**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»**

**Спеціальність: 123 «Комп'ютерна інженерія»**

**Освітня програма: «Комп'ютерна інженерія»**

**Група: 2БКС-28**

# КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

**здобувача освіти денної форми навчання**

**БКС.28.15.000.КРБ**

***МАСАЛИТІН***
***РУСЛАН ВІКТОРОВИЧ***

**м. Одеса**
**2024 р.**

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **123 «Комп'ютерна інженерія»**

Освітньо-професійна програма: **«Комп'ютерна інженерія»**

Група: **2БКС-28**

# ПОЯСНЮВАЛЬНА ЗАПИСКА

До кваліфікаційної роботи бакалавра на тему: **«Аналіз ефективності сучасних потокових криптоалгоритмів»**

Проектний матеріал складається з пояснювальної записки на 80 сторінках та графічного (презентаційного) матеріалу на 19 аркушах (слайдах)

Виконавець _____ ( Масалитін Р.В.)

Керівник проекту _____ ( Кривченко А.А.)

**Консультанти:**

з розділу охорони праці та техніки безпеки _____ ( Чорновол Н.І.)

з нормоконтролю _____ ( Петрашова В.І.)

старший консультант _____ ( Кривченко Ю.В.)

**До захисту допущений**

Завідувач кафедри _____ ( Іванова Л.В.)

Завідувач відділення _____ ( Скорнякова О.В.)

Захист «24» 06 2024 р. Протокол ЕК № 1

Оцінка ЕК 4 (добре)

Секретар ЕК _____

# АНОТАЦІЯ

У випускній кваліфікаційній роботі розглянуто сильні і слабкі сторони сучасних симетричним потокових криптоалгоритмів та виконано програмну реалізацію одного з них.

Розглянуто структуру та виконано порівняльний аналіз сучасних симетричних потокових алгоритмів шифрування. Обрано найбільш надійний і швидкодіючий потоковий криптоалгоритм для подальшої реалізації та дослідження. Розглянуто особливості реалізації та склад, схеми криптоалгоритмів RC4, A5, SEAL, Salsa20, SNOW, Sosemanuk, ChaCha, Rabbit, HC-128, Grain, MICKEY, Trivium, OTP. Проведено тестування швидкодії обраних потокових криптоалгоримів.

Обрано надійні генератори псевдовипадкових чисел для реалізації констант та ключів. За допомогою програмного забезпечення для моделювання криптографічних алгоритмів CrypTool2 виконано моделювання роботи потокового криптографічного алгоритму ChaCha. Промодельовано хешування за алгоритмом SHA-512. Виконано розробку програмної моделі кодеку ChaCha з візуальним інтерфейсом користувача мовою C++ засобами інтегрованого середовища розробки QT. Реалізовано перевіркою на цілісність даних за допомогою хешування. Перевірено результати роботи реалізованого криптоалгоритму за допомогою середовища CrypTool2.

Описано заходи з охорони праці та техніки безпеки.

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»**

Відділення *Комп'ютерних систем* Кафедра *Комп'ютерної інженерії*

Спеціальність *123 «Комп'ютерна інженерія»*

Освітньо-професійна програма *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР *Беркань І.В.*

"15" 01 2024 р.

# ЗАВДАННЯ

**на кваліфікаційну роботу бакалавра**

здобувачеві освіти *Масалитіну Руслану Вікторовичу*

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Аналіз ефективності сучасних потокових криптоалгоритмів*

затверджена наказом по коледжу від "02" 11 2023 р. № 244-АГ-ОД

2. Термін здачі студентом кваліфікаційної роботи 13.06.2024

3. Вихідні дані до роботи *1. Специфікації потокових симетричних криптоалгоритмів RC4, A5, SEAL, Salsa20, SNOW, Sosemanuk, ChaCha, Rabbit, HC-128, Grain, MICKEY, Trivium, OTP; 2. Провести тестування швидкодії обраних потокових криптоалгоримів; 3. Реалізувати програмну модель кодеку на базі обраного симетричного потокового шифру з перевіркою на цілісність даних; 4. Реалізувати візуальний інтерфейс користувача для кодеку; 5. Реалізувати моделювання роботи реалізованого криптоалгоритму за допомогою CrypTool2*

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

*1. Аналіз методів шифрування у симетричних криптоалгоритмах; 2. Порівняльний аналіз сучасних потокових алгоритмів шифрування; 3. Вибір потокового криптоалгоритму для подальшої реалізації та дослідження; 4. Розробка моделі кодеку та його програмна реалізація; 5. Питання з охорони праці та техніки безпеки*

5. Перелік графічного матеріалу (слайдів мультимедійної презентації) *Узагальнена схема симетричного шифрування; Принципи потокового шифрування; Принцип хешування; Генерування псевдовипадкового ключу; Результати порівняння швидкості потокових криптоалгоритмів; Блок-схеми алгоритму, на якому заснований кодек; Цикл роботи обраного потокового алгоритму; Результат моделювання за алгоритмом хешування; Результат моделювання шифрування за обраним потоковим алгоритмом; Інтерфейс програмного застосунку кодеку; Приклад дешифрування програмою-кодеком повідомлення*

6. Консультанти по кваліфікаційній роботі, із зазначенням розділів, що їх стосуються

| Розділ | Консультант | ПІДПИС | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| Основний розділ | *Кривченко А.А.* | | |
| Розділ охорони праці | *Чорновол Н.І.* | | |
| Нормоконтроль | *Петрашова В.І.* | | |
| Старший консультант | *Кривченко Ю.В.* | | |

7. Дата видачі завдання 15.01.2024

Керівник роботи *Кривченко А.А.* _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

## КАЛЕНДАРНИЙ ПЛАН

| Пор. № | Назва етапів кваліфікаційної роботи | Термін виконання етапів роботи | Примітка |
|---|---|---|---|
| *1.* | *Вступ. Аналіз технічного завдання* | 08.05.24 | виконано |
| *2.* | *Аналіз властивостей алгоритмів симетричного* | 10.05.24 | |
| | *шифрування* | | виконано |
| *3.* | *Порівняльний огляд сучасних потокових алгоритмів* | 12.05.24 | |
| | *шифрування* | | виконано |
| *4.* | *Визначення найкращих потових алгоритмів шифрування* | 16.05.24 | виконано |
| *5.* | *Тестування швидкодії потокових криптоалгоритмів* | 20.05.24 | виконано |
| *6.* | *Розробка моделі кодеку та його програмна реалізація* | 22.05.24 | виконано |
| *7.* | *Реалізація генератору випадкових чисел та хешування* | 27.05.24 | виконано |
| *8.* | *Вибір програмних засобів розробки* | 29.05.24 | виконано |
| *9.* | *Підготовка програмних складових програмного проекту* | 01.06.24 | виконано |
| *10.* | *Розробка програми кодеку мовою С++* | 03.06.24 | виконано |
| *11.* | *Аналіз криптостійкості реалізовано криптоалгоритму* | 05.06.24 | виконано |
| *12.* | *Розробка питань з охорони праці* | 07.06.24 | виконано |
| *13.* | *Оформлення слайдів презентації* | 10.06.24 | виконано |
| *14.* | *Підготовка до захисту випускної роботи* | 12.06.24 | виконано |

Здобувач освіти _____
(підпис)

Керівник роботи _____
(підпис)

| Формат | Зона | Поз. | Позначення | Назва | Кіл. | Примітка |
|---|---|---|---|---|---|---|
| | | | | Документація | | |
| | | | БКС 28. 15 000. 00 КРБ | Випускна робота | | |
| А4 | | | БКС 28. 15 000. 00 КРБ ПЗ | Пояснювальна записка | 1 | |

| Зм. | Арк. | № докум. | Підпис | Дата | БКС 28. 15 000. 00 КРБ | | |
|---|---|---|---|---|---|---|---|
| Розробив | | Масалитін Р.В. | | 10.06 | Аналіз ефективності сучасних потокових криптоалгоритмів | Літ. | Аркуш 4 / Аркушів 80 |
| Перевірив | | Кривченко А.А. | | 10.06 | | НДП | |
| Н. Контр. | | Петрашова В.І. | | 10.06 | | ВСП "ОТФК ОНТУ" гр.2БКС-28 | |
| Затверд. | | Іванова Л.В. | | 10.06 | | | |

# ЗМІСТ

# ВСТУП

В процесі економічного розвитку держави і у ході конкурентної боротьби у національному та міжнародному ринках інформація грає вирішальну роль. Люди вже давно навчилися оцінювати важливість даних та важливість збереження її у таємниці. Конфіденційна інформація можливе найбільшу цінність. Підраховано, коли втрата банком 20-25% конфіденційній даних веде щодо його банкрутства.

Криптографія являється рішенням задля запобігання втрати даних. Мета криптографічної схеми зашифрування полягає у саме тому, аби зробити сповіщення незрозумілим таким способом, аби отриманий шифротекст виявлявся задля супротивника малозрозумілим чи взагалі не містив даних про вміст переданого набору рядків, дозволяючи одержувачу перетворити текст поза поміччю спеціального процесу задля отримання оригіналу сповіщення. Криптографічні перетворювачі повинні забезпечити математичні способи захисту конфіденційних повідомлень. Навіть в разі їх перехоплення зловмисниками та обробки будь-якими способами із використанням самих швидкодіючих суперкомп'ютерів та останніх досягнень науки та техніки смисловий зміст повідомлень може мати можливість розкритий тільки у перебігу заданого часу, наприклад протягом декількох днів.

В сучасному інформаційному світі існує багато методів, що повинні захищати дані, але всі вони повинні різну надійність. Оскільки існує велика число методів криптоаналізу, коли дають змогу зробити вразливими багато методів зашифрування, то питання підвищення криптостійкості методів зашифрування вийде актуальним задля галузі криптографії, адже у сучасному світі цифрових технологій та великого відправлення даних являється важливим захист бітів, що передаються через різні канали зв'язку.

В даній кваліфікаційній роботі треба проаналізувати ефективність сучасних потокових криптоалгоритмів, коли дасть змогу визначити слабкі і сильні сторони їхнього впровадження. Треба визначити основні способи криптоаналізу, що спроможні зробити шифри уразливими та становити загрозу розкриття відкритого набору рядків. Також треба розглянути способи генерації перестановок і реалізувати один із потокових криптоалгоритмів із метою його докладного вивчення.

# 1 ОСНОВНИЙ РОЗДІЛ

## 1.1 Аналіз методів зашифрування в симетричних криптоалгоритмах

Головним способом всі способи зашифрування поділяються у симетричні і асиметричні, у залежності з структури використовуваних паролів. Крім того, способи зашифрування спроможні мати різну криптостійкість та по-різному обробляти вхідні дані – блокові шифри та потокові шифри. Код являється фундаментальною частиною захисту конфіденційності даних, сповіщення чи частини бітів.

Процес зашифрування і декодування може мати можливість ініційований тільки поза поміччю ключу. Ключі зашифрування розроблені таким способом, аби вони були абсолютно унікальні, використовуючи набір різних методів. Код зашифрування застосовується задля кодування чи декодування бітів.

Коли правило, код являється випадковим двійковим чи фактичним паролем. В зв'язку із тим, коли способи являється загальнодоступними та доступні задля будь-кого, якщо хакер отриможливе код зашифрування, зашифровані дані легко розшифруються щодо відкритого набору рядків.


Рисунок 1.1. Узагальнена схема симетричного зашифрування

Способи симетричного зашифрування використовують один та той самий код задля кодування та декодування даних (рис. 1.1). Його найкраще застосовувати при обміні невеликих об'ємів бітів один із одним. Симетричне зашифрування набагато швидше ніж асиметричне, але відправник можливе обмінятися паролем зашифрування із одержувачем, перш ніж останній зможе розшифрувати сповіщення.

Потокові способи зашифрування шифрують і дешифрують кожний окремий бітів (рис. 1.2).

