)encrypt(left, right, p_key);
    }

    cout<<endl<<"Encrypted message by bytes:"<<endl<<endl;
    printBytes(v_s);

    cout<<endl<<"Encrypted message:"<<endl<<endl;
    for(int i =0; i <v_s.size(); i++)
    {
        constchar* c = v_s.at(i).c_str();
        cout  << c;
    }
    cout<<endl;

    for(int i =0; i <v_s.size(); i++)
    {
        constchar* c = v_s.at(i).c_str();

        int* left =(int*)c;
        int* right = left+1;

        c =(char*)decrypt(left, right, p_key);
    }

    cout<<endl<<endl<<"Decrypted message by
bytes:"<<endl<<endl;
    printBytes(v_s);

    cout<<endl<<"Decrypted message:"<<endl<<endl;
    for(int i =0; i <v_s.size(); i++)
    {
        constchar* c = v_s.at(i).c_str();
        cout  << c;
    }
    cout<<endl<<endl;

    system("PAUSE");
    return0;
     }
```

ЛАБОРАТОРНА РОБОТА №7
Дослідження крипто стійкості паролів

Мета роботи: дослідити крипто стійкість паролів різного ступеню складності з використанням методу прямого перебору

7. 1 Короткі теоретичні відомості

Інформація з погляду інформаційної безпеки володіє наступними категоріями:

конфіденційність – гарантія того, що конкретна інформація доступна тільки тому колу осіб, для кого вона призначена; порушення цієї категорії називається розкраданням або розкриттям інформації;

цілісність – гарантія того, що інформація зараз існує в її початковому вигляді, тобто при її зберіганні або передачі не було проведено несанкціонованих змін; порушення цієї категорії називається фальсифікацією повідомлення;

аутентичність – гарантія того, що джерелом інформації є саме та особа, яка заявлена як її автор; порушення цієї категорії також називається фальсифікацією, але вже автора повідомлення;

апелюємість – досить складна категорія, але часто вживана в електронній комерції – гарантія того, що при необхідності можна буде довести, що автором повідомлення є саме заявлена людина, і не може бути ніхто інший; відмінність цієї категорії від попередньої в тому, що при підміні автора, хтось інший намагається заявити, що він автор повідомлення, а при порушенні апелюємості – сам автор намагається "відхреститися" від своїх слів, сказаних ним одного разу.

Відносно інформаційних систем застосовуються інші категорії :

- надійність – гарантія того, що система поводиться в нормальному і позаштатному режимах так, як заплановано;
- точність – гарантія точного і повного виконання всіх команд;
- контроль доступу – гарантія того, що різні групи осіб мають різний доступу до інформаційних об'єктів, і ці обмеження доступу постійно виконуються;
- контрольованість – гарантія того, що у будь-який момент може бути проведена повноцінна перевірка будь-якого компоненту програмного комплексу;
- контроль ідентифікації – гарантія того, що клієнт, підключений в даний момент до системи, є саме тим, за кого себе видає;

- стійкість до навмисних збоїв – гарантія того, що при навмисному внесенні помилок в межах наперед обумовлених норм система поводитиметься так, як обумовлено наперед.

Кожен користувач, перш ніж здійснювати які-небудь дії в комп'ютерній системі, повинен ідентифікувати себе. У свою чергу, система повинна перевірити достовірність особи користувача, тобто що він є саме тим, за кого себе видає.

Стандартним засобом перевірки достовірності (аутентифікації) – пароль, хоча у принципі можуть використовуватися також особисті картки, біометричні пристрої (сканування рогівки ока або відбитків пальців) або їх комбінація.

Перебір паролів по словнику був якийсь час однією з найпоширенішої техніки підбору паролів. Нині, як хоч найменший результат пропаганди інформаційної безпеки, він став складати свої позиції. Хоча розвиток швидкодії обчислювальної техніки і все більш складні алгоритми складання слів-паролів не дають "загинути" цьому методу. Технологія перебору паролів народилася в той час, коли найскладнішим паролем було скажемо слово "brilliant", а в русифікованих ПК воно ж, але для "хитрості" набране в латинському режимі, але дивлячись на російські літери (ця тактика на жаль до цих пір надзвичайно поширена, хоч і збільшує інформаційну насиченість пароля всього на 1 біт). У той час простенька програма із словником в 5000 іменників давала позитивний результат в 60% випадків. Величезне число інцидентів із зламуваннями систем примусило користувачів додавати до слів 1-2 цифри з кінця, записувати першу і/або останню літеру у верхньому регістрі, але це збільшило час на перебір варіантів з врахуванням зростання швидкодії ПК всього у декілька разів.

Існує ціла область знань, що розглядає питання аутентифікації, і зокрема, проблеми пов'язані з паролями. Наприклад, обговорюються необхідність і періодичність зміни паролів користувачами, вимоги до забезпечення секретності паролів, необхідність використання згенерованих програмними засобами (а не вибраних вручну) паролів, використання комбінованих паролів (звичайного введення символів плюс біометричні технології або введення смарт - карт), необхідність надання користувачу деякої реєстраційної інформації (дати і часу останнього входу в систему і т.п.) і ін. У цій лабораторній роботі не розглядатимуться методи отримання паролів на основі методів соціальної інженерії, які спрямовані на здобування паролів або інформації, що дозволяє отримати пароль методом відмінним від методу прямого перебору. Однак, безумовно, однією з основних вимог є достатня крипто стійкість паролів, що не дозволяє системам зламування паролів за відносно невеликий час зламати пароль.

Крипто стійкість паролів визначається кількістю символів пароля (що звичайно вводяться з клавіатури), використовуваними наборами символів і кількістю наборів. Можна виділити наступні набори символів:

- заголовні (прописні) латинські літери;
- рядкові латинські літери;
- цифри;
- спеціальні символи, розділові (наприклад № % « !) знаки.

Чим більше символів містить пароль, і чим більша кількість використовуваних наборів символів, тим довше виконуватиметься злом пароля і тим вище крипто стійкість пароля.

При створенні пароля необхідно враховувати, що системи злому паролів шляхом перебору часто використовують словники, що містять слова тієї або іншої мови. Тому використання паролів у вигляді слів природної мови знижує їх криптостійкість. Крім того, криптостійкій пароль, створений з урахуванням приведених вище рекомендацій, складно запам'ятати, і тому велика спокуса його фіксації на папері, що в свою чергу, різко підвищує ймовірність розкрадання пароля і реалізацію несанкціонованого доступу. Для створення одночасно і криптостійкого пароля, що запам'ятовується, рекомендується:

- набирати пароль на іншій розкладці клавіатури;
- використовувати замість літер інші схожі спеціальні символи, наприклад замість S - $, замість l – 1, замість K - |< і т.д.

Наступною модифікацією підбору паролів є перевірка паролів, що встановлюються в системах за умовчанням. В деяких випадках адміністратор програмного забезпечення, встановивши або одержавши новий продукт від розробника, не спромагається перевірити, з чого складається система безпеки. Як наслідок, пароль, встановлений у фірмі розробнику за замовчуванням, залишається основним паролем в системі. У мережі можна знайти величезні списки паролів за замовчуванням практично до всіх версій програмного забезпечення, якщо вони встановлюються на ньому виробником.

Основні вимоги до інформаційної безпеки, засновані на аналізі даного методу, наступні:

Вхід всіх користувачів в систему повинен підтверджуватися введенням унікального для клієнта пароля.

Пароль повинен ретельно підбиратися так, щоб його інформаційна ємкість відповідала часу повного перебору пароля. Для цього необхідно детально інструктувати клієнтів про поняття "простий до підбору пароль", або передати операцію вибору пароля у ведення інженера по безпеці.

Паролі за замовчуванням повинні бути змінені до офіційного запуску системи і навіть до прилюдних випробувань програмного комплексу. Особливе це відноситься до мережевого програмного забезпечення.

Всі помилкові спроби увійти до системи повинні враховуватися, занотовуватися у файл журналу подій і аналізуватися через "розумний" проміжок часу. Якщо в системі передбачена можливість блокування клієнта або всієї системи після певної кількості невдалих спроб входу, цією можливістю необхідно скористатися. Така можливість є основним бар'єром до підбору паролів повним перебором. Розумно блокувати клієнта при третій підряд неправильній спробі набору пароля, і, відповідно, блокувати систему $K = \max(\text{int}\,(N * 0.1 * 3) + 1.3)$ невдалих спроб входу за деякий період (годину, зміну, добу). У цій формулі N – середня кількість тих клієнтів, що підключаються за цей період до системи, 0.1 – 10% межа "забудькуватості пароля", 3 – ті ж самі три спроби на створення пароля. Інформація про блокування клієнта або системи повинна автоматично надходити на пульт контролю за системою.