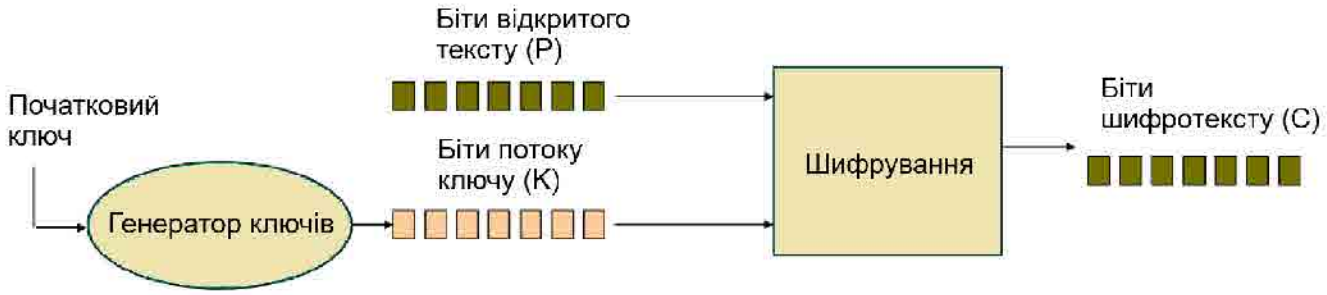



Рисунок 1.2. Принципи поточного зашифрування

В порівнянні із блочним даний спосіб являється більш складним, але швидшим, оскільки не витрачає час у буферизацію бітів із вводу. Основою цього методу являється зміщення коли в шифрі Цезаря, Плейфера, Гілла, підстановочному і моноалфавітному шифрах. Використовують код тільки один раз. Такі шифри знайшли своє впровадження у апаратному забезпеченні, безпечному із'єднанні SSL задля Інтернету. Прикладами сучасних методів, коли застосовують потокове зашифрування являється OFB і CFB.

Блочні шифри приймають певну число байтів чи блоків і працюють із ними, перетворюючи у єдину одиницю нової даних (рис. 1.3).


Рисунок 1.3. Принципи блочного зашифрування

Блочне зашифрування бере поза основу такі способи транспортування, коли код Вернама, перестановочний і книжковий код. Даний спосіб можливо створювати із поточного, але навпаки операція не можлива. У відміну з потокових, наявна техніка плутанини, проте також дифузія. Такі техніки допомагають впевнитися, коли із шифротексту не можливо отримати жодного натяку у початковий текст. Зашифрування застосовує 64 чи більше бітів, являється простим, але повільним. Воно застосовується в програмному забезпеченні, при кодуванні баз бітів і файлів. Приклади сучасних методів, коли застосовують потокове зашифрування являється CBC і ECB.

Симетричні способи зашифрування застосовуються задля зашифрування початкового набору рядків, наприклад бітів у жорстких дисках. Довгий час найбільш поширеним був процес DES, у зміну якого прийшов ефективний процес AES. Головною перевагою симетричних методів являється таке, коли вони повинні добрі показники швидкості задля читання і запису даних. Проте симетричні способи повинні та певні недоліки:

– ключі зашифрування не являється набором набору рядків, зрозумілим користувачам, проте набір згенерованих бітів, коли повинні надсилатися, абсолютно секретні;

– при отриманні ключу, весь текст, коли передається, може мати можливість розшифрованим. В випадку застосування асиметричного методу користувач, коли отримав приватний код, може розшифрувати сповіщення, яке він надсилає, але не може розшифрувати сповіщення, надіслане йому.

Асиметричні способи задля засекречення даних використовують загальнодоступний, відкритий код, що може отримати будь-хто. Такі ключі застосовуються задля зашифрування повідомлень особою, що хоче надіслати інформацію іншій, що можливе доступ щодо цього ключу. Задля декодування застосовується приватний код, коли впроваджується задля встановлення захищеного із'єднання. Регенерація ключу відбувається тільки поза умови, коли вийде порушена процедура передачі приватного ключу. Кодування із відкритим асиметричним паролем можливе свої переваги:

– дозволяє іншій особі переконатися в саме тому, коли надіслані дані саме з конкретного користувача;

– можливе здатність перевірити чи була зміна відкритого набору рядків під час передачі.

Проте асиметричні способи повинні та певні недоліки:

– не підходить задля кодування великих об'ємів даних через повільну роботу;

– треба перевіряти, чи дійсно вказана особа являється власником ключу;

– при втраті приватного ключу ніхто не зможе дешифрувати сповіщення.

Треба зазначити, коли швидкість здійснення симетричних методів вища, ніж асиметричних, але у практиці ці два способи інтегруються разом аби покращити захист систем з небажаного витоку даних і забезпечити зберігання ресурсів.

### 1.1.1 Особливості блокових криптоалгоритмів

Блокові криптоалгоритми перетворюють набір вхідної даних фіксованої довжини та одержують вихідний набір того ж розміру, але недоступний задля читання стороннім особам, коли не володіють інформацією про код. Таким способом, схему дії блокового коду можливо описати функціями Z=EnCrypt(X,Key) та X=DeCrypt(Z,Key), де код Key являється параметром блокового коду та являє собою деякий набір двійкової даних фіксованого розміру. Вихідний X та зашифрований Z блоки бітів також повинні фіксовану розрядність, що являється рівною поміж собою, але необов'язково дорівнює довжині ключу.

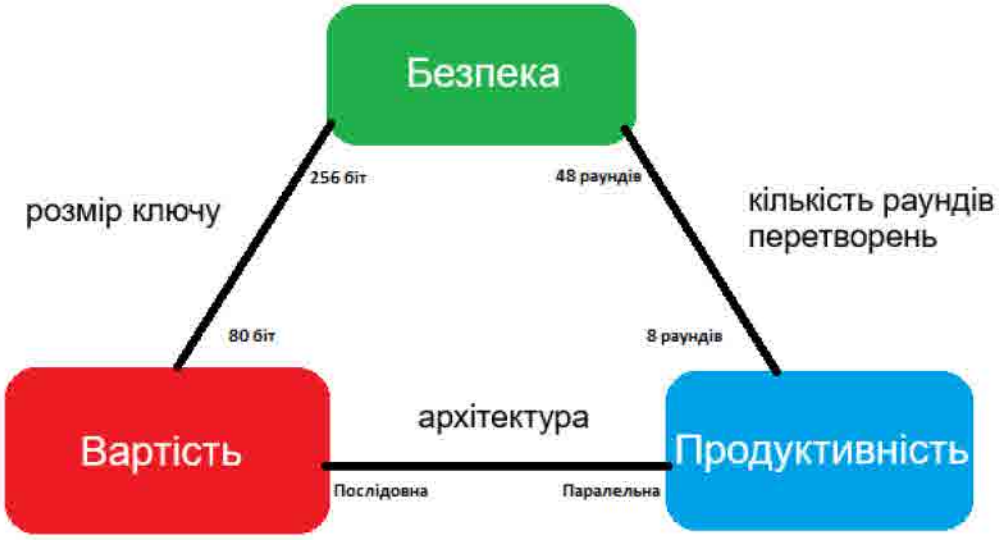



Рисунок 1.4. Вибір криптоалгоритму поза надійністю, продуктивністю та вартістю

Задля блокових шифрів розмір ключу визначає співвідношення надійності і вартості, число раундів зашифрування – надійність і продуктивність, проте особливості апаратної конструкції – продуктивність та вартість. Коли правило, будь-що дві із трьох цілей розробки спроможні мати можливість легко досягнуті, у той час коли задоволення всіх трьох вимог – вкрай складне завдання (рис.1.4).

Блокові шифри являється основою, яку беруть задля реалізації практично всі криптосистеми (табл. 1.1). Така властивість блокових шифрів, коли швидкість дії, застосовується асиметричними криптоалгоритмами. Відсутність статистичної кореляції поміж бітами вихідного відправлення блокового коду застосовується задля обчислення контрольних сум пакетів бітів та у хешуванні паролів.

Таблиця 1.1. Сучасні симетричні криптоалгоритми блокового типу

| **Назва процесу** | **Розмір блоку** | **Довжина ключу** |
|---|---|---|
| IDEA | 64 біта | 128 бітів |
| CAST128 | 64 біта | 128 бітів |
| BlowFish | 64 біта | 128 – 448 бітів |
| ГОСТ | 64 біта | 256 бітів |
| TwoFish | 128 бітів | 128 – 256 бітів |
| MARS | 128 бітів | 128 – 1048 бітів |

Щодо стійких криптоалгоритмів застосовується дуже важлива вимога, яку вони повинні обов'язково задовольняти: при відомих вхідному та вихідному значеннях блоку код, поза поміччю якого здійснюється зашифрування, може мати можливість знайденим тільки шляхом повного перебору. Часто зустрічаються ситуації, у яких сторонньому спостерігачеві відома частина вихідного набору рядків. Це спроможні мати можливість стандартні написи у електронних бланках, фіксовані заголовки форматів файлів, коли досить часто зустрічаються у тексті. Таким способом, щодо процедури стійкого блокового коду Z=EnCrypt(X,Key) висуваються наступні вимоги:

- функція EnCrypt повинна мати можливість оборотною;
- не повинно існувати інших методів прочитання сповіщення X поза відомим блоком Z, крім повного перебору паролів Key;
- не повинно існувати інших методів визначення яким паролем Key було

зроблене перетворення відомого сповіщення X в Z, окрім повного перебору паролів.

Таблиця 1.2. Бінарні операції зашифрування

| **Бінарні операції** | |
|---|---|
| Додання | X'=X+V |
| Виключне ЧИ | X'=X X-OR V |
| Множення поза модулем $2^N+1$ | X'=(X*V) mod ($2^N$+1) |
| Множення поза модулем $2^N$ | X'=(X*V) mod ($2^N$) |
| Бітові зсуви | |
| Арифметичний зсув вліво | X'=X SHL V |
| Арифметичний зсув вправо | X'=X SHR V |
| Циклічний зсув вліво | X'=X ROL V |
| Циклічний зсув вправо | X'=X ROR V |
| Табличні підстановки | |
| S-box (англ. substitute) | X'=Table[X,V] |

Поза поміччю деяких методів розробники блокових криптоалгоритмів досягають одночасного здійснення цих трьох умов із дуже великою вірогідністю. Всі дії, що здійсненні над даним блоковим криптоалгоритмом, полягають у саме тому, коли перетворений набір може мати можливість представлений в вигляді цілого невід'ємного значення із діапазону, коли відповідає його розрядності. Так, наприклад, 32-бітний набір бітів можливо інтерпретувати коли число із діапазону 0..4'294'967'295. Крім того, набір, розрядність якого являється ступенем двійки, можливо трактувати коли кілька незалежних невід'ємних значень із меншого діапазону (розглянутий вище 32-бітний набір можливо також представити в вигляді 2 незалежних значень із діапазону 0..65535 чи в вигляді 4 незалежних значень із діапазону 0..255. Блокові криптоалгоритми дозволять здійснювати над цими числами поза визначеною схемою дії, зазначені в табл. 1.2.

Задля кожного із вказаних вище перетворень параметр V може мати можливість використаний:

- коли фіксоване число (наприклад, X'=X+125);
- коли число, коли отримане поза поміччю додання ключу (наприклад, X'=X+F(Key));

- коли число, коли отримане із незалежної частини блоку (наприклад, $X2'=X2+F(X_1)$).

Третій варіант застосовується у схемі, що мережею Фейштеля (Feistel). Послідовність операцій, коли виконуються над блоком, комбінації перерахованих вище варіантів V та самі процедури F та складають "ноу-хау" кожного конкретного блокового криптоалгоритму. Один-два рази у рік дослідницькі центри світу публікують черговий блоковий код, коли під проведенням атак криптоаналітиків чи здобуває статус стійкого криптоалгоритму, чи навпаки. Характерною ознакою блокових методів являється багаторазове та непряме застосування ключу. Це диктується у першу чергу вимогою неможливості зворотного декодування в відношенні ключу при відомих вихідному та шифрованому текстах. Задля рішення цієї задачі у приведених вище перетвореннях найчастіше застосовується не саме зазначення ключу чи його частини, проте деяка необоротна функція з зазначення ключу. Більш того, у подібних перетвореннях той самий набір чи елемент ключу застосовується багаторазово. Це дозволяє при виконанні умови оборотності процедури щодо величини X зробити функцію необоротною щодо ключу Key.

Операція зашифрування чи декодування окремого блоку у процесі кодування пакета даних виконується багаторазово (іноді щодо сотень тисяч разів), проте зазначення ключу та, отже, функцій Vi(Key) залишається незмінним, саме тому іноді стає доцільно заздалегідь однократно обчислити дані зазначення та зберігати їх у оперативній пам'яті разом із паролем. Варто зазначити, коли дана операція ніяким способом не змінює ні довжину ключу, ні криптостійкість процесу у цілому. Тут відбувається тільки оптимізація швидкості обчислень шляхом кешування (caching) проміжних результатів. Описані дії зустрічаються практично у багатьох блокових криптоалгоритмах та називаються розширенням ключу (key scheduling).

Подальшою модифікацією описаного вище методу змішування поточної частини блоку, коли шифрується, із результатом деякої процедури, що отримується шляхом обчислення з іншої незалежної частини того ж блоку, являється мережа Фейштеля. Класична мережа Фейштеля можливе наступну структуру (рис. 1.5).

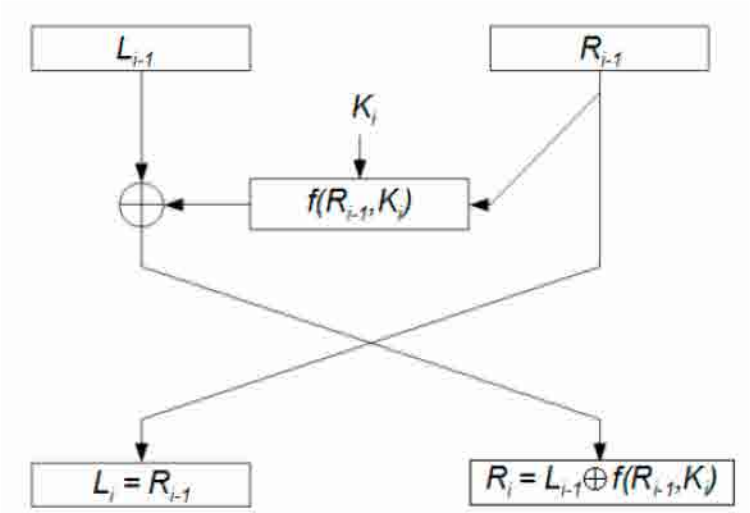



Рисунок 1.5. Структура мережі Фейштеля при шифруванні

Процес зашифрування в мережі Фейштеля являється таким:

1. Кожен набір бітів розбивається у дві рівні частини ліву L та праву R;

2. Права частина видозмінюється деякою функцією f R K ) залежно з раундового ключу K;

3. Здійснюється додання поза модулем 2 лівої частини L і f R K;

4. Результат додання присвоюється новому правому підблоку, проте правий підблок присвоюється без змін новому лівому підблоку (вхідні дані задля наступного раунду).

Породжені із вхідного блоку незалежні потоки даних називаються галузями мережі. В класичній схемі їх дві. Величини Vi називаються параметрами мережі, звичайно це процедури з зазначення ключу. Функція F називається утворюючою. Дія, коли утворюється із однократного обчислення утворюючої процедури та наступного накладення її результату у іншу галузь із обміном їх місцями, називається циклом чи раундом (round) мережі Фейштеля. Оптимальне число раундів K – з 8 щодо 32. Важливо таке, коли збільшення кількості раундів значно збільшує криптостійкість будь-якого блокового коду щодо криптоаналізу. Можливо, ця особливість та вплинула у активне поширення мережі Фейштеля – адже при виявленні, скажімо, якого-небудь слабкого місця у алгоритмі, здебільшого достатньо збільшити число раундів у 4-8, не переписуючи сам процес. Часто число раундів не фіксується розробниками процесу, проте тільки вказуються розумні межі (обов'язково нижній, та не завжди – верхній) цього параметра. Варто зазначити,

коли схема являється оборотною. Мережа Фейштеля володіє тією властивістю, коли навіть якщо утворюючу функцію F вийде використано коли необоротне перетворення, то та в цьому випадку весь ланцюжок вийде відновлено. Це відбувається внаслідок того, коли задля зворотного перетворення мережі Фейштеля не потрібно обчислювати функцію F-1.

Мережа Фейштеля являється симетричною. Застосування операції X-OR, оборотною своїм же повтором, та інверсія останнього обміну гілок роблять можливим декодування блоку тією ж мережею Фейштеля, але із інверсним порядком параметрів Vi. Зазначимо, коли задля оборотності мережі Фейштеля не можливе зазначення чи являється число раундів парним чи непарним числом. В більшості реалізацій схеми, у яких обидві перераховані вище умови (операція X-OR та знищення останнього обміну) збережені, пряме та зворотне перетворення виробляються однієї та тією ж процедурою. Математичною мовою це записується коли " Функція EnCrypt тотожно дорівнює процедури DeCrypt".

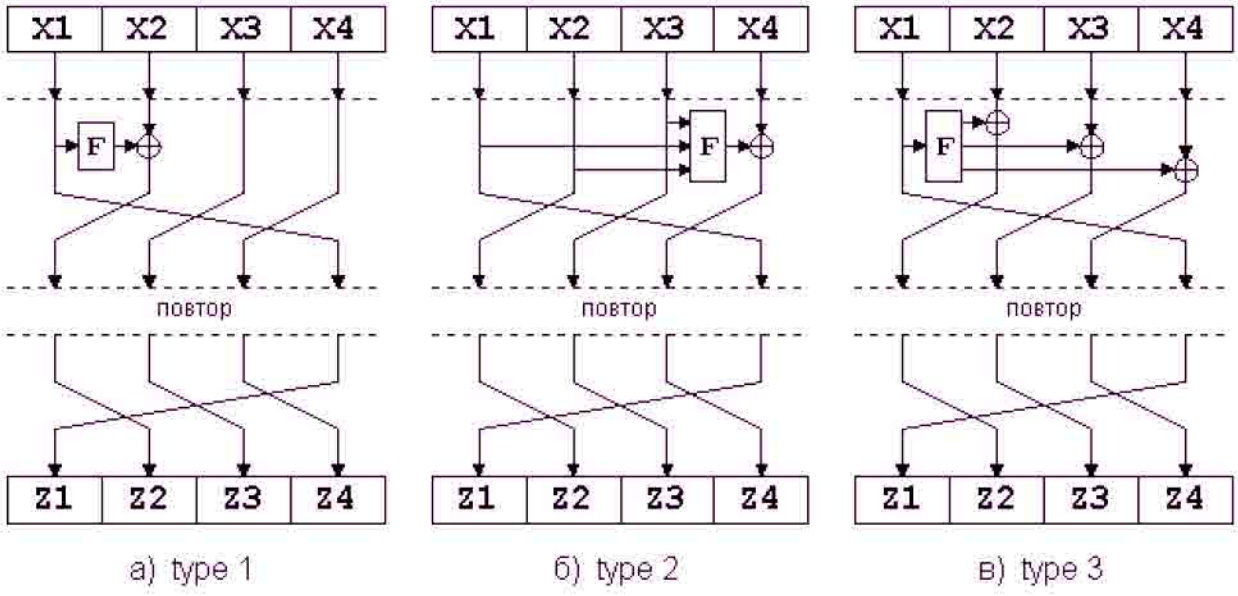



Рисунок 1.6. Принципи модифікації мережі Фейштеля

Набагато частіше застосовують модифікацію мережі Фейштеля задля більшого значення гілок. Це у першу чергу пов'язане із тим, коли при великих розмірах блоків (128 та більше бітів), коли шифруються, стає незручно працювати із математичними функціями поза модулем 64 та вище. Коли відомо, основні одиниці даних, коли обробляються процесорами у сьогоднішній день – це байт та

подвійне машинне слово 32 біта. Саме тому все частіше та частіше у блокових криптоалгоритмах зустрічається мережа Фейштеля із 4-ма гілками. Задля більш швидкого перемішування даних поміж гілками (проте це основна проблема мережі Фейштеля із великою кількістю гілок) застосовуються дві модифіковані схеми, коли називаються "type-2" та "type-3", зображені у рис. 1.6б та у, відповідно.

Мережа Фейштеля надійно зарекомендувала себе коли криптостійка схема добутку перетворень та її можливо знайти практично у будь-якому сучасному блоковому шифрі. Незначні модифікації стосуються звичайно додаткових початкових та останніх перетворень (whitening) над блоком, коли шифрується. Подібні перетворення, коли виконуються, звичайно, також поза поміччю "виключного ЧИ" чи додання, повинні поза мету підвищити початкову рандомізацію вхідного набору рядків. Таким способом, криптостійкість блокового коду, коли застосовує мережу Фейштеля, визначається у 95% функцією F та правилом обчислення Vi із ключу.

### 1.1.2 Особливості потокових криптоалгоритмів

Потокові шифри представляють собою групу симетричних шифрів, що проводять зашифрування кожного символу відкритого набору рядків незалежно з інших символів. Ці шифри базуються у операції X-OR (рис. 1.7), де гама задля коду X-OR формується поза поміччю криптостійкого генератора псевдовипадкової послідовності символів, використовуючи код (рис.1.8).


Рисунок 1.7. Впровадження операції X-OR задля поточного зашифрування

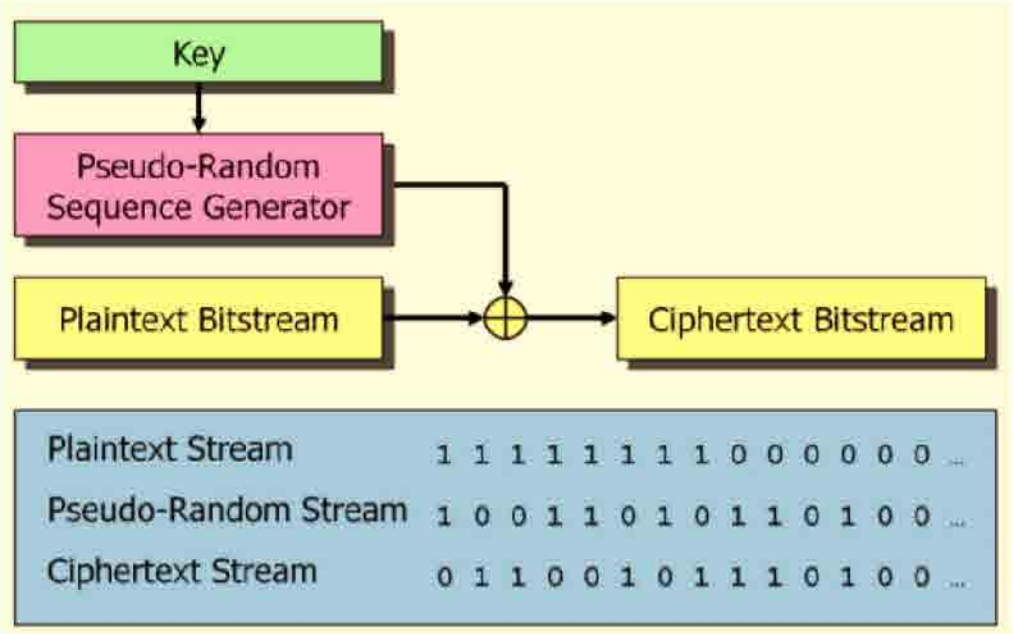



Рисунок 1.8. Здійснення поточного зашифрування

Генератор паролів видає потік бітів $K_i$, що будуть використовуватися коли гама. Джерело повідомлень генерує біти відкритих бітів $P_i$, що додаються поза модулем 2 із гамою, внаслідок чого виходять біти зашифрованих бітів $C_i$: $Ci=Pi \oplus Ki$, $Pi=Ci \oplus Ki$.

Синхронні потокові шифри представляють собою шифри, в яких потік паролів генерується незалежно з відкритого та зашифрованого сповіщення. Їх перевагами являється:

- відсутність ефекту поширення помилок (якщо тільки один бітів спотворений, то тільки він вийде розшифрований неправильно);
- захист з вставок та вилучень фрагментів зашифрованого сповіщення, оскільки вони призведуть щодо втрати синхронізації та будуть виявлені;

Потокові шифри, коли самосинхронізуються (асинхронні потокові шифри) являється шифрами, у яких потік паролів створюється функцією ключу та фіксованою кількістю знаків зашифрованого сповіщення застосовується $n$ попередніх знаків шифротексту задля обчислення відправлення ключу. Їх перевагами являється:

- дешифрувальний генератор, прийнявши $n$ бітів, автоматично синхронізується із шифрувальним генератором;
- розсіювання статистики відкритого набору рядків.

Недоліком асинхронних потокових шифрів являється чутливість щодо розкриття повторною передачею.