У момент відправки пакету підтвердження або відторгнення пароля в системі повинна бути встановлена розумна затримка (2-5 секунд). Це не дозволить зловмиснику, потрапивши на лінію з добрим зв'язком до об'єкту атаки перебирати по сотні тисяч паролів за секунду.

Всі дійсні в системі паролі бажано перевіряти сучасними програмами підбору паролів, або оцінювати особисто адміністратору системи.

Через певні проміжки часу необхідна примусова зміна пароля у клієнтів. Найбільш часто використовуваними інтервалами зміни пароля є рік, місяць і тиждень (залежно від рівня конфіденційності інформації і частоти входу в систему).

Всі невживані протягом довгого часу імена реєстрації повинні переводитися в закритий (недоступне для реєстрації) стан. Це відноситься до співробітників, що знаходяться у відпустці, хворіють, у відрядженні, а також до імен реєстрації, створених для тестів, випробувань системи і т.п.

Від співробітників і всіх операторів терміналу необхідно вимагати суворе нерозголошення паролів, відсутність яких-небудь взаємозв'язків пароля з широковідомими фактами і даними, і відсутність паперових записів пароля "із-за поганої пам'яті".

Наступною по частоті використання є методика отримання паролів з самої системи. Проте тут вже немає можливості дати які-небудь загальні рекомендації, оскільки всі методи атаки залежать тільки від програмної і апаратної реалізації конкретної системи. Основними двома можливостями з'ясування пароля є несанкціонований доступ до носія, що містить їх, або використання не документованих можливостей і помилок в реалізації системи.

Отримання доступу до паролів завдяки не документованим можливостям систем зустрічається в даний час вкрай рідко. Раніше ця методика використовувалася розробниками набагато частіше в основному в цілях відладки, або для екстреного відновлення працездатності системи. Але поступово з розвитком, як технологій зворотної компіляції, так і інформаційної зв'язаності світу вона поступово стала зникати. Будь-які не документовані можливості рано чи пізно стають відомими, після чого новина про це із великою швидкістю облітає світ і розробникам доводиться розсилати всім користувачам скомпрометованої системи "програмні латочки" або нові версії програмного продукту. Єдиною мірою профілактики цього методу є постійний пошук на серверах, присвячених комп'ютерній безпеці, оголошень про всі неприємності з програмним забезпеченням. Для розробників же необхідно пам'ятати, що будь-яка подібна вбудована можливість може на порядок знизити загальну безпеку системи, як би добре вона не була завуальована в коді програмного продукту.

Наступною поширеною технологією отримання паролів є копіювання буфера клавіатури у момент набору пароля на терміналі. Цей метод використовується рідко, так як для нього необхідний доступ до термінального комп’ютера з можливістю запуску програм. Але якщо зловмисник все-таки отримує подібний доступ, дієвість даного методу дуже висока:

Робота програми-перехоплювача паролів (так званого "троянського коня") на робочій станції непомітна.

Подібна програма сама може відправляти результати роботи на заздалегідь задані сервера або анонімним користувачам, що різко спрощує саму процедуру отримання паролів третьою стороною, і ускладнює пошук і доказ його провини. Наприклад, широкого поширення набула подібна троянська програма, що прописується в архіви, що само розпаковуються.

Двома основними методами боротьби з копіюванням паролів є:

Адекватний захист робочих станцій від запуску невідомих програм:

- відключення змінних носіїв інформації (гнучких дисків);
- спеціальні драйвера, блокуючи запуск здійснимих файлів без відома оператора, або адміністратора;
- монітори, що повідомляють про будь-які зміни системних настройок і списку програм, що автоматично запускаються;

Дієва, але незручна міра – система одноразових паролів (при кожній реєстрації в системі клієнтам з дуже високим рівнем відповідальності самою системою генерується новий пароль).

Сканування сучасними антивірусними програмами також може допомогти у виявленні "троянських" програм, але тільки тих з них, які

набули широкого поширення по країні. А отже, програми, написані зловмисниками спеціально для атаки на Вашу систему, будуть пропущені антивірусними програмами без яких-небудь сигналів.

Наступний метод отримання паролів відноситься тільки до мережевого програмного забезпечення. Проблема полягає в тому, що в багатьох програмах не враховується можливість перехоплення будь-якої інформації, що йде по мережі – так званого мережевого трафіку. Спочатку, з впровадженням локальних комп'ютерних мереж так воно і було. Мережа розташовувалася в межах 2-3 кабінетів, або будівлі з обмеженим фізичним доступом до кабелів. Проте, стрімкий розвиток глобальних мереж спрямував на спільний ринок ті ж версії програмного забезпечення без якого-небудь зволікання для посилення безпеки. Більше половини протоколів мережі Інтернет передають паролі в нешифрованому вигляді – відкритим текстом. До ним відносяться протоколи передачі електронної пошти SMTP і POP3, протокол передачі файлів FTP, одна з схем авторизації на WWW-серверах.

Сучасне апаратне і програмне забезпечення дозволяє одержувати всю інформацію, що проходить по сегменту мережі, до якого підключений конкретний комп'ютер, і аналізувати її в реальному масштабі часу.

Можливі декілька варіантів прослуховування трафіку: 1) це може зробити службовець компанії з свого робочого комп'ютера, 2) зловмисник, що підключився до сегменту за допомогою ноутбука або смартфона. Нарешті, трафік, що йде по мережі, технічно може прослуховуватися з боку безпосереднього провайдера, з боку будь-якої організації, що надає транспортні послуги для мережі Інтернет (листування усередині країни в середньому йде через 3-4 компанії, за межі країни – через 5-8).

Для комплексного захисту від подібної можливості викрадення паролів необхідно виконувати наступні заходи:

- фізичний доступ до мережевих кабелів повинен відповідати рівню доступу до інформації.
- при визначенні топології мережі слід при будь-якій нагоді уникати широкомовних топологій. Оптимальною одиницею сегментації є група операторів з рівними правами доступу, або якщо ця група складає більше 10 чоловік, то кімната або відділ усередині групи. У жодному випадку на одному кабелі не повинні знаходитися оператори з різними рівнями доступу, якщо тільки весь трафік не шифрується, а ідентифікація не проводиться по прихованій схемі без відкритої передачі пароля.

До всіх інформаційних потоків, що виходять за межі фірми, повинні застосовуватися ті ж правила, що описані вище для об'єднання різнорівневих терміналів.

Програма Elcomsoft Advanced Archive Password Recovery (рис. 7.1 – 7.2) є однією з програм розроблених компанією Elcom Ltd і призначених для відновлення пароля архівів, створених з використанням популярного архіватора ZIP [25]. Програма має можливість гнучкої настройки з вказівкою довжини пароля, використовуваних наборів символів (включаючи набір користувача), застосування маски (коли відомі деякі з символів пароля і їх місце) і підключення словників.

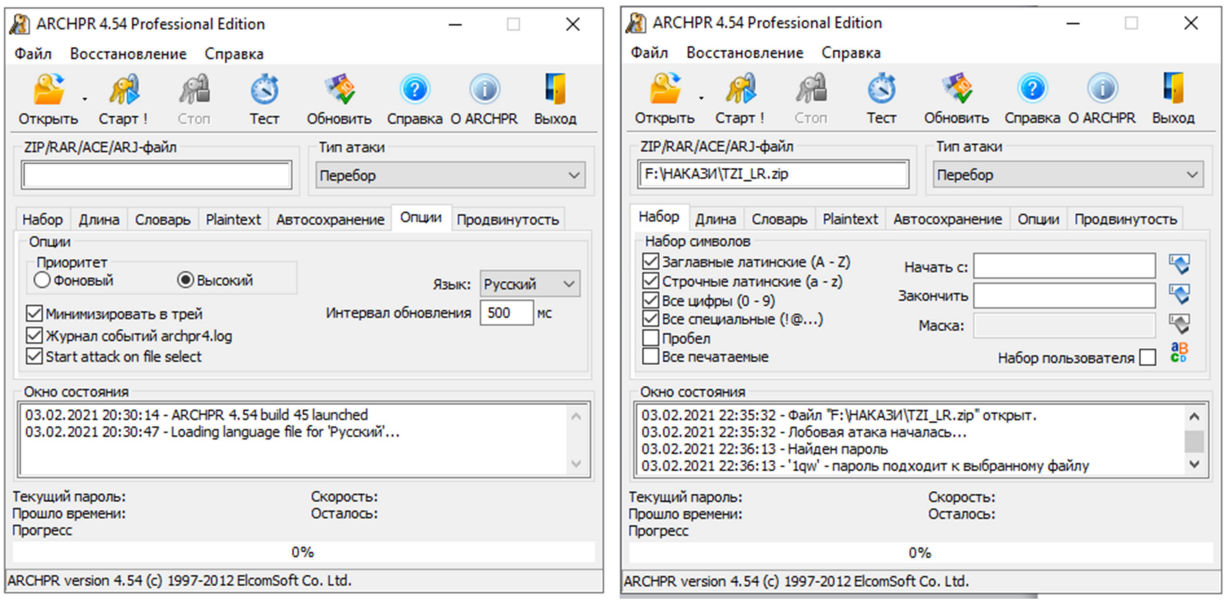


Рисунок 7.1 – Інтерфейс програми Elcomsoft Advanced Archive Password Recovery

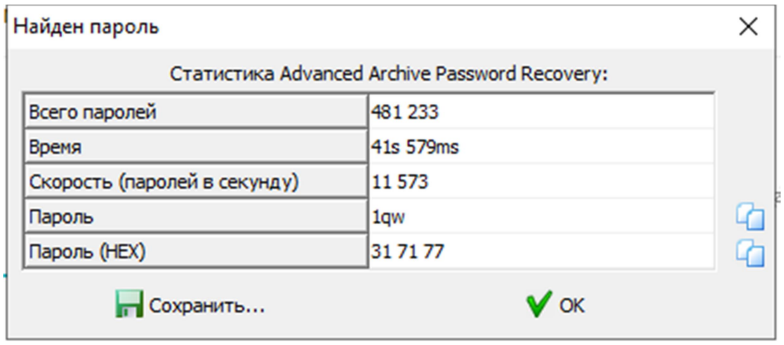


Рисунок 7.2 – Вікно з підібраним паролем і параметрами, що характеризують процес підбору

В лабораторній роботі пропонується використовувати програму Elcomsoft Advanced Archive Password Recovery для дослідження криптостійкості паролів з різною кількістю і різними наборами символів.

7.2 Порядок виконання роботи

1. Встановити програму Elcomsoft Advanced Archive Password Recovery в ознайомлювальному режимі (термін роботи продукту 30 днів).

2. За допомогою програми архівації даних створіть архівний файл і встановіть на нього пароль завдовжки 2 рядкових латинських символу.

3. Відкрити створений файл в програмі Elcomsoft Advanced Archive Password Recovery і вкажіть використовуваний набір – рядкові латинські символи, мінімальну довжину пароля, максимальну довжину пароля і виконаєте перебір. Зафіксуйте кількість перебраних паролів і час перебору в таблиці 7.1.

Таблиця 7.1 – Результати перебору паролів в програмі Elcomsoft Advanced Archive Password Recovery

№ п/п	Набір символів	Довжина пароля, символів	Кількість перебраних паролів	Час перебору, с	Швидкість перебору, паролів/с

4. Повторіть дослідження архівного файлу захищеного паролями довжиною 3, 4, 5 і 6 рядкових латинських символу і занесіть результати в табл. 7.1.

5. Повторіть дослідження архівного файлу у відповідності з пунктами 1-4 для комбінації з наборів рядкових і заголовних латинських символів і занесіть результати в табл. 7.1.

6. Повторіть дослідження архівного файлу у відповідності з пунктами 1-4 для комбінації з наборів рядкових, заголовних латинських символів і цифр, виконайте дослідження криптостійкості і занесіть результати в табл. 7.1.

7. Повторіть дослідження архівного файлу у відповідності з пунктами 1-4 для комбінації з наборів рядкових, заголовних латинських символів, цифр і спеціальних символів, виконайте дослідження криптостійкості і занесіть результати в табл. 7.1.

8. Побудуйте два графіки: залежність кількості перебраних паролів від довжини пароля для різних використаних наборів символів і залежність часу підбору пароля від довжини пароля для різних використаних наборів символів.

7.3 Контрольні питання

1. Що називають аутентифікацією і ідентифікацією користувачів?
2. Які способи підвищення криптостійкості і надійності паролів надаються сучасними операційними системами?
3. Чим визначається криптостійкість пароля?
4. Як реалізується метод повного перебору значень пароля?
5. Які існує комбіновані методи аутентифікації.

Лабораторна робота № 8

Дослідження асиметричних крипто алгоритмів шифрування інформації з відкритим ключем на основі алгоритму RSA та алгоритму обміну ключами Діффі-Хелмана

Мета роботи:дослідити асиметричні криптоалгоритми шифрування інформації з відкритим ключем на основі алгоритму RSA, та алгоритм обміну ключами Діффі-Хелмана, реалізувати один із наведених варіантів алгоритму у відповідності з завданням.

8. Асиметричні криптосистеми

Одним з найефективніших способів подолання недоліків симетричних криптосистем стало винайдення асиметричних способів шифрування, коли для зашифровування і розшифрування використовуються різні криптографічні ключі [26].

Крипостійкість таких систем ґрунтується на так званих "***односторонніх функціях***", які легко обчислюються в прямому напрямі та утворюють математичну проблему надзвичайної обчислювальної складності при спробі розв'язку оберненої задачі.

Як правило, для зашифровування інформації використовують так званий ***відкритий*** або ***публічний ключ*** (хоча в деяких випадках його можна використати і для розшифрування), який непотрібно приховувати. Навпаки, цей ключ розміщують на ресурсі, доступному для всіх учасників інформаційного обміну та захищають від підміни. Кожен, хто хоче написати конфіденційного листа власникові публічного ключа, використає його для зашифровування, а власник ключа, маючи парний до цього публічного, ***приватний*** (або ***секретний***) ключ, зможе розшифрувати повідомлення.

Особливістю такої криптосистеми є те, що за допомогою публічного ключа неможливо розшифрувати зашифроване на ньому повідомлення. Таким чином, ніхто, окрім власника приватного ключа, не зможе прочитати призначене йому повідомлення. Приватний ключ непотрібно розповсюджувати, він зберігається після генерування у його власника і використовується лише для розшифрування.

Отже, процес такого інформаційного обміну схематично можна зобразити таким чином (рис. 8.1):

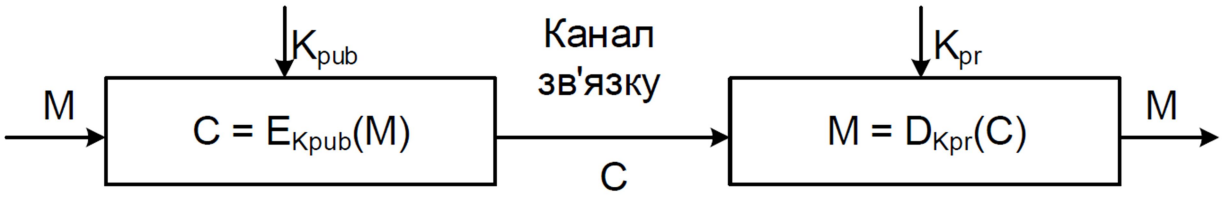


Рисунок 8.1 – Схема роботи асиметричної криптосистеми

Криптосистеми, в яких для зашифровування та розшифрування повідомлень використовуються різні криптографічні ключі, називаються асиметричними.

Асиметричні криптосистеми вирішили основну проблему симетричних криптосистем, проблему розповсюдження криптографічних ключів, адже для зашифровування інформації багатьма користувачами потрібно мати лише одну пару ключів: публічний та парний йому приватний.

8.1 Алгоритм шифрування RSA

Найпершою криптосистемою з відкритим ключем із запропонованих у відкритій літературі (1978р.), була система Райвест, Шаміра і Едлмана. Вона стала відома під назвою RSA [27]. Схема RSA набула набільшого визнання і реалізована практично у всіх методах шифрування з відкритим ключем. RSA є блоковий шифр, в якому відкритий і шифрований текст представляється цілими числами в діапазоні від 0 до n-1 для деякого n.

Генерація ключів

Для того, щоб згенерувати пари ключів виконуються такі дії:

1. вибираються два великих простих числа ***p*** і ***q.***

Просте число - натуральне (ціле додатне) число, що має рівно два різних натуральних дільника – одиницю і себе. Іншими словами число***x*** є простим, якщо воно більше 1 і при цьому ділиться без залишку тільки на 1 і на ***x***.

Послідовність простих чисел до 100 починається так: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59,61, 67, 71, 73, 79, 83, 89, 97….

2. обчислюється їх добуток $n=p\times q$

3. обчислюється Функція Ейлера $\varphi(n)=(p-1)(q-1)$

4. вибирається ціле число ***e*** таке, що $1<e<\varphi(n)$ та ***e*** взаємно просте з $\varphi(n)$.

Взаємно прості числа — натуральні або цілі числа, які не мають спільних дільників більших за 1, або, інакше кажучи, якщо їх найбільший

спільний дільник дорівнює 1. Таким чином, 2 і 3 — взаємно прості, а 2 і 4 — ні (діляться на 2).