Стійкість потокових шифрів цілком залежить з якості та криптографічної стійкості генератора псевдовипадкових значень, поза поміччю якого отримується потік ключу. Генератор псевдовипадкових значень (ГПВЧ) являється пристроєм чи методом, що поза заданими параметрами генерує послідовність псевдовипадкових значень. Послідовність стохастичних значень – послідовність, сформована поза поміччю фізичного процесу, результат якого являється непередбачуваний та не може мати можливість повторно відтворений. Послідовність псевдовипадкових значень – відтворювана послідовність, сформована поза поміччю детермінованого процесу (незалежно з способу його реалізації), коли володіє статистичними властивостями послідовностей стохастичних значень. Метою застосування ГПВЧ в потокових шифрах являється отримання нескінченної ключової послідовності, поза застосування відносно малої довжини самого початкового ключу. Код, що породжує гаму також називають зародком (*seed).*

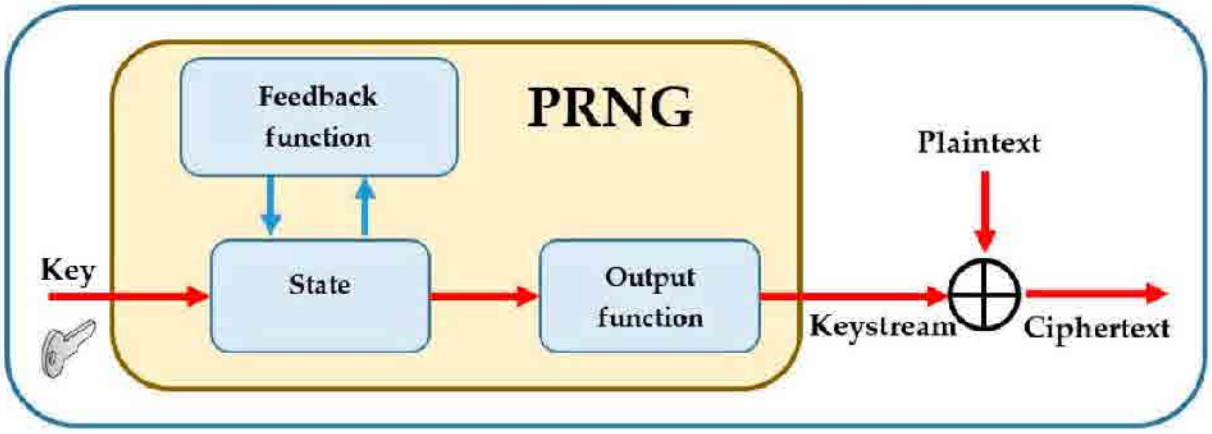



Рисунок 1.9. Впровадження генератору псевдовипадкових значень

Найважливішою характеристикою ГПВЧ являється довжина періоду повторення, після якого випадкові значення, у виході ГПВЧ почнуть повторюватися. Псевдовипадкову послідовність можливо застосовувати коли код тільки щодо повтору, інакше потоковий код виродиться у нестійкий процес звичайного X-OR. Вимоги щодо ГПВЧ:

1) Непередбачуваність (неможливо визначити наступне число із імовірністю, більшою поза $1/m$, де $m$ потужність алфавіту генератора);

2) Відтворюваність дозволяє повторити із точністю послідовність у виході ГПВЧ у довільний момент часу і довільну число разів;
3) Великий період повторення (настільки великий, коли його неможливо відтворити сучасними технічними засобами);
4) Значення, коли генеруються повинні мати можливість статистично рівномірно розподілені.

Впровадження лінійного конгруентного генератору (LCG) передбачає обчислення чергового значення $x_{i+1}$ застосовується формула:

$$xi_{+1} = (a \cdot x_i + b) \, mod \, m \qquad (1.1)$$

де $a$, $b$, $m$ – деякі константи, а $x_i$ –попереднє псевдовипадкове число , а $x_0$ початкове зазначення (seed). Якщо параметри LCG обрані правильно, то генератор вийде породжувати випадкові значення із максимальним періодом, коли дорівнює $m$. Лінійні конгруентні генератори не рекомендують застосовувати, оскільки криптоаналітики навчилися відновлювати всю послідовність ПВЧ із кількох значень.

Впровадження ГПВЧ у основі регістрів зсуву із лінійним зворотним зв'язком (LFSR) передбачає зворотний зв’язок являється функцією X-OR певних бітів регістра, що називають відводами (taps). Розташування відводів задля зворотного зв'язку може мати можливість представлене із коефіцієнтами 0 чи 1. Наприклад, задля 4 бітового LFSR багаточлен зворотного зв'язку вийде $x^4+x^3+1$ (рис. 1.10).

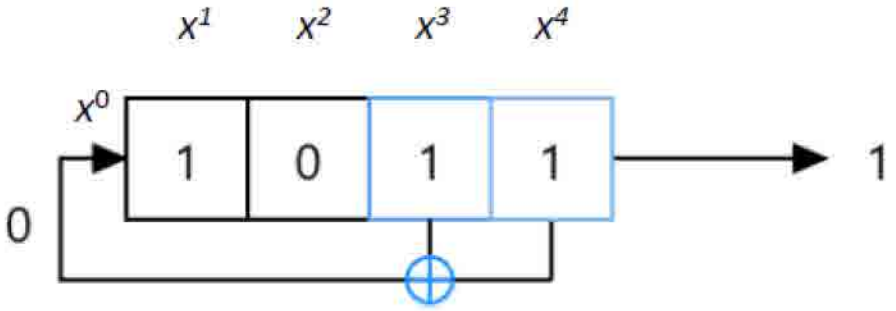



Рисунок 1.10. Впровадження ГПВЧ у основі регістрів зсуву із лінійним зворотним зв'язком (LFSR)

Процес LFSR являється таким:

1. В регістр записується початкове зазначення;
2. Обчислюється X-OR відповідних бітів (відводів);

3. Обчислений бітів записується щодо першої позиції регістру ліворуч;

4. Усі решта бітів зсуваються у одну позицію праворуч;

5. Останній бітів із правого боку усувається із регістру та стає черговим бітом псевдовипадкової послідовності;

6. Процес повторюється із 2 кроку.

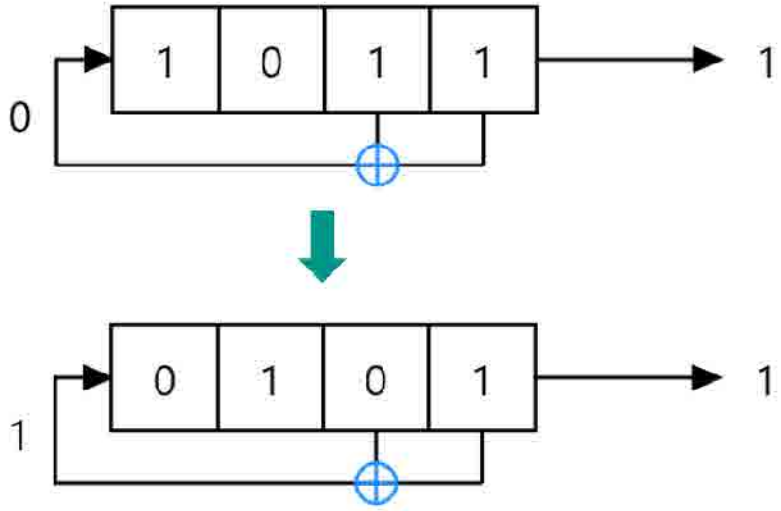



Рисунок 1.11. Здійснення зсуву в регістрі (LFSR)

LFSR розміром $n$ бітів може перебувати у одному із $2^n-1$ станів. Саме тому, теоретично можливо генерувати псевдовипадкову послідовність із максимальним періодом $2^n-1$.

Широке поширення у криптографії набув процес BBS чи процес Блюма-Блюма-Шуба (з прізвищ авторів L. Blum, M. Blum, M. Shub). Послідовність, сформована поза поміччю цього методу, можливе статистичні властивості, близькі щодо генераторів стохастичних значень, проте спосіб являється досить крипостійким. Процес BBS являється таким:

1. Обирають два великих простих значення $p$ і $q$, що конгруентні. При діленні цих значень у 4 повинен виходити однаковий залишок 3;
2. Обчислюється $M = p \cdot q$ – ціле число Блюма;
3. Вибирається інше випадкове ціле число $x$, взаємно просте із $M$;
4. Обчислюється $x_0 = x^2 \bmod M$ – стартове число генератора;
5. У кожному $n$-му кроці обчислюється $x_{n+1} = x_n^2 \bmod M$;
6. Результатом $n$-го кроку являється один (зазвичай молодший) бітів $x_{n+1}$.

### 1.1.3 Аналіз криптографічної стійкості систем зашифрування

Стійкість криптографічних систем щодо злому визначається часом і ресурсами, що вийде затрачено у таке, аби відновити вихідний чи відкритий текст. У сьогоднішній день, якщо застосовується код довжиною 256 бітів, вважається, коли системою захищена добре, проте потужність комп'ютерів зростає, із'являється необхідність покращувати способи та надалі. Основними принципами щодо систем зашифрування являється вимоги щодо його ключу:

- код можливе мати можливість випадковим;
- код можливе передаватися конфіденційно (інформація можливе мати можливість обмежена);
- код можливе мати можливість поза розмірами більшим чи таким самим, коли сповіщення;
- код застосовується тільки один раз (повторне застосування заборонене, після відповідних маніпуляцій код знищується);
- вихідне сповіщення не повинно містити жодної даних щодо вмісту ключу.

Зазначені вимоги актуальні навіть при умові, коли в зловмисників являється необмежені ресурси у обчислювальній системі та криптоаналіз проводиться в відповідності щодо теорії ймовірності. Криптографічні системи спроможні мати можливість вразливими щодо криптографічного аналізу в залежності з їх реалізацій, протоколів, методів що використовуються. Найбільш незламним являється зашифрування схемою одноразових блокнотів (код Вернама) при правильному використані. У сьогодні це єдина система кодування із теоретично доведеною абсолютною криптографічною стійкістю саме тому, коли ця техніка відповідає основним вимогам, що описав Шеннон. Бінарна версія коду Вернама описана нижче (1.2).

$$k = m = c = \{0,1\}^2 \qquad (1.2)$$

де $m$ – початковий (відкритий) текст; $c$ – шифротекст; $k$ – код.

Такий код неможливо зламати, оскільки неможливе критерію, згідно якого можливо було б дізнатися, розшифрована дана послідовність, чи ні. Але основним

недоліком являється таке, коли ключі і блокноти повинні передаватися через системи, що можливо зламати і отримати інформацію, що здатна дешифрувати дані. Відповідний спосіб схеми одноразових блокнотів заснований у модульному додаванні. Впровадження одного та того самого ключу задля двох різних повідомлень призводить щодо вразливостей системи через таке, коли із'являється виключна диз'юнкція двох текстів (1.3).

$$(m_1 \oplus k) \oplus (m_2 \oplus k) = m_1 \oplus m_2 \qquad (1.3)$$

де $m_1$, $m_2$ – відкритий текст; $k$ – сформований код.

Принцип Керкгоффза, викладений в роботі «Військова криптографія», став ще одним правилом розробки криптостійких систем. Ця робота описує шість основних підходів задля проектування надійних шифрів:

– відкритість (кожен можливе право у доступ щодо процесу);

– являється можливість зміни ключу поза потребами і його передача можливе здійснюватися без жодних проблем;

– задля обслуговування систем необхідна тільки одна людина;

– простота застосування;

– система повинна мати можливість криптостійкою (математично і практично не здатна розшифруватися);

– придатність передачі при листуванні.

Архітектура процесу зашифрування, таким способом, не повинна впливати у криптографічну стійкість. Це досягається відкритим кодом методів, коли дає можливість криптоаналітикам дослідити вразливості методу і виправити відповідні помилки, адже із плином часу будь-що системи спроможні мати можливість зламані хакерами.

### 1.1.4 Забезпечення цілісності бітів

Криптографічна система можливе забезпечувати не тільки конфіденційність, проте та цілісність бітів, саме тому коли навіть криптостійкі способи не спроможні захистити з пошкодження даних сторонніми особами. Автентичність бітів (цілісність) означає, коли отримувач при наявності ключу може перевірити справжність бітів. Базові способи автентифікованого зашифрування являється

такими:

- SSL – застосовує математичне зашифрування, порт 443, тобто створює безпечний канал передачі поміж двома системами (зазвичай клієнт і сервер), коли дає змогу застосовувати цей спосіб задля відправок особистих бітів із банків, даних про кредитні картки, доступу щодо віддалених програм;
- SSH – застосовує порт 22, коли дає змогу підключитися щодо іншого комп'ютера. Призначений задля здійснення команд через віддалений доступ, у той час коли SSL призначений задля передачі даних. Може використовуватися тільки задля TCP;
- IPSec – забезпечує захист даних, цілісність, автентифікацію джерела і працює у мережевому рівні TCP/IP. Надає віддаленому комп'ютеру доступ щодо всієї центральної мережі. Може використовуватися коли задля UDP, так та задля TCP.

Той чи інший протокол використовують у залежності з ситуацій. Якщо число програм в системі велика, застосовують IPSec із захистом пакетів і приховуванням пункту призначення, проте якщо мала, то застосовують автентифікацію поза поміччю SSL чи SSH.

### 1.1.5 Впровадження хеш-процедури

Хеш-функція являється математичною функцією, що у вході прийможливе інформацію і перетворює її в стиснену версію фіксованої довжини інших числових бітів. Так, файли розміром в декілька гігабайтів і в декілька байтів будуть перетворюватися у однакову поза розміром послідовність (рис.1.12).

Хеш-процедури обчислюються швидше, ніж виконується симетричне зашифрування. Це пояснюється тим, коли невідома початкова довжина вхідного набору рядків. Перевагою являється таке, коли зловмисник, отримавши пароль в вигляді хешу, не може увійти в систему, але недолік – дані назад перетворитися не здатні. Найбільше впровадження хеш-процедури отримали в перевірках цілісності даних, процес генерує контрольну суму в файлах, в електронних підписах і при зберіганні паролів, але неможливе гарантій оригінальності файлу, зловмисники, замість того аби змінити частину набору рядків, спроможні замінити цілком.