5. за допомогою розширеного алгоритма Евкліда знаходиться число ***d*** таке, що ***e×d≡1(modφ(n)).***

Число ***n*** називається модулем, а числа ***e*** і ***d*** — відкритою й секретною експонентами, відповідно. Пари чисел ***(n, e)*** є відкритою частиною ключа, а ***(n,d)*** — секретною. Числа ***p*** і ***q*** після генерації пари ключів можуть бути знищені, але в жодному разі не повинні бути розкриті.

РОЗГЛЯНЕМО ЯК ПРАЦЮЄ ЦЕЙ АЛГОРИТМ.

Вибір p, q. Щоб зашифрувати і розшифрувати щось, необхідна пара ключів - відкритий і закритий. Для того щоб їх виготовити, знадобиться пара простих чисел p і q. *Простими називаються числа, які діляться без залишку тільки на себе і на одиницю.* Число 11 - просте, а 4 - ні, так як ділиться крім одиниці і четвірки ще й на 2. Отже, вибираємо пару простих чисел p=3, q=7.

Обчислення n . Наступний етап у виготовленні ключів - отримання числа n , рівного добутку p і q : n = p×q =3×7=21. Це число називають модулем порівняння при шифруванні і розшифровці.

Обчислення ***f(n)***. Тепер потрібно обчислити величину ***f(n)***, так звану функцією Ейлера за формулою: (p -1)×(q-1)=2×6=12. Це число не просте, і його в нашому випадку можна розкласти на прості множники ***f(n)***=12= 2×2×3.

Вибір e. Наступний крок - підбір числа ***e***, яке повинно відповідати двом критеріям: бути менше ***n*** і **НЕ МАТИ** загальних множників з ***f(n)***, тобто в розкладанні на прості множники числа ***e*** не повинно бути ні двійки, ні трійки. Цим вимогам задовольняє число ***5*** - воно менше ***n***, і до того ж просте, тобто ні на які співмножники не розкладається. Отже, ***e=5***.

Обчислення d. Треба знайти число ***d*** таке, що:

$$e \times d \equiv 1 \bmod \varphi(n)$$

$$5 \times d \equiv 1 \bmod 12$$

Для цього потрібно вибрати таке число ***d*** щоб ***e×d-1*** ділиться ***f(n)***. Без остатку:

якщо d=15, тобто e×d-1=5×15-1=70, 70/12=5,83 - тобто націло ***НЕ*** ділиться.

якщо d=17, тобто e×d-1=5×17-1= 84, а 84/12 = 7 - тобто дійсно ділиться.

Отже умова $5 \times 17 \equiv 1 \bmod 12$ виконується.

$$5 \times 17 = 85 \bmod 12 = 1$$

Відкритий і закритий ключ. Тепер у нас є і відкритий ключ, пара чисел ***(n,e) - (21,5),*** і пара чисел ***(n,d) - (21,17)*** - закритий ключ.

Відкритий текст.

Далі припустимо, що ця пара належить абоненту А. Абонент Б, володіючи відкритим ключем абонента А, може послати йому в зашифрованому вигляді кількість стукотів в двері - число 3.

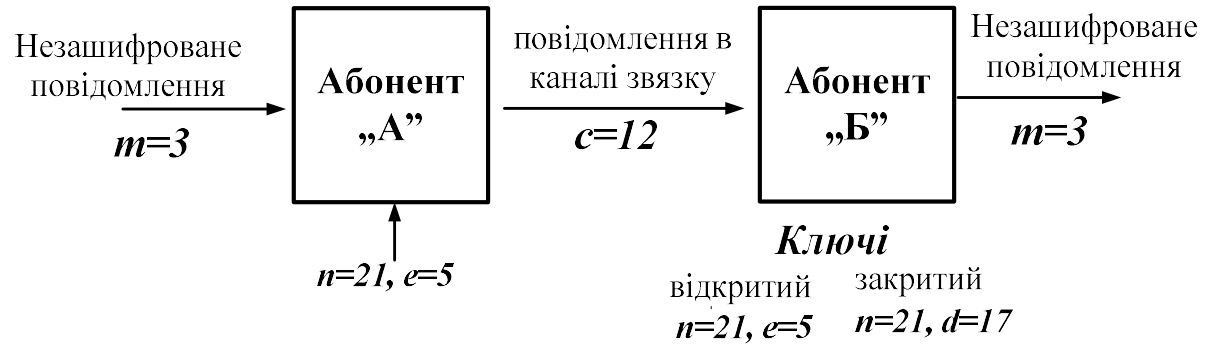


Рисунок 8.2 – Схема використання алгоритму RSA

Шифрування. Нехай відкрите повідомлення це пароль до сейфу число m=3. Щоб зашифрувати це число, ***абонент А*** повинен виконати обчислення згідно з формулою:

$$c=m^e \bmod n,$$

де ***m*** - повідомлення, ***(n,e)*** - відкритий ключ абонента **Б**, ***c*** - зашифроване повідомлення.

У нашому випадку $c=m^e=3^5=243$. Тоді $c=243 \bmod 21$. Це є залишок від ділення 243 на 21 і рівний ***12***. Отже, зашифроване число пароль сейфу складає c=12. Його й відсилаємо абоненту ***Б***.

Розшифровування. Для розшифрування повідомлення абонент Б використовує формулу:

$$m= c^d \bmod n,$$

де ***c*** – отримане зашифроване повідомлення, ***m***- розшифроване повідомлення, ***(n,d)*** - особистий ключ абонента Б ***(21,17)***.

Щоб розшифрувати повідомлення, доведеться звести ***c*** в степінь 17.

$$12^{17} = 2218611106740436992.$$

Колосальна кількість, але залишок від ділення (тобто mod 21) 2218611106740436992 на 21 дорівнює 3 – абонент Б розшифрував повідомлення від абоненту А (тобто отримав код сейфу).

8.2 Алгоритм цифрового підпису (автентифікація) на основі RSA.

Припустимо, що абонент А хотів би послати абоненту Б повідомлення, вміст якого він не вважає секретним, але бажає, щоб абонент Б був

впевнений в тому, що повідомлення прийшло саме від нього. В цьому випадку абонент Б використовує свій особистий ключ для шифрування повідомлення.

Коли абонент А отримає шифрований текст, він з'ясує, що його можна розшифрувати тільки за допомогою відкритого ключа Абонента Б. А це доведе, що повідомлення могло бути зашифровано тільки абонентом Б. Ніхто інший не має особистого ключа абонента Б, тому ніхто інший не міг створити зашифровані текст, дешифрований відкритим ключем абонента Б. У цьому випадку все шифрування повідомлення виступає в ролі цифрового підпису.

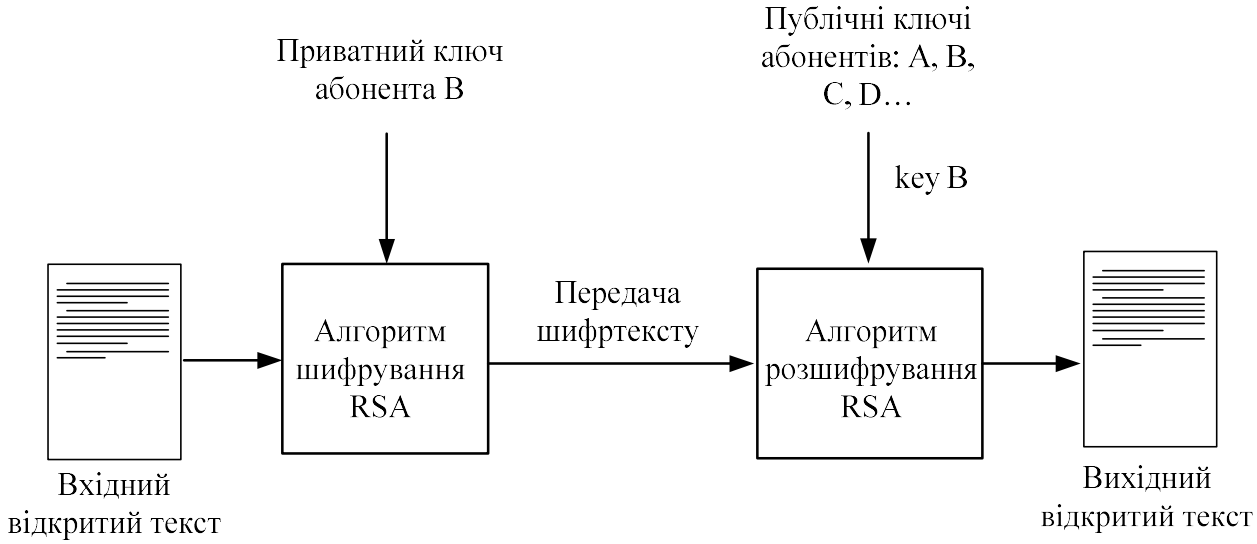


Рисунок 8.3 – Криптографія з відкритим ключем (автентифікація)

Але ця схема вимагає від системи досить багато ресурсів. Тому більш ефективним підходом виявляється шифрування невеликого блоку даних, що є функцією документа.

В цьому випадку цифровий підпис документа зазвичай створюється так: з документа генерується так званий дайджест (message digest) і до нього додається інформація про те, хто підписує документ, штамп часу та інше. Отриманий рядок далі зашифровується особистим ключем підписанта з використанням того чи іншого алгоритму. Одержаний зашифрований набір біт і являє собою підпис. До підпису зазвичай прикріплюється відкритий ключ підписанта.

Одержувач спочатку вирішує для себе чи довіряє він тому, що відкритий ключ належить саме тому, кому повинен належать і потім розшифровує підпис за допомогою відкритого ключа. Якщо підпис розшифрувався вірно, і його вміст відповідає документу, то повідомлення вважається підтвердженим.

Важливо пам'ятати, що в разі підпису, процес шифрування не забезпечує конфіденційності. Тобто пересилати таким чином повідомленням гарантує захист від підміни, але не від перехоплення.

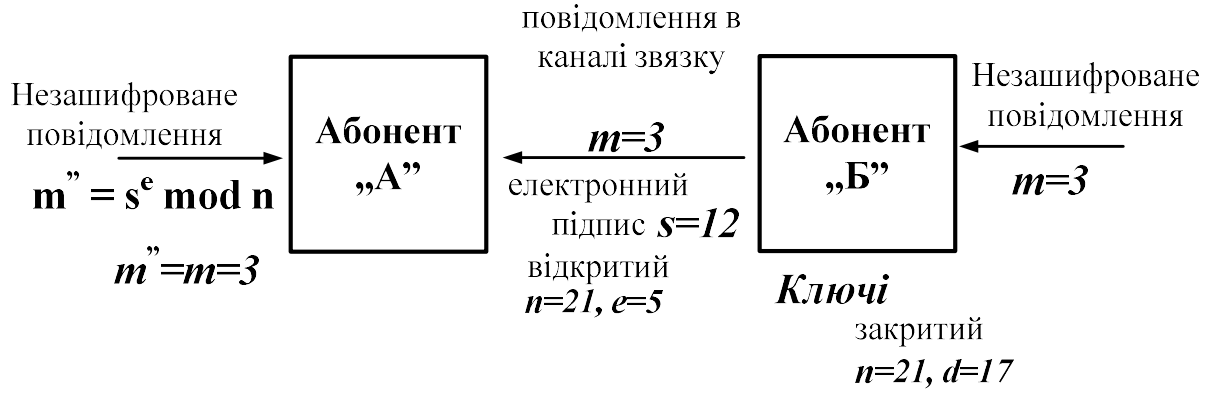


Рисунок 8.4 – Схема використання алгоритму RSA для автентифікації

Нехай абонент Б хоче відправити Абоненту А число 3 (код сейфу), але на цей раз не зашифроване, а підписана цифровим підписом. Щоб створити підпис Абонент Б використовує свій особистий ключ (21,17):

$$s = m^d mod\ n,$$

де: s - підпис, m- повідомлення.

У нашому випадку виходить $m^d = 3^{17} = 129140163$. Тоді s = 129140163 mod 21. Залишок дорівнює 12. Отже, абонент Б відправляє саме незашифроване повідомлення - 3, і підпис рівний 12, також з повідомленням відправляється відкритий ключ.

Отримавши повідомлення Абонент А перевіряє підпис. Для цього за допомогою відкритого ключа абонента Б йому необхідно перевірити, що $m = s^e \bmod n$, де s - підпис, m - повідомлення.

У нашому випадку, $m=12^5 \bmod 21=248832 \bmod 21 = 3$.

Все правильно. Повідомлення надіслане абонентом Б.

8.3 Алгоритм обміну ключами Діффі-Хелмана

Алгоритм Діффі–Хеллмана може бути використано задля розподілу ключів (генерування секретного ключа), але його не можна використовувати для шифрування повідомлення [28].

Згідно з цим алгоритмом, учасники інформаційного процесу ***A*** та ***B*** домовляються щодо значення великого простого числа ***p***і простого дискретного кореня цього числа ***a*** (рисунок 8.5).

Завдання дискретного виведення кореня звучить наступним чином:

За даними ***n***(***n***- просте), ***a***, ***к*** потрібно знайти всі ***x***, що задовільняють умові: $x^k \equiv a\ (mod\ p)$

$2^1 \text{mod } 7 = 2 \text{ mod } 7 = 2$
$2^2 \text{mod } 7 = 4 \text{ mod } 7 = 4$
$2^3 \text{mod } 7 = 8 \text{ mod } 7 = 1$
$2^4 \text{mod } 7 = 16 \text{ mod } 7 = 2$
$2^5 \text{mod } 7 = 32 \text{ mod } 7 = 4$
$2^6 \text{mod } 7 = 64 \text{ mod } 7 = 1$

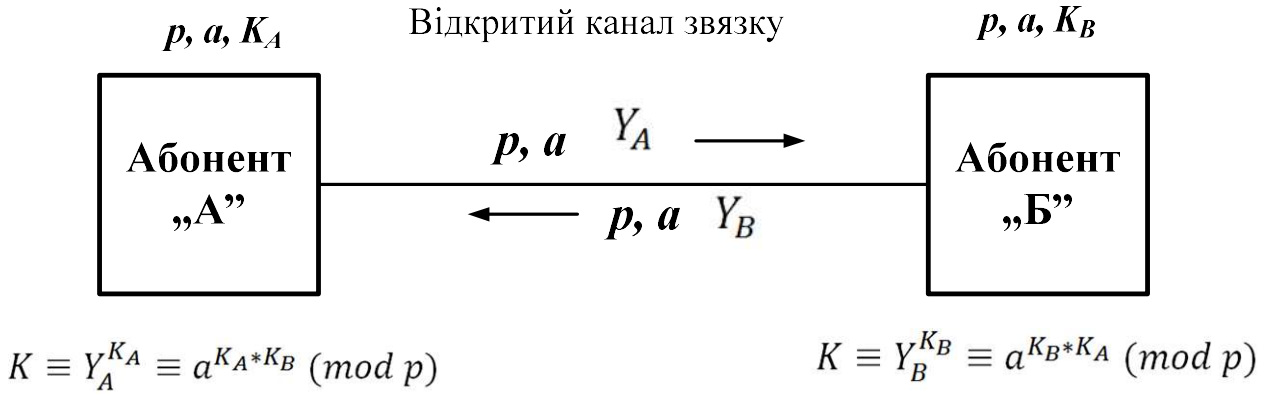


Рисунок 8.5 – Алгоритм обміну ключами Діффі-Хелмана

Сторона ***A*** обирає випадкове число $К_А$, а сторона ***B*** –випадкове число $К_В$ у такий спосіб, щоб виконувалися умови

$$1 < К_А < p - 1 \text{ та } 1 < К_В < p - 1.$$

Числа $К_А$ та $К_В$ тримаються сторонами *A* та *B* в секреті. **(секретні ключі).**

Сторона *A* формує відкритий ключ за правилом

$$Y_A \equiv a^{K_A} \ (mod\ p)$$

Аналогічно сторона *B* формує відкритий ключ за правилом

$$Y_B \equiv a^{K_B} \ (mod\ p)$$

Після обміну несекретними ключами Y_A та Y_B **(несекретні ключі)** сторони обчислюють значення секретного числа *K*:

$$K \equiv Y_В^{K_A} \equiv a^{K_A * K_B} \ (mod\ p)$$

$$K \equiv Y_А^{K_B} \equiv a^{K_B * K_A} \ (mod\ p)$$

Здобуте число *K* для ймовірного зловмисника є секретним, оскільки розв’язання рівнянь Y_A та Y_B для великих чисел є неможливе.

Найпростіша та оригінальна реалізація алгоритму використовує мультиплікативну групу цілих чисел за модулем ***p***, де ***p*** є простим, а ***a*** - примітивним кореневим модулем ***p***. Ці два значення вибираються таким чином, щоб гарантувати, що отримана загальна таємниця може приймати будь-яке значення від **1** до ***p* –1**.

1. Абоненти А та В публічно погоджуються використовувати модуль $\boldsymbol{p} = 23$ та основу $\boldsymbol{a} = 5$ (що є примітивним кореневим модулем 23). (або $\boldsymbol{p} = 17$; $\boldsymbol{a} = 3$).