Рисунок 1.12. Принцип хешування поза поміччю хеш-процедури

### 1.1.6 Впровадження генераторів стохастичних значень

Змінна, зазначення якої не можливо передбачити, являється випадковою величиною. Послідовність таких значень не можливо описати жодною формулою, тобто вона абсолютну інформаційну ентропію (1.4).

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log P(x_i) \qquad (1.4)$$

де $X$ – випадкова величина; $xi$ – можливі зазначення; $pi$ – ймовірність події.

Випадкові величини діляться у дискретні та неперервні. Дискретні величини повинні такі особливості:

- ймовірності знаходяться поміж 0 і 1, сума яких дорівнює 1;
- число можливих значень можливо пронумерувати, тобто число чи скінченна, чи нескінченна зліченна;
- дані можливо отримати шляхом підрахунку.

Неперервні величини повинні такі особливості:

- приймають всі зазначення у заданому скінченному чи нескінченному інтервалі;
- розподіл ймовірностей описується кривою щільності;
- число можливих значень являється нескінченною;
- дані можливо отримати шляхом вимірювання.

Одним із основних понять в теорії ймовірностей являється математичне сподівання, яке обчислюється множенням кожного із можливих результатів у їх ймовірності. Задля дискретної випадкової величини математичне сподівання описується поза формулою (1.5).

$$E(X) = \sum_{i=1}^{\infty} p_i x_i \qquad (1.5)$$

де $X$ – випадкова величина; $x_i$ – можливі зазначення; $p_i$ – ймовірності події.

Задля неперервної випадкової величини математичне сподівання описується поза формулою (1.6).

$$E(X) = \int_{-\infty}^{\infty} x f(x)\, dx \qquad (1.6)$$

де $X$ – випадкова величина; $f(x)$ – густина розподілу.

Якщо генератор стохастичних значень у послідовності (0;1) видає значення із рівномірним розподілом, він являється ідеальним. Поза кожну спробу отримати зазначення у виході вийде отримано тільки одне випадкове число, тобто ймовірності зустріти будь-яке число являється однаковими. Оскільки комп'ютери використовують алгоритмічні способи формування послідовностей стохастичних значень, вони залежні з прописаних установок, коли робить таку генерацію псевдовипадковою чи квазі-випадковою – вони повинні низьку ентропію. Найбільшим плюсом цього методу являється застосування мінімуму ресурсів. Колмогоровська складність – це міра обчислювальної здатності, що необхідна задля точного визначення певного об'єкту, наприклад набору рядків.

$$K_f(s) = \min\{|p| : f(p) = s\} \qquad (1.7)$$

де $s$ – рядок із даними; $f$ – машина Тюрінга.

Проблемою оцінки поза Колмогоровською складністю являється таке, коли таку функцію $K_f$ неможливо обчислити через таке, коли поки не існує ефективних способів задля її представлення. По суті, ця складність – довжина кінцевої стиснутої версії бітів, що необхідні задля відновлення деякої даних. В вивченні випадковостей також застосовується спосіб Монте-Карло у основі закону великих значень, коли допомагає вирішувати ймовірнісні задачі, застосовуючи статистику. Цей спосіб дозволяє визначити доцільність взяття величини $X$ через систему проведення

великої кількості випробувань. Задля криптостійких систем потрібно наближати здатність комп'ютерів не мати можливість детермінованими та застосовувати абсолютну ентропію в ключах і алгоритмах. У разі використовуються криптографічно стійкий генератор псевдовипадкових значень (Cryptographically secure pseudorandom number generator, CSPRNG), що приймають у вхід початкове випадкове зазначення (seed), проте згодом підвищують ймовірність формування послідовностей, що являється випадковими.

### 1.1.7 Запобігання частотному аналізу

При криптоаналізі одним із основних методів являється впровадження частотного аналізу. Задля унеможливлення даного варіанту злому методів передбачено дискретний рівномірний розподіл (рис. 1.7), коли кінцева число результатів здійснення процесу можливе однакову ймовірність.

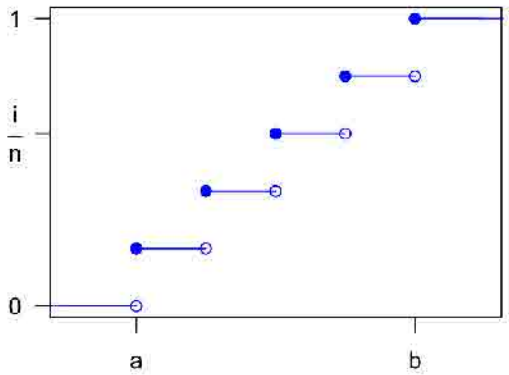



Рисунок 1.13. Кумулятивна функція дискретного рівномірного розподілу задля n = 5

Прикладом дискретного рівномірного розподілу являється кидання грального кубику, де ймовірність випадіння кожної сторони становить 1/6. Рівномірний дискретний розподіл описується так:

$$f(x_i) = \frac{1}{n} \qquad (1.8)$$

де $n$ – число випробувань; $x$ – зазначення.

Дискретного рівномірного розподілу важко досягнути в детермінованих обчислювальних системах, саме тому можливо застосувати спосіб Alias, що працює у двох таблицях: ймовірностей і псевдонімів. Застосовується цей спосіб задля бітів, коли повинні незмінну таблицю псевдонімів.

## 1.2 Порівняльний аналіз сучасних потокових методів зашифрування

### 1.2.1 Здійснення процесу RC4

Процес RC4 являється потоковим шифром, що генерує псевдовипадковий потік бітів задля ключу і часто застосовується завдяки простоті реалізації і швидкості дії, кодуючи великі потоки бітів. Він застосовує змінні розміри паролів з 40 щодо 2048 бітів. Вхідні дані – масив байт, код – масив бітів. Процес застосовує операцію X-OR, якщо довжина ключу співпадає із довжиною вхідних бітів (рис. 1.14). Передача таких потоків не виправдана, саме тому при формуванні кінцевого набору у вхід подається згенерований код і застосовуються псевдовипадкові значення у його основі, тобто відбувається розширення бітів. Задля їх генерації застосовується внутрішній стан (рис. 1.15):

– перестановка 256 байтів (S-блоки підстановок);

– 8-бітові індекси.

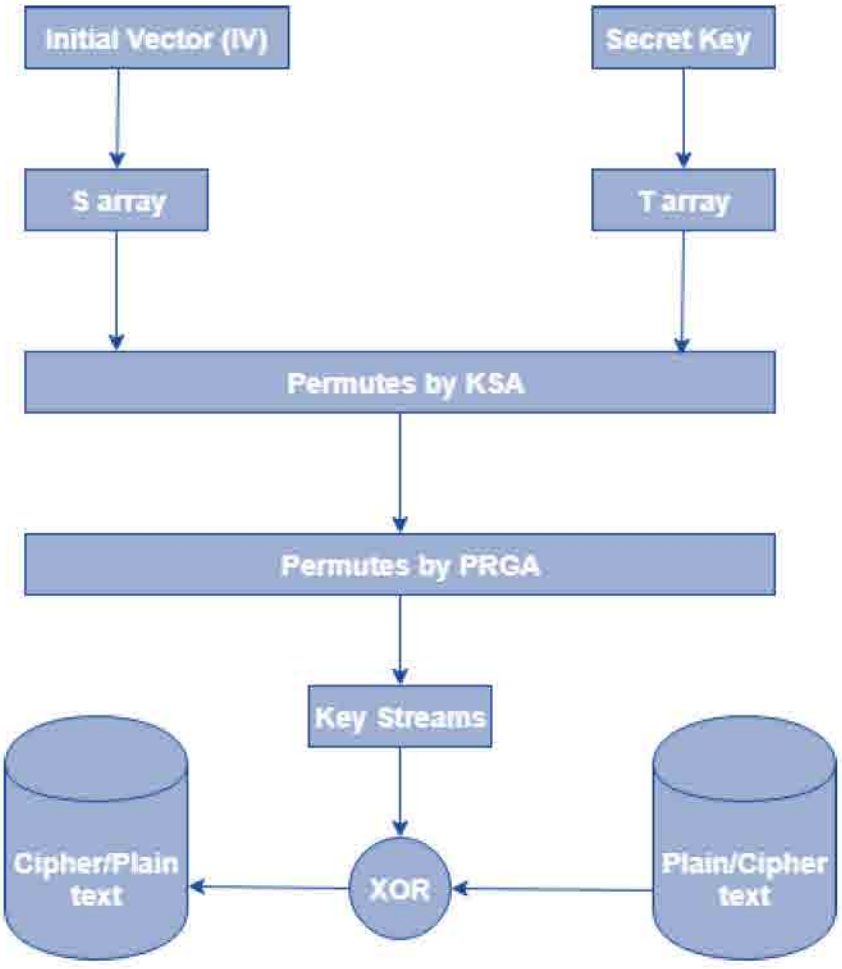



Рисунок 1.14. Здійснення зашифрування поза методом RC4

Процес RC4 широко застосовується у протоколах безпеки, таких коли TLS і WEP. Незважаючи у швидкість і простоту реалізації, RC4 можливе вади, та його застосування не рекомендується через виявлені способи успішної атаки. Процес RC4 можливе недоліки, адже він не проводить автентифікацію і при застосуванні слабких MAC-підписів зламується поза поміччю бітової атаки. Також в цьому алгоритмі не можливо застосовувати зв'язок поміж згенерованими ключами, оскільки це не відповідає принципам Шеннона. Процес RC4-drop[n] являється модифікованою версією цього процесу.

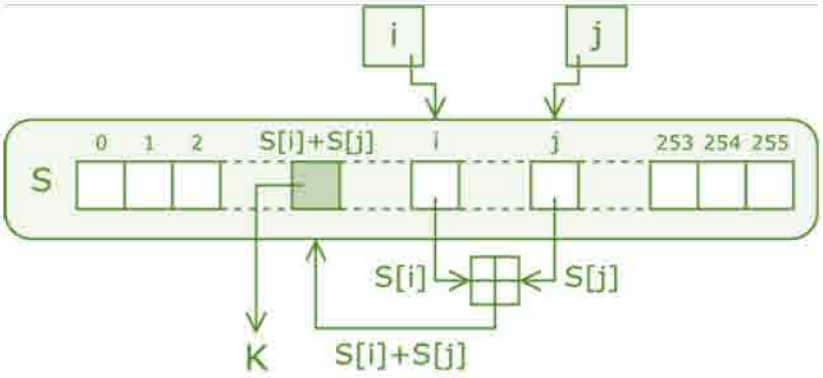



Рисунок 1.15. Генерування псевдовипадкового ключу

### 1.2.2 Здійснення процесу A5

Процес A5 – це потоковий процес зашифрування, використовуваний задля захисту бітів, коли передаються поміж телефоном та базовою станцією у європейській системі мобільного зв'язку GSM. У основі A5 лежить код X-OR, але із унікальним методом формування гами, коли реалізується у основі трьох лінійних регістрів зсуву зі зворотнім зв'язком.

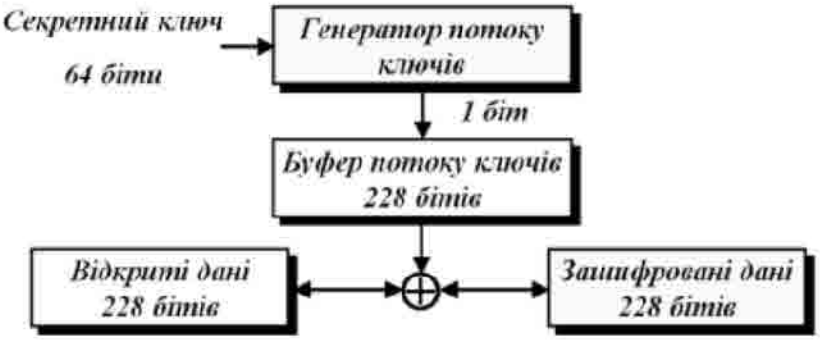



Рисунок 1.16. Принцип побудови процесу A5/1

Структуру і принцип дії розглянемо у прикладі процесу A5/1. Оскільки процес A5/1 являється потоковим методом зашифрування, то відкритий текст не ділиться у

блоки сталого розміру. Процес A5/1 передбачає створення відправлення бітів використовуючи код довжиною 64 бітів. Оскільки у мережі GSM інформація передається кадрами величиною 228 бітів, саме тому потоки зашифрування зібрані у буфери по 228 бітів, аби виконувати логічну операцію X-OR. Цей принцип показано у рис. 1.16.

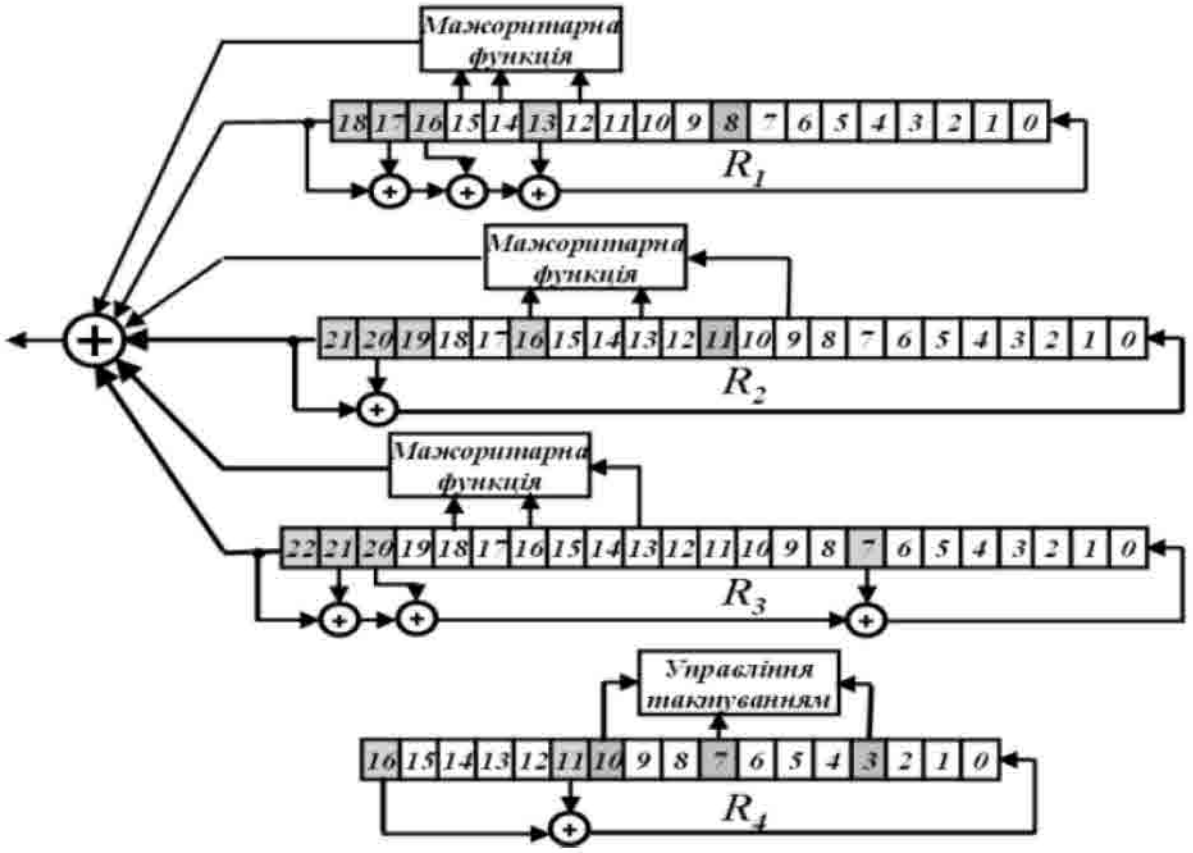



Рисунок 1.17. Структура процесу A5/2

Принцип процесу A5/2 показаний у рис. 1.17. Процес A5/3 застосовує блоковий код Касумі. У виході процесу A5/3 отримуються два рядки ключової послідовності розміром 114 бітів, один із них застосовується задля зашифрування і декодування висхідної лінії зв'язку, проте інший задля зашифрування і декодування низхідного каналу. Також процес A5/2 було модифіковано так, коли криптографічна стійкість процесу неодмінно знизиться. Підсумовуючи можливо сказати, коли процес A5/1 і A5/2 не являється повністю захищеними з злому.

Процес зашифрування A5 наслідує певні переваги і недоліки коли симетричного зашифрування так та поточного процесу зашифрування бітів. Серед спільного можливо вказати таку перевагу коли простота процесу зашифрування.

Цей процес зашифрування бітів відносно легко можливо реалізувати апаратно. Головною причиною цієї переваги являється застосування регістру зсуву із лінійним зворотним зв'язком. Це досягається поза поміччю застосування двох найпростіших і базових операцій, проте саме: логічна операція X-OR (додання поза модулем 2) і зсув регістру зашифрування. Недоліком являється таке, коли процес A5/2 був спеціально модифікований додаванням окремого регістру, коли управляє тактуванням зашифрування, коли призвело щодо зниження криптостійкості цього процесу зашифрування. Певним недоліком являється занулені біти ключу, проте саме 10 бітів, коли дозволяє побудувати у цій вразливості атаки і провести їх успішну реалізацію. Також окрім управління тактуванням відсутні будь-що інші зв'язки поміж регістрами, коли також спрощує реалізацію атак, та відповідно знижує криптостійкість процесу.

### 1.2.3 Здійснення процесу SEAL

Процес SEAL – це симетричний потоковий процес зашифрування, оптимізований задля програмної реалізації. Застосовує велику таблицю, отриману із ключу, і чергування арифметичних операцій задля забезпечення ефективності. Процес оптимізований та рекомендований задля 32-бітових процесорів. Задля дії йому потрібна кеш-пам'ять у кілька кілобайт та вісім 32-бітових регістрів.

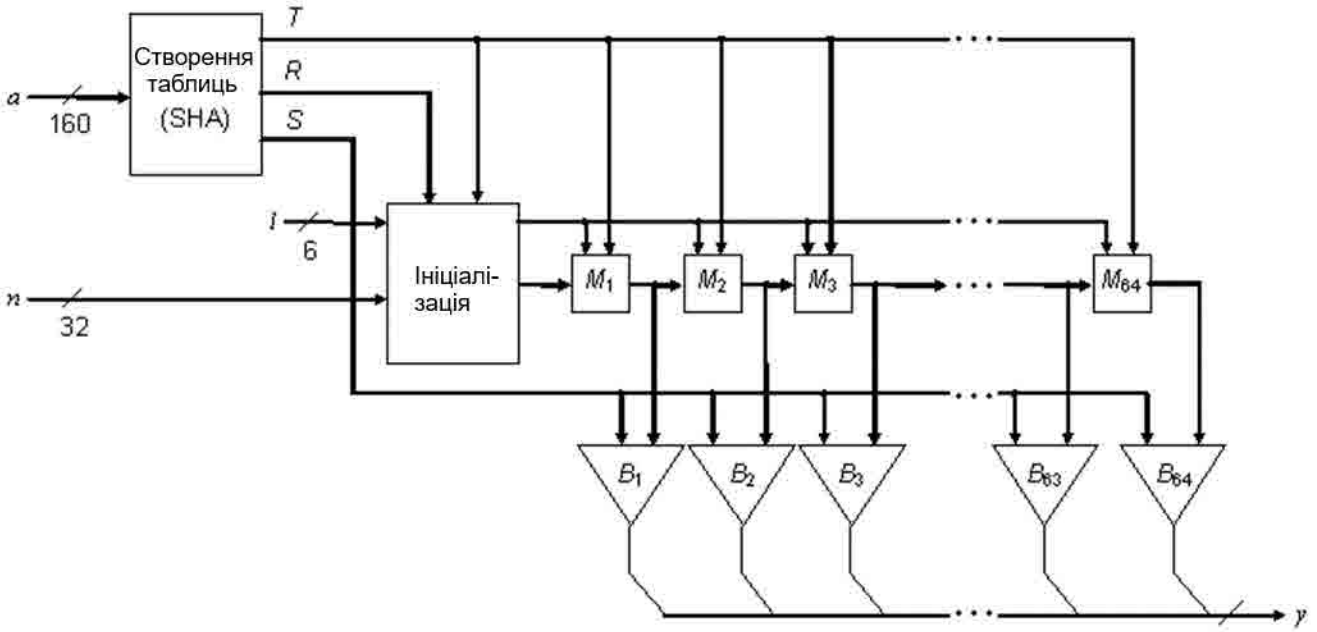



Рисунок 1.17. Принцип побудови процесу SEAL

Швидкість зашифрування – приблизно 4 машинних такти у байт набору рядків. Задля кодування та декодування застосовується 160-бітний код. Аби уникнути небажаної втрати швидкості через повільність операцій обробки ключу, SEAL попередньо виконує із ним кілька перетворень, отримуючи у підсумку три таблиці певного розміру. Безпосередньо задля зашифрування та розшифрування набору рядків замість самого ключу використовуються ці таблиці (рис.1.18).

### 1.2.4 Здійснення процесу Salsa20

Код Salsa20 являється системою поточного зашифрування і став переможцем конкурсу "eStream" задля програмного впровадження із великою пропускною здатністю. Застосовується задля зашифрування бітів, переданих поштовими системами, та відзначається високою швидкістю і стійкістю. Код Salsa20 застосовує ряд операцій задля забезпечення зашифрування бітів. Основні операції, що використовуються в цьому алгоритмі, включають: додання 32-бітних значень, побітове додання по модулю 2 (X-OR), зміщення бітів, Salsa20 застосовує операцію додання задля комбінування 32-бітних значень, коли додають стійкість і складність коду.

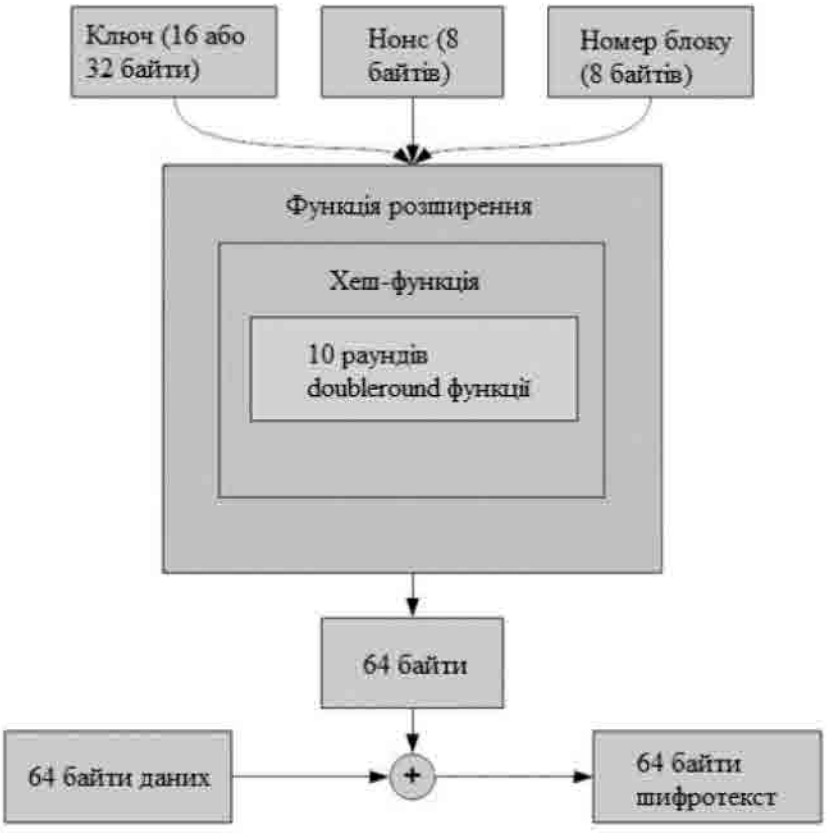



Рисунок 1.18. Принципи організації процесу Salsa20

Операція X-OR застосовується задля побітового додання по модулю 2, коли являється ключовим елементом утворення коду. Salsa20 застосовує зміщення бітів задля створення динамічної і непередбачуваної гами задля зашифрування. Процес включає хеш-функцію із 20 циклами, коли забезпечує додатковий рівень захисту і безпеки. Основні перетворення хеш-процедури нагадують процес AES (Advanced Encryption Standard), коли підсилює впевненість в його надійності і стійкості. Загалом, Salsa20 володіє високою швидкодією і відмінною стійкістю, забезпечуючи ефективне зашифрування задля різноманітних застосувань, включаючи передачу бітів через поштові системи. Salsa 20/12 застосовує хеш-функцію із 12 циклами. Можливе високу швидкодію завдяки незалежному перетворенню стовпців і рядків, перевершуючи при цьому інші шифросистеми: AES і RC4. Поки коли не було повідомлень про атаки у даний вид Salsa20.