2. Абонент А вибирає ***таємне ціле число*** $\boldsymbol{K_A = 4}$, а потім надсилає Абоненту В

$$Y_A = a^{K_A}(mod\ p) = 5^4(mod\ 23) = 625(mod\ 23) = 4$$

3. Абонент В вибирає ***таємне ціле число*** $\boldsymbol{K_B = 3}$, а потім надсилає Абоненту А

$$Y_\text{B} = a^{K_\text{B}}(mod\ p) = 5^3(mod\ 23) = 125(mod\ 23) = 10$$

4. Абонент А обчислює

$$K = Y_\text{B}^{K_A} = 10^4(mod\ 23) = 10000\ (mod\ 23) = 18$$

5. Абонент В обчислю

$$K = Y_\text{A}^{K_\text{B}} = 4^3(mod\ 23) = 64\ (mod\ 23) = 18$$

Третя сторона (зловмисник**)** має у своєму розпорядженні лише значення:

$$p = 23;\ a = 5;\ Y_A = 4;\ Y_\text{B} = 10.$$

Будь-яка кількість користувачів може брати участь в обміні даними, виконуючи ітерації протоколу договору та обмінюючись проміжними даними (які самі по собі не потрібно зберігати в таємниці). Наприклад, Абоненти А, В та С можуть брати участь в обміні даними по алгоритму Діффі-Хеллмана наступним чином , всі операції береться за модулю p:

1. Сторони домовляються про параметри алгоритму p і a .
2. Сторони генерують свої приватні ключі під назвою K_A , K_B і K_C .
3. Абонент А обчислю $Y_A = a^{K_A}$ і відправляє його Абоненту Б.
4. Абоненту Б обчислює $Y_\text{B} = (a^{K_A})^{K_\text{B}} = a^{K_A * K_\text{B}}$ і відправляє його Абоненту С.
5. Абонент С обчислює $Y_\text{B} = (a^{K_A * K_\text{B}})^{K_\text{B}} = a^{K_A * K_\text{B} * K_\text{c}}$ і використовує це як **свій таємний ключ.**
6. Абонент В обчислює $Y_\text{B} = a^{K_\text{B}}$ і відправляє його Абоненту С.
7. Абоненту С обчислює $Y_\text{C} = (a^{K_\text{B}})^{K_\text{C}} = a^{K_\text{B} * K_\text{C}}$ і відправляє його Абоненту А.
8. Абоненту А обчислює$Y_\text{A} = (a^{K_\text{B} * K_\text{C}})^{K_\text{A}} = a^{K_\text{B} * K_\text{C} * K_\text{A}}$ і і використовує це як **свій таємний ключ.**
9. Абонент С обчислює $Y_\text{C} = a^{K_\text{C}}$ і відправляє його Абоненту А.
10. Абоненту А обчислює $Y_\text{C} = (a^{K_\text{C}})^{K_\text{A}} = a^{K_\text{C} * K_\text{A}}$ і відправляє його Абоненту В.
11. Абоненту В обчислює$Y_\text{C} = (a^{K_\text{C} * K_\text{A}})^{K_\text{B}} = a^{K_\text{C} * K_\text{A} * K_\text{B}}$ і використовує це як **свій таємний ключ.**

Третя сторона **зловмисник** має у своєму розпорядженні лише значення:

a^{K_A}; a^{K_B}; a^{K_C}; $a^{K_A*K_B}$; $a^{K_A*K_C}$; $a^{K_B*K_C}$, але не може використовувати будь-яку їх комбінацію для ефективного відтворення $a^{K_A*K_B*K_C}$.

8.3 Порядок виконання роботи

1. Розглянути основні теоретичні відомості асиметричних криптосистем шифрування інформації та алгоритмів обміну ключами.
2. Проаналізувати наведені в додатку Е приклади програмного коду.
3. На підставі проведеного аналізу у відповідності з варіантом лабораторної роботи реалізувати систему передачі повідомлення з відкритим ключем або обміну ключами з урахування властивостей певного алгоритму.
4. Реалізувати дані алгоритми у вигляді програмного коду.

8.4 Зміст звіту

1. Звіт з лабораторної роботи повинен бути виконаний на аркушах формату А4.
2. Звіт повинен містити: назву лабораторної роботи, її мету і короткі теоретичні відомості у відповідності до завдання.
3. В розділі «Результати виконання лабораторної роботи» студент повинен представити результати виконання лабораторної роботи у відповідності зі своїм варіантом, а також навести код програмної реалізації.
4. Написати висновки до даної лабораторної роботи.
5. Оформити звіт та представити його викладачу на захист.

Таблиця 8.1 - Варіанти завдань

Варіант	Завдання	Додаткові параметри
1.	Автентифікація на основі RSA	***p***= 97; ***q***=131;
2.	Алгоритм шифрування RSA	***p***= 97; ***q***=131;
3.	Алгоритм Діффі-Хелмана	к-ть абонентів 3; ***p***= 97;
4.	Автентифікація на основі RSA	***p***= 113; ***q***=877;
5.	Алгоритм шифрування RSA	***p***= 113; ***q***=877;
6.	Алгоритм Діффі-Хелмана	к-ть абонентів 4; ***p***= 113;
7.	Автентифікація на основі RSA	***p***= 151; ***q***=587;
8.	Алгоритм шифрування RSA	***p***= 151; ***q***=587;
9.	Алгоритм Діффі-Хелмана	к-ть абонентів 4; ***p***= 151;
10.	Автентифікація на основі RSA	***p***= 43; ***q***=677;
11.	Алгоритм шифрування RSA	***p***= 43; ***q***=677;
12.	Алгоритм Діффі-Хелмана	к-ть абонентів 2; ***p***= 43;
13.	Автентифікація на основі RSA	***p***= 283; ***q***=557;
14.	Алгоритм шифрування RSA	***p***= 283; ***q***=557;
15.	Алгоритм Діффі-Хелмана	к-ть абонентів 3; ***p***= 283;
16.	Автентифікація на основі RSA	***p***=109; ***q***=827;

Продовження таблиці 8.1 - Варіанти завдань

Варіант	Завдання	Додаткові параметри
17.	Алгоритм шифрування RSA	***p***=109; ***q***=827;
18.	Алгоритм Діффі-Хелмана	к-ть абонентів 4; ***p***= 109;
19.	Автентифікація на основі RSA	***p***=79; ***q***=313;
20.	Алгоритм шифрування RSA	***p***=79; ***q***=313;
21.	Алгоритм Діффі-Хелмана	к-ть абонентів 2; ***p***=79;
22.	Автентифікація на основі RSA	***p***=59; ***q***=701;
23.	Алгоритм шифрування RSA	***p***=59; ***q***=701;
24.	Алгоритм Діффі-Хелмана	к-ть абонентів 3; ***p***=59;
25.	Автентифікація на основі RSA	***p***=421; ***q***=317;
26.	Алгоритм шифрування RSA	***p***=421; ***q***=317;
27.	Алгоритм Діффі-Хелмана	к-ть абонентів 4; ***p***=421;
28.	Автентифікація на основі RSA	***p***=127; ***q***=229;
29.	Алгоритм шифрування RSA	***p***=127; ***q***=229;
30.	Алгоритм Діффі-Хелмана	к-ть абонентів 3; ***p***=127;

Контрольні запитання

1. Що таке одностороння функція?
2. Що розуміють під виразом асиметричний криптоалгоритм?
3. Що таке взаємно прості числа?
4. Що таке публічний ключ?
4. Як обчислити функцію Ейлера для алгоритму RSA?
5. Що таке приватний ключ?
6. Як вирахувати секретну експоненту на основі відкритої експоненти?
7. Що таке генерація так званого дайджеста?

Додаток Е

```
#include <conio.h>
#include <Windows.h>
#include <math.h>
#include <string>
using namespace std;
string SimpleNumbersFn(int fn) {
  string simpleNumbers;
  int div = 2;
  while (fn > 1)
  {
    while (fn % div == 0)
    {
      simpleNumbers += to_string(div);
      fn = fn / div;
    }
    div++;
```

```
  }
  return simpleNumbers;
}
bool is_prime(int x) {
  for (int i = 2; i <= sqrt(x); i++) {
    if (x % i == 0) {
      return false;
    }
  }
  return true;
}
int E(string wrongNumbrs) {
  int n = 2;
  string str;
  while (1) {
    if (is_prime(n)) {
      for (int i = 0; i < wrongNumbrs.length(); i++) {
        str = wrongNumbrs[i];
        if (to_string(n) == str) break;
        if (i == wrongNumbrs.length() - 1) return n;
      }
    }
    n++;
  }
}
int D(int e, int fn) {
  int k = 1;
  while (1) {
    if (((k * e) - 1) % fn == 0) return k;
    else k++;
  }
}
int main()
{
  SetConsoleCP(1251);
  SetConsoleOutputCP(1251);
  long long p, q, n, fn, e, d, numb, encryptNumb;
  string choice, simpleNumbers;
  do {
    cout << "Введіть p = ";
    cin >> p;
    cout << "Введіть q = ";
    cin >> q;
    n = p * q;
  } while (n < 1);
  fn = (p - 1) * (q - 1);
  simpleNumbers = SimpleNumbersFn(fn);
  e = E(simpleNumbers);
  d = D(e, fn);
  while (1) {
```

```
    cout << "1. Зашифрувати\n2. Дешифрувати\n3. Інфо\n4.