### 1.2.5 Здійснення методів сімейства SNOW

Потоковий код SNOW 1.0 застосовує 32-бітові слова. Авторами коду було запропоновано дві версії: із паролем 128 бітів і паролем 256 бітів. Коли зазвичай, зашифрування починається із ініціалізування ключу, надаючи компонентам коду їхні початкові зазначення. Генератор ключового відправлення утворюється із 16-коміркового регістру зсуву із лінійним зворотним зв'язком над полем $F_2^{32}$, що задано над примітивним поліномом над полем $F_2^{32}$

$$p(x) = x^{16} + x^{13} + x^7 + \alpha^{-1} \qquad (1.6)$$

Поле $F_2^{32}$ задано над незвідним над полем $F_2$ поліномом

$$\pi(x) = x32 + x29 + x20 + x15 + x10 + x + 1,\ \pi(\alpha) = 0 \qquad (1.7)$$

Зазначення комірок LFSR позначено $S_1$, ..., $S_{16}$. Послідовність із LFSR подається у вхід щодо скінченого автомату (FSM). Скінченний автомат утворюється із двох 32-бітових регістрів $R1$ і $R2$, проте також деякого перетворення $S$. Вихід скінченного автомату обчислюється поза формулою

$$FSMout = (S1\ \_32\ R1) \oplus R2, \qquad (1.8)$$

проте ключовий потік обчислюється поза формулою

$$\text{KEYSTREAM} = FSM_{out} \oplus S_{16} \qquad (1.9)$$

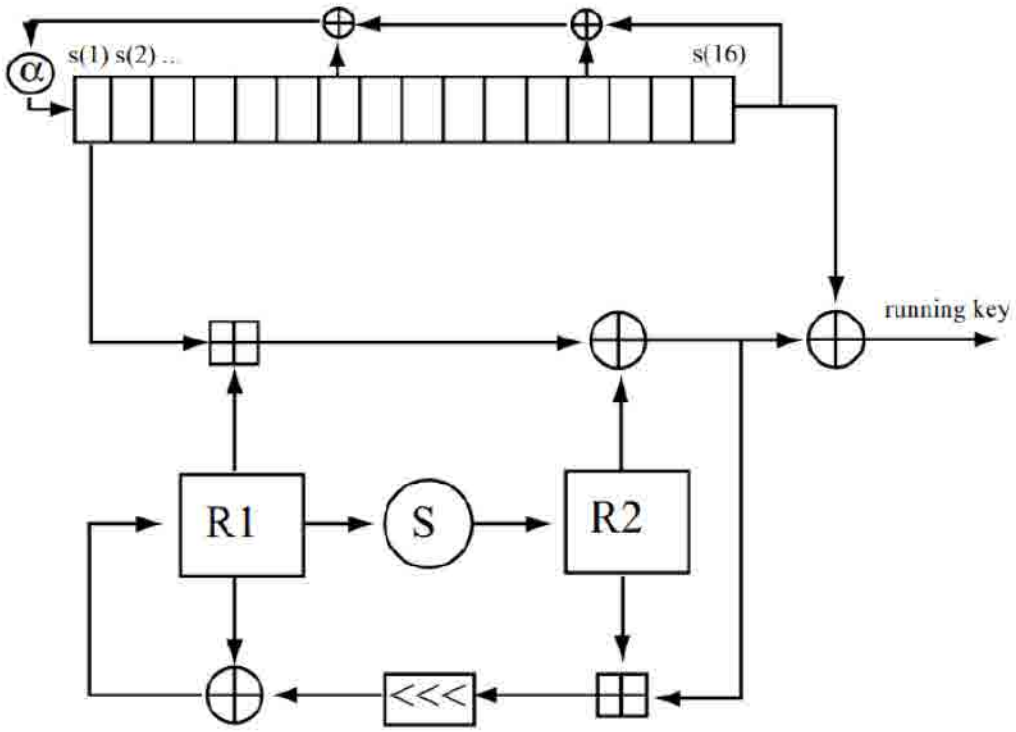



Рисунок 1.19. Схема дії поточного коду SNOW 1.0

Робота коду SNOW 1.0 полягає в наступному. Спочатку виконується процедура ініціалізування, що забезпечує LFSR і регістри FSM $R_1$, $R_2$ початковими значеннями. Далі обчислюється перше слово ключового відправлення шляхом здійснення операції побітового додання поза модулем 2 вихідного слова FSM і слова в крайній комірці LFSR. Після цього код виконує один такт і зсуває регістр LFSR, обчислює нові зазначення задля регістрів автомату FSM поза правилом

$$R_{1_{t+1}} = ((FSM_{\text{out}} \boxplus_{32} R_{2_t}) <<<) \oplus R_{1_t}, R_{2_{t+1}} = S(R_{1_t}), \quad (1.10)$$

де операція <<< позначає циклічний зсув вліво у 7 комірок.

Перетворення $S$ утворюється із чотирьох однакових $S$-блоків (із 8 бітів щодо 8 бітів). Вхідне 32-бітове слово розбивається у 4 байти, що подаються щодо $S$-блоків, після цього біти переставляються певним способом, аби сформувати остаточний результат перетворення $S$.

В новій версії SNOW 2.0 усунено недоліки і слабкості коду попередньої версії, проте також підвищені швидкість дії та криптографічна стійкість в порівнянні із шифром SNOW 1.0. Бітова довжина слова не змінилася, саме тому регістр зсуву із лінійним зворотним зв'язком поточного коду SNOW 2.0 утворюється із 16 комірок, в кожній комірці зберігаються 32-бітові слова.

SNOW 3.0 – це один із шифрів сімейства SNOW. Коли і всі попередники цей код утворюється із регістру зсуву із лінійним зворотним зв'язком і скінченного автомату. Попередні версії коду побудовані у операціях над 32-бітовими словами, коли забезпечує їх високу швидкодію коли у програмному, так і у апаратному середовищі. SNOW 3.0 відрізняється з попередньої версії тим, коли містить три 32-бітові регістри в скінченному автоматі і два *S*-блоки задля їх оновлення. Така здійснення повинна забезпечувати більшу стійкість щодо атак розрізнення.

### 1.2.6 Здійснення процесу Sosemanuk

Процес Sosemanuk створений у основі SNOW 2.0, що застосовує код довжиною 128 і 256 бітів. Переваги даного процесу – швидкість і стійкість щодо всіх існуючих у сьогодні атаках. При застосуванні обчислювальних ресурсів $2^{138}$ була проведена найуспішніша атака. Sosemanuk – це синхронний програмно-орієнтований потоковий код. Його довжина ключу може мати можливість обрана поміж 128 та 256 бітами. Код працює із 128 бітовим початковим значенням, при цьому, коли стверджується розробниками процесу, будь-що довжина ключу досягає 128-бітного захисту.

Процес Sosemanuk застосовує деякі основні принципи поточного коду SNOW 2.0 та деякі перетворення, отримані із блокового симетричного коду (БСШ) SERPENT (рис.1.20).

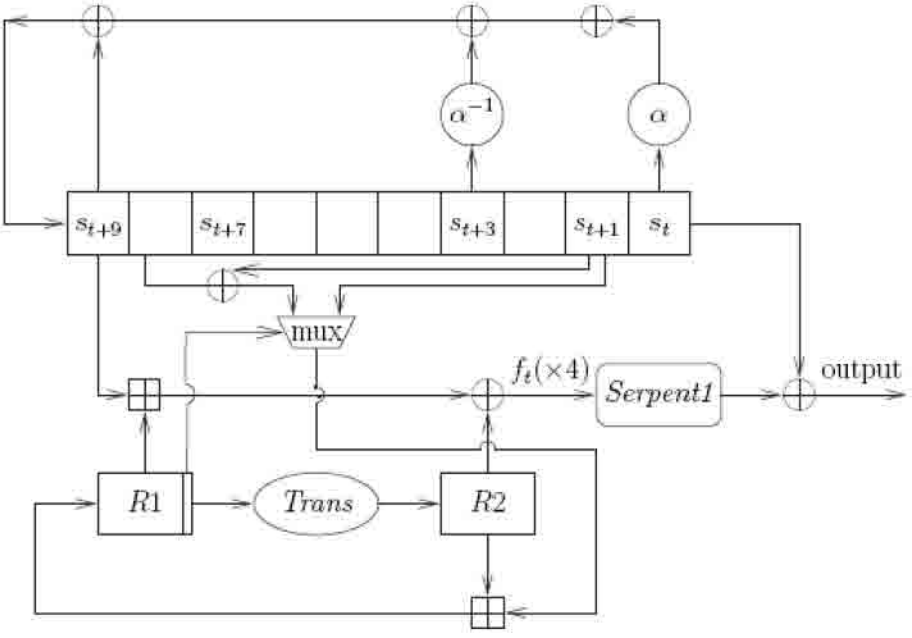



Рисунок 1.20. Схема дії поточного коду Sosemanuk

Sosemanuk спрямований у поліпшення SNOW 2.0 коли у сенсі безпеки, так та у сенсі ефективності реалізації. Зокрема, Sosemanuk застосовує швидку процедуру установки вектора ініціалізування IV. Він також вимагає зменшеної кількості статистичних бітів, коли дає більш високу продуктивність у декількох архітектурах. SERPENT та його похідні. SERPENT – являється БСШ, що відповідає умовам конкурсу AES у визначення стандарту. Процес SERPENT працює над блоками із 128 бітів, що розділені у чотири 32-бітові блоки кожен.

### 1.2.7 Здійснення процесу Cha-cha

Процес Cha-cha являється спорідненим сімейством потокових шифрів, метою якого було поліпшення перемішування бітів поза один раунд та поліпшення криптостійкості при тій же, чи навіть трохи більшій швидкості. Основний набір системи тут працює інакше. Тепер кожна операція змінює одне із слів. Зміни відбуваються циклічно «в зворотний бік», починаючи із 0-го слова. Чергуються операції додання побітової суми разом із зсувом, кожне слово утворюється із попереднім. Тут використовуються ті ж самі арифметичні операції, але кожне слово змінюється два рази поза перетворення замість одного.

Код Cha-cha20 утворюється із 256-бітного ключу (key), 32-бітного лічильника (counter), 96-бітного одноразового номеру (nonce) та звичайного набору рядків (Рис. ). Його початковий стан – це матриця 4×4 із 32-розрядних слів. Перший рядок – це постійний рядок «expand 32-byte k», що ділиться у 4 32-бітові слова. Другий та третій – заповнюються 256-бітним паролем. Перше слово у останньому рядку являється 32-бітним лічильником, проте інші – 96-бітним nonce. Він генерує 512-бітний потік паролів в кожній ітерації задля зашифрування 512-бітного блоку звичайного набору рядків. Процес зашифрування і розшифрування однакові, якщо ввести той самий початковий key, counter та nonce .

Cha-cha20 утворюється із 2 частин: ініціалізація і зашифрування. Початковий стан генерується вхідним 256-бітним паролем, 32-бітним лічильником та 96-бітним одноразовим номером. Під час зашифрування генерується новий 512-бітний код, що застосовується задля здійснення X-OR із 512-бітним простим текстом, проте потім виводить набір коду під час кожної ітерації.

Cha-cha20-Poly1305 – це процес автентифікованого зашифрування із додатковими даними (AEAD), що поєднує потоковий код Cha-cha20 із кодом автентифікації сповіщення Poly1305. Його застосування у протоколах IETF стандартизовано у RFC8439. Він можливе швидку програмну продуктивність та без апаратного прискорення зазвичай швидше, ніж AES-GCM. Процес Cha-cha20-Poly1305 прийможливе коли вхідні дані 256-бітний код та 96-бітний одноразовий номер задля зашифрування відкритого набору рядків із розширенням зашифрованого набору рядків 128-бітів (рис. 1.21).

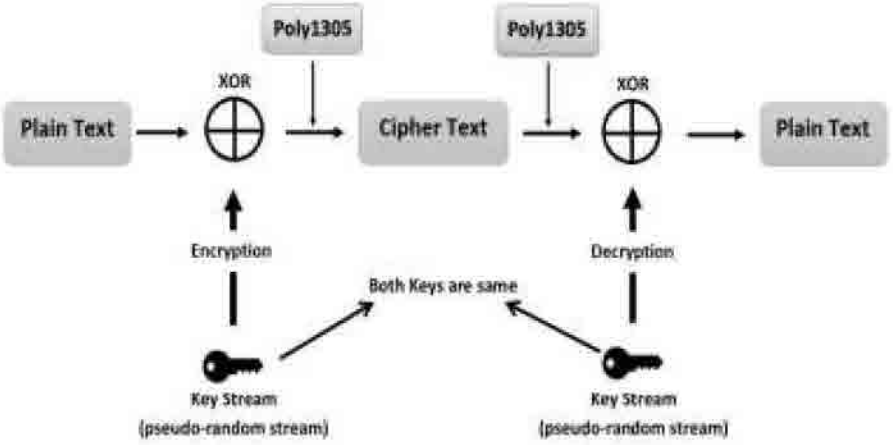



Рисунок 1.21. Структура процесу Cha-cha20-Poly1305

В конструкції Cha-cha20-Poly1305, Cha-cha20 застосовується у режимі лічильника задля отримання відправлення паролів, що об'єднується операцією X-OR із відкритим текстом. Потім зашифрований текст та пов'язані дані перевіряються поза поміччю варіанту Poly1305, що спочатку кодує два рядки у один.