Змінити p та q\n\nОберіть
         пункт меню: ";
    cin >> choice;
    if (choice == "1") {
      cout << "\nВведіть число для шифрування: ";
      cin >> numb;
      encryptNumb = pow(numb, e);
      encryptNumb = encryptNumb % n;
      if (encryptNumb < 0) cout << "Занадто великі числа для
обрахунків.";
      else cout << "Зашифроване число: " << encryptNumb;
    }
    else if (choice == "2") {
      cout << "\nВведіть число для дешифрування: ";
      cin >> encryptNumb;
      numb = pow(encryptNumb, d);
      numb = numb % n;
      if (numb < 0) cout << "Занадто великі числа для
обрахунків.";
      else cout << "Деашифроване число: " << numb;
    }
    else if (choice == "3") {
      cout << "\np = " << p << "\nq = " << q << "\nn = " << n
<< "\nf(n) = "
           << fn << "\ne = " << e << "\nd = " << d;
    }
    else if (choice == "4") {
      do {
        cout << "Введіть p = ";
        cin >> p;
        cout << "Введіть q = ";
        cin >> q;
        n = p * q;
      } while (n < 1);
      fn = (p - 1) * (q - 1);
      simpleNumbers = SimpleNumbersFn(fn);
      e = E(simpleNumbers);
      d = D(e, fn);
    }
    else {
      cout << "\nERROR!";
    }
    _getch();
    numb = 0; encryptNumb = 0;
    cout << endl << endl;
  }
}
```

ГЛОСАРІЙ

Автентифікація (authentication) – процедура встановлення належності користувачеві інформації в системі пред'явленого ним ідентифікатора. З позицій інформаційної безпеки автентифікація є частиною процедури надання доступу для роботи в інформаційній системі, наступною після ідентифікації і передує авторизації.

Ардуіно (Arduino) – апаратна обчислювальна платформа для аматорського конструювання, основними компонентами якої є плата мікроконтролера з елементами вводу/виводу та середовище розробки Processing/Wiring на мові програмування, що є спрощеною підмножиною C/C++.

Атрибут (Attribute) – це параметр файлу, що визначає певні властивості файлів і втановлюється як властивість конкретного файла.

Біт (Bit) – мінімальна одиниця кількості інформації, яка дорівнює одному двійковому розряду, який може бути рівним одному з двох значень/станів, застосовуваних для представлення даних у двійковій системі числення.

Біт-стафінг (Beat staffing) – вставка неінформаційних бітів в потік даних. Застосовується при передачі даних і в телекомунікації.

Вікно (Window) – візуально відокремлена область екрана з певним інтерфейсом користувача. Зазвичай вікно має прямокутну форму. В ньому відображаються елементи вводу і, можливо, виводу для одного або декількох процесів. Вікнами можна управляти за допомогою курсора миші та клавіатури.

Дайджест (Digest) – в криптографії результат перетворення повідомлення довільної довжини, в хешований рядок фіксованої довжини.

Дешифрування (Decryption) – процес несанкціонованого отримання інформації з зашифрованих даних. При цьому ключ дешифрування зазвичай невідомий. Вивчається крипто аналізом.

Ідентифікація (Identification) – процедура розпізнавання користувача в системі, як правило, за допомогою наперед визначеного імені (ідентифікатора) або іншої апріорної інформації про нього, яка сприймається системою.

Інтегроване середовище розробки (Integrated development environment (IDE)) – комплексне програмне рішення для розробки програмного забезпечення. Зазвичай, складається з редактора початкового коду, інструментів для автоматизації складання та відлагодження програм.

Інтерфейс апаратний (Hardware interface) – сукупність алгоритмів обміну і технічних засобів, що забезпечують обмін інформацією між пристроями. Апаратний інтерфейс є системою шин, роз'ємів, узгоджувальних пристроїв, алгоритмів і протоколів, що забезпечують зв'язок всіх частин мікропроцесорної системи між собою. Від характеристик інтерфейсу залежить швидкодія і надійність системи. У розгорнутих мікропроцесорних системах для розвантаження процесора апаратний інтерфейс забезпечується контролерами.

Інтерфейс користувача (User interface) – різновид інтерфейсу, в якому одна сторона представлена людиною (користувачем), інша – машиною/пристроєм. Це сукупність засобів і методів, за допомогою яких користувач взаємодіє з різними, найчастіше складними, машинами, пристроями, апаратурою і програмним забезпеченням. Часто термін застосовується щодо комп'ютерних програм, однак під ним може матися на увазі набір засобів, методів і правил взаємодії будь-якої системи, керованої людиною. Інтерфейс вважається двонаправленим (інтерактивним), якщо пристрій, отримавши команди від користувача і виконавши їх, видає інформацію користувачу наявними у нього засобами – візуальними, звуковими, тактильними тощо. Інтерфейс – це сукупність елементів, які, в свою чергу, також можуть складатися з елементів.

Інтерфейс програмний (Software interface) – набір готових класів, процедур, функцій, структур і констант, що надаються інформаційною системою (бібліотекою, сервісом) для використання у зовнішніх програмних продуктах. Використовується програмістами для написання власних програмних засобів.

Кадр (Frame) – фрагмент даних мережевого протоколу канального рівня, що передається по лінії зв'язку.

Ключ (Key) – параметр криптографічної системи, який використовується для шифрування і/або дешифрування повідомлення при шифруванні. Накладення та перевірки коду автентифікації повідомлень або електронного цифрового підпису.

Кодування (Coding) – перетворення будь-якої інформації на послідовність імпульсів, що мають властивість самосинхронізації для передачі через канали зв'язку.

Комп'ютер (Computer) – багатозначний термін, найчастіше вживається для означення програмно керованого електронного пристрою обробки інформації. Разом з тим, це може бути будь-який механічний, немеханічний (електронний) пристрій, що призначений для проведення обчислень.

Криптостійкість (Cryptoresistance) – здатність криптографічного алгоритму протистояти криптоаналізу. У більшості випадків крипостійкість не можна математично довести, можна тільки довести уразливості криптографічного алгоритму.

Масив (Massif) – впорядкований набір фіксованої кількості однотипних елементів, що зберігаються в послідовно розташованих комірках оперативної пам'яті, мають порядковий номер і спільне ім'я, що надає їм користувач. Характеристика масиву: розмірність (кількість індексів елемента – одновимірний, двовимірний, багатовимірний); розмір (загальна кількість елементів у масиві). За типом масиви поділяються на числові та символьні.

Мікроконтролер (Microcontroller) – або одно кристальний мікрокомп'ютер — виконаний у вигляді мікросхеми спеціалізований комп'ютер, що включає мікропроцесор, оперативну та постійну пам'ять для збереження виконуваного коду програм і даних, порти вводу-виводу і блоки зі спеціальними функціями.

Одностороння функція (One-way function) – математична функція, яка легко обчислюється для будь-якого вхідного значення, але важко знайти аргумент за заданим значенням функції. Тут «легко» і «важко» повинні розумітися з точки зору теорії складності обчислень. Розрив між складністю прямого і зворотного перетворень визначає криптографічну ефективність односторонньої функції.

Порт мікроконтролера (Microcontroller port) – це пристрої введення / виведення, що дозволяють мікроконтролеру передавати або приймати дані. Стандартний порт мікроконтролера AVR має вісім розрядів даних, які можуть передаватися або прийматися паралельно. Кожному розряду (або біту) відповідає вивід (ніжка) мікроконтролера.

Програма (Program) – послідовність інструкцій, призначених для виконання пристроєм управління обчислювальної машини. Програма – один з компонентів програмного забезпечення. Залежно від контексту цей термін може відноситися також і до вихідних текстів програми.

Програматор (Programmer) – апаратно-програмний пристрій, призначений для запису / зчитування інформації в постійний запам'ятовуючий пристрій (одноразово записується, флеш-пам'ять, внутрішню пам'ять мікроконтролерів).

Програмне забезпечення (Software) – сукупність програм, процедур і правил, а також документації, що стосуються функціонування системи оброблення даних з використанням електронних обчислювальних машин.

Раунд (Round) – в криптографії називають один з послідовних кроків обробки даних в алгоритмі блочного шифрування. В шифрах побудованих відповідно до архітектури мережі Фейстеля і близьких йому по архітектурі шифрах - один крок шифрування, в ході якого одна або кілька частин шифруючого блоку даних піддається модифікації шляхом застосування кругової функції.

Розшифрування (Decryption) — процес санкціонованого перетворення зашифрованих даних у придатні для читання. Вивчається криптографією.

Система числення (calculation system) – сукупність правил і знаків, за допомогою яких можна відобразити (кодувати) будь-яке число. До систем числення висуваються певні вимоги, серед яких найбільш важливими є вимоги однозначного кодування невід'ємних чисел 0, 1, … до деякої їх скінченної множини за скінченне число кроків і можливості виконання з числами арифметичних і логічних операцій. Крім того, системи числення розв'язують задачу нумерації, тобто ефективного переходу від зображень чисел до номерів, які в даному випадку повинні мати мінімальну кількість цифр.

Скетч (Sketch) – це програма, написана для платформи Arduino і має певну структуру. Скетч обов'язково містить 2 функції: функцію Setup і функцію Loop .

Скремблювання (Scrambling) – полягає в побітному обчисленні результуючого коду на підставі бітів вихідного коду і отриманих в попередніх тактах бітів результуючого коду.

Цикл (Cycle) – різновид керівної конструкції у високорівневих мовах програмування, призначена для організації багаторазового виконання набору інструкцій. Також циклом може називатися будь-яка багатократно виконувана послідовність команд, організована будь-яким чином.

Шифрування (Encryption) (в системах обробки інформації) – алгоритмічне (криптографічне) перетворення даних, яке виконується у посимвольній послідовності з метою одержання шифрованого тексту.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Wiring [Електронний ресурс] – Режим доступу до ресурсу: http://wiring.org.co/.

2. Arduino IDE 1.8.13 [Електронний ресурс] – Режим доступу до ресурсу: https://www.arduino.cc/en/software.

3. Матеріали по програмуванню Arduino [Електронний ресурс] – Режим доступу до ресурсу: https://www.arduino.cc/.

4. Arduino Uno [Електронний ресурс] // Arduino – Режим доступу до ресурсу: https://doc.arduino.ua/ru/hardware/Uno.

5. Корченко О. Г. Прикладна криптологія: системи шифрування: підручник / О. Г. Корченко, В. П. Сіденко, Ю. О. Дрейс. – К. : ДУТ, 2014. – 448 с.

6. Воробиенко П. П. Формирование служебной информации в процессе сеанса связи сетевых компьютерных приложений: матеріали 64 науково-технічної конференції професорсько-викладацького складу, науковців, аспірантів та студентів «Сучасні інформаційні системи і технології», 1 - 4 грудня 2009 р., м. Одеса / П. П. Воробиєнко, М. І. Струкало, С. М. Струкало. – О.: ОНАЗ ім. О. С. Попова, 2009. – С. 92-94.

7. Katie Terrell Hanna. Bit stuffing. [Електронний ресурс] – Режим доступу до ресурсу: https://www.techtarget.com/searchnetworking/definition/bit-stuffing.

8. Методи цифрового кодування [Електронний ресурс] // StudFiles. – 2016. – Режим доступу до ресурсу: https://studfile.net/preview/7543085/page:10/.

9. Яцків В. В. Методи кодування даних для фізичного рівня комп’ютерних мереж / В. В. Яцків, Ю. В. Кудряшов. // Вісник Національного університету “Львівська політехніка”. – 2012. – №443. – С. 131–134.

10. Інформаційно-методичне забезпечення навчального процесу студентів спеціальності "Інформаційно-вимірювальні технології" [Електронний ресурс] // НТУУ "КПІ ім. Ігоря СІКОРСЬКОГО" – Режим доступу до ресурсу: https://itcj.sethost.net/content/?page_id=1908.

11. Шувалов В. П., Захарченко Н. В., Шварцман В. О. Передача дискретных сообщений. Учебник для вузов / отв. Шувалова В. П.. — М.: Радио и связь, 1990. — 464 с. — ISBN 5-256-00852-8.

12. Хлобистова О. А. Технології захисту інформації [Електронний ресурс]: навчальний посібник./ О. А. Хлобистова, Ю. Г. Савченко, М. В. Гладка – К.: НУХТ, 2014. – 84 с.

13. Методи цифрового кодування [Електронний ресурс] // StudFiles. – 2016. – Режим доступу до ресурсу: https://studfile.net/preview/8983581/

14. Пономаренко В. С. Основи захисту інформації. Навчальний посібник / В. С. Пономаренко, І. В. Журавльова. – Харків: Вид. ХДЕУ, 2003. – 176 с.

15. Карачка А. Ф. Технології захисту інформації [Текст лекцій]. – Тернопіль: ТНЕУ, 2017– 86 с.

16. Кузнецов О. О. Стеганографія : навчальний посібник / О. О. Кузнецов, С. П. Євсеєв, О. Г. Король. – Х. : Вид. ХНЕУ, 2011. – 232 с.

17. Остапов С. Е. Технології захисту інформації / С. Е. Остапов, С. П. Євсеєв, О. Г. Король. – Харків: Вид. ХНЕУ, 2013. – 476 с.

18. Інформаційні основи побудови телекомунікаційних мереж / В. В. Казимір, В. В. Шкаляр, С. М. Литвинов, С. В. Зайцев. – Чернігів: Чернігівський державний технологічний університет, 2013. – 340 с. – ISBN 978-966-7496-46-3

19. ДСТУ 3396.2–97. Захист інформації. Технічний захист інформації. Терміни і визначення. - К.: Держстандарт України, 1998.

20. Feistel Ciphers: Security Proofs and Cryptanalysis / Valerie Nachef, Jacques Patarin, Emmanuel Volte. – Springer Nature eBook, 2017. – 309 p. – ISBN 978-331-949-530-9

21. Мережа Фейстеля [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://znaimo.com.ua/%D0%9C%D0%B5%D1%80%D0%B5%D0%B6%D0%B0_%D0%A4%D0%B5%D0%B9%D1%81%D1%82%D0%B5%D0%BB%D1%8F#link32.

22. TEA [Електронний ресурс] // Матеріал з Вікіпедії — вільної енциклопедії.. – 2020. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/TEA#cite_note-teasite-1.

23. Шифр TEA [Електронний ресурс] // Сучасні криптосистеми – Режим доступу до ресурсу: https://sites.google.com/site/sucasnikriptosistemik/home/simetricni-kriptosistemi/blocni-sifri/sifr-tea.

24. A. Menezes, P. van Oorschot, S. Vanstone. 7.6 IDEA / / Handbook of Applied Cryptography - www.cacr.math.uwaterloo.ca/hac/about/chap7.pdf. - CRC-Press, 1996. - P. 263. - (Discrete Mathematics and Its Applications). - ISBN 0-8493-8523-7

25. Розблокувати захищені паролем ZIP –файли. [Електронний ресурс] / – 2021. – Режим доступу до ресурсу: https://ciksiti.com/uk/chapters/6864-unlock-password-protected-zip-files

26. Технології захисту інформації [Електронний ресурс] / Ю. А. Тарнавський; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 2,04 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2018. – 162 с.

27. Хорошко В. О. Комп`ютерна стеганографія : [навчальний посібник] / В. О. Хорошко, Ю. Є. Яремчук, В. В. Карпінець – Вінниця: ВНТУ, 2017. – 155 с.

28. Лужецький В. А. Основи інформаційної безпеки. Навчальний посібник. / В. А. Лужецький, А. Д. Кожухівський, О. П. Войтович – Вінниця: ВНТУ, 2009. – 268 с.

Електронне навчальне видання
комбінованого використання.
Можна використовувати в локальному та мережному режимах

Белзецький Руслан Станіславович
Яровий Андрій Анатолійович
Іванчук Ярослав Володимирович

ТЕХНОЛОГІЯ ЗАХИСТУ ІНФОРМАЦІЇ

Лабораторний практикум

Рукопис оформив *Р. Белзецький*

Видається в авторській редакції

Оригінал-макет виготовила *О. Кушнір*

Підписано до видання 15.06.2022 р.
Зам. № Р2022-030

Видавець та виготовлювач
Вінницький національний технічний університет,
редакційно-видавничий відділ.
ВНТУ, ГНК, к. 114.
Хмельницьке шосе, 95,
м. Вінниця, 21021.
Тел. (0432) 65-18-06.
press.vntu.edu.ua;
Email: irvc.vntu@gmail.com.
Свідоцтво суб'єкта видавничої справи
серія ДК № 3516 від 01.07.2009 р.