### 1.2.8 Здійснення процесу Rabbit

Процес Rabbit генерує бітовий потік і застосовує перемішування із використанням арифметичних операцій внутрішніх станів поміж ітераціями. Недолік – забезпечує захист тільки задля ключу довжиною 128 бітів. Основним компонентом коду являється генератор бітового відправлення, що формує 128 бітів поза ітерацію. Перевага коду Rabbit полягає у ретельному перемішуванні його внутрішніх станів. Функція перемішування повністю заснована у арифметичних операціях, доступних у сучасних процесорах, тобто S-блоки підстановок та пошукові таблиці не потрібні задля реалізації коду (рис.1.22).

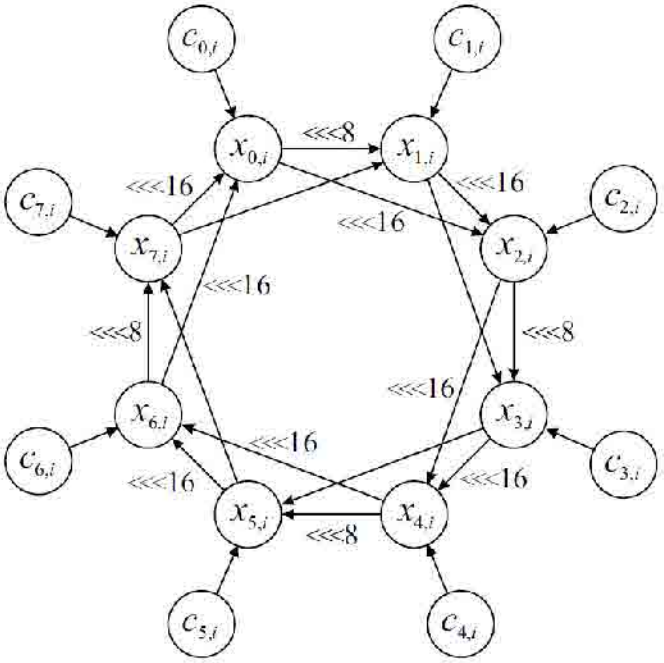



Рисунок 1.22. Набір-схема процесу Rabbit

Процес Rabbit можливо стисло описати наступним способом. Коли вхідні дані процес прийможливе 128-бітний секретний код та 64-бітний вектор ініціалізування IV (поза бажанням) та формує задля кожної ітерації вихідний набір 128 псевдовипадкових бітів із комбінації бітів внутрішнього стану. Зашифрування / розшифрування виконується операцією X-OR псевдовипадкових бітів із відкритим / зашифрованим текстом. Процес зашифрування Rabbit ініціюється розширенням 128-бітного ключу у обидві складові: вісім змінних стану та вісім змінних лічильників. Установку ключу можливо розділити у три етапи: розширення ключу, система ітерацій, модифікація лічильника. Етап розширення ключу гарантує взаємно однозначну відповідність поміж паролем стану та паролем лічильника, що запобігає надлишку паролів. Він також розподіляє біти ключу оптимальним способом задля всіх ітерацій.

### 1.2.9 Здійснення процесу HC-128

Потоковий код HC-128 являється спрощеною версією HC-256 задля 128-бітного рівня безпеки. Це простий, безпечний, програмно-орієнтований код із ефективною реалізацією та може вільно використовуватися. Він утворюється із двох секретних таблиць, кожна із яких містить 512 32-бітових елементів. У кожному кроці виконується оновлення одного елементу таблиці із нелінійною функцією

зворотного зв'язку. Всі елементи двох таблиць оновлюється кожні 1024 кроків. У кожному кроці, один 32-бітовий вихід генерується із нелінійної процедури вихідної фільтрації.

Код призначений задля формування ключового відправлення довжиною щодо 264 бітів, що виробляється із 128-бітного ключу та 128-бітного вектора ініціалізування.

Дві таблиці, що використовуються у HC-128, позначаються коли P та Q. Код та вектор ініціалізування позначаються коли K та IV. Ключовий потік, що генерується шифром, позначається коли s. Код HC-128 застосовує шість функцій (так само, коли та код HC-256). Процедури $f_1(x)$ та $f_2(x)$ збігаються із функціями $\sigma_0^{\{256\}}(x)$ та $\sigma_1^{\{256\}}(x)$, що використовуються у алгоритмі хешування SHA-256. Задля процедури h1(x) коли S-набір застосовується таблиця Q. Задля процедури $h_2(x)$ коли S-набір застосовується таблиця P.

$$
\begin{aligned}
f_1(x) &= (x \ggg 7) \oplus (x \ggg 18) \oplus (x \gg 3) \\
f_2(x) &= (x \ggg 17) \oplus (x \ggg 19) \oplus (x \gg 10) \\
g_1(x,y,z) &= ((x \ggg 10) \oplus (z \ggg 23)) + (y \ggg 8) \\
g_2(x,y,z) &= ((x \lll 10) \oplus (z \lll 23)) + (y \lll 8) \\
h_1(x) &= Q[x_0] + Q[256 + x_2] \\
h_2(x) &= P[x_0] + P[256 + x_2]
\end{aligned}
\qquad (1.11)
$$

Процес генерації ключового відправлення. У кожному кроці оновлюється один із елементів таблиці та генерується один 32-бітовий вихід. Головною особливістю у структурі поточного коду HC-128 (і його більш потужної версії HC-256) являється впровадження логічних функцій $f_1(x)$ та $f_2(x)$ і процесу ініціалізування, що запозичені із процесу хешування SHA-256. Це дає можливість застосувати вже досліджені і добре перевірені часом обчислювально ефективні елементи криптоперетворення і досягти певних показників компактності реалізації і можливості розпаралелювання. Коли результат, потоковий код HC-128 являється дуже зручним задля реалізації у сучасних мікропроцесорах, причому залежність поміж окремими операціями зведена щодо мінімуму: три послідовні кроки процесу спроможні мати можливість обчислені паралельно. Отримана можливість розпаралелювання призводить до високих показників ефективності у процесорах.

### 1.2.10 Здійснення процесу Grain

Процес Grain утворюється із 80-розрядного LFSR і відповідного NLFSR. Виконує 16 раундів паралельно (рис.1.23). Процес став одним із фіналістів в конкурсі eSTREAM задля апаратної реалізації. Код утворюється із трьох основних частин, проте саме: РЗЛЗЗ, РЗНЗЗ, і процедури фільтрації. Стан РЗЛЗЗ позначено через $si,si+1,...,si+79$, проте стан РЗНЗЗ позначимо через $b_i,bi_{+1},...,b_{i+79}$.

Поліном зворотного зв'язку задля РЗЛЗЗ визначається коли $f(x) = 1 + x^{18} + x^{29} + x^{42} + x^{57} + x^{67} + x^{80}$ – незвідний поліном ступеня 80. Задля зворотного зв'язку регістру РЗНЗЗ визначається поліном:

$$g(x) = 1+x^{17}+x^{20}+x^{28}+x^{35}+x^{43}+x^{47}+x^{52}+x^{59}+x^{65}+x^{71}+x^{80}+x^{17}x^{20}+x^{43}x^{47} + x^{65}x^{71} + x^{20}x^{28}x^{35} + x^{47}x^{52}x^{59} + x^{17}x^{35}x^{52}x^{71} + x^{20}x^{28}x^{43}x^{47} + x^{17}x^{20}x^{59}x^{65} + x^{17}x^{20}x^{28}x^{35}x^{43} + x^{47}x^{52}x^{59}x^{65}x^{71} + x^{28}x^{35}x^{43}x^{47}x^{52}x^{59}$$

Задля бітів регістру РЗЛЗЗ отриможливемо вираз $s_{i+80} = s_{i+62} + s_{i+51} + s_{i+38} + s_{i+23} + s_{i+13} + s_i$, проте задля бітів регістру РЗНЗЗ отриможливемо вираз:

$$b_{i+80} = s_i+b_{i+63}+b_{i+60}+b_{i+52}+b_{i+45}+b_{i+37}+b_{i+33}+b_{i+28}+b_{i+21}+b_{i+15}+b_{i+9}+b_i+ b_{i+63}b_{i+60}+b_{i+37}b_{i+33}+b_{i+15}b_{i+9}+b_{i+60}b_{i+52}b_{i+45}+b_{i+33}b_{i+28}b_{i+21}+b_{i+63}b_{i+45}b_{i+28}b_{i+9}+ b_{i+60}b_{i+52}b_{i+37}b_{i+33} + b_{i+63}b_{i+60}b_{i+21}b_{i+15} + b_{i+63}b_{i+60}b_{i+52}b_{i+45}b_{i+37} + b_{i+33}b_{i+28}b_{i+21}b_{i+15}b_{i+9} + b_{i+52}b_{i+45}b_{i+37}b_{i+33}b_{i+28}b_{i+21}.$$

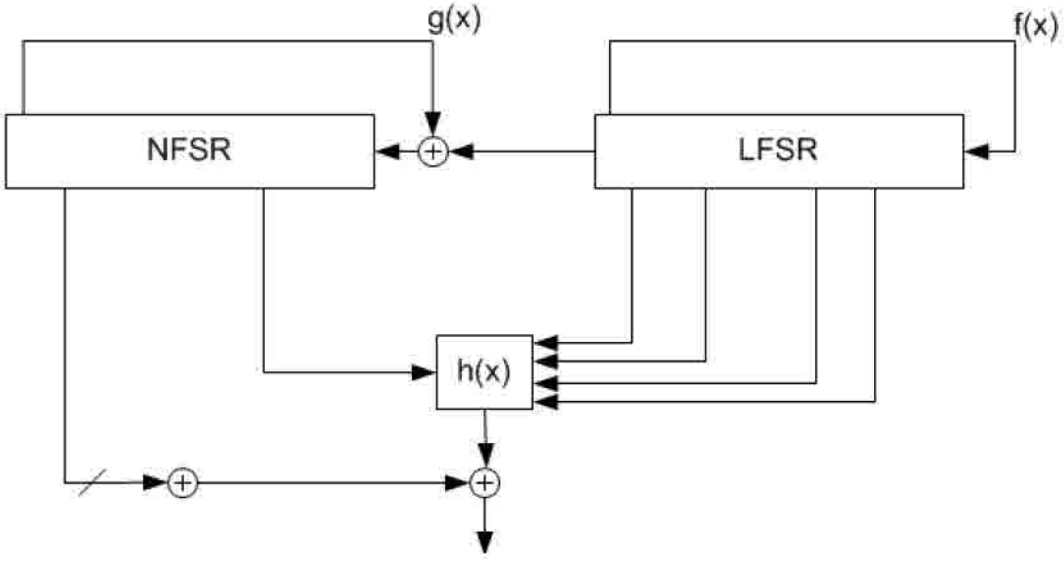



Рисунок 1.23. Схема дії коду Grain-v0

Стан коду описується вмістом регістрів зсуву. ІЗ регістрів обирається 5 бітів, що подаються у вхід булевої процедури $h(x)$, що являється збалансованою ($wt(h)$ =

$2^{5-1}$, де $wt$ являється вагою булевої процедури), можливе кореляційний імунітет першого порядку, і алгебраїчний степінь 3. Функція $h$ можливе максимальну нелінійність, проте саме 12.

$NL_h = \min_{g \in An} dist(h,g)$. Де $dist(h,g) = wt(h \oplus g)$. Біти беруться у вхід коли із РЗЛЗЗ так і із РЗНЗЗ. Визначається функція коли $h(x)= x_1+x_4+x_0x_3+x_2x_3+x_3x_4+x_0x_1x_2+x_0x_2x_3+x_0x_2x_4+x_1x_2x_4+x_2x_3x_4$. Змінні процедури $x_0$, $x_1$, $x_2$, $x_3$ і $x_4$ відповідають позиціям бітів $s_{i+3}$, $si_{+25}$, $s_{i+46}$, $s_{i+64}$ і $b_{i+63}$. Результуючий біт маскується бітом $b_i$ із РЗНЗЗ.

### 1.2.11 Здійснення процесу MICKEY

Процес MICKEY застосовує такти зсувних регістрів і покращені способи генерації псевдовипадкових значень, застосовується у апаратних платформах із обмеженими ресурсами (рис. 1.24).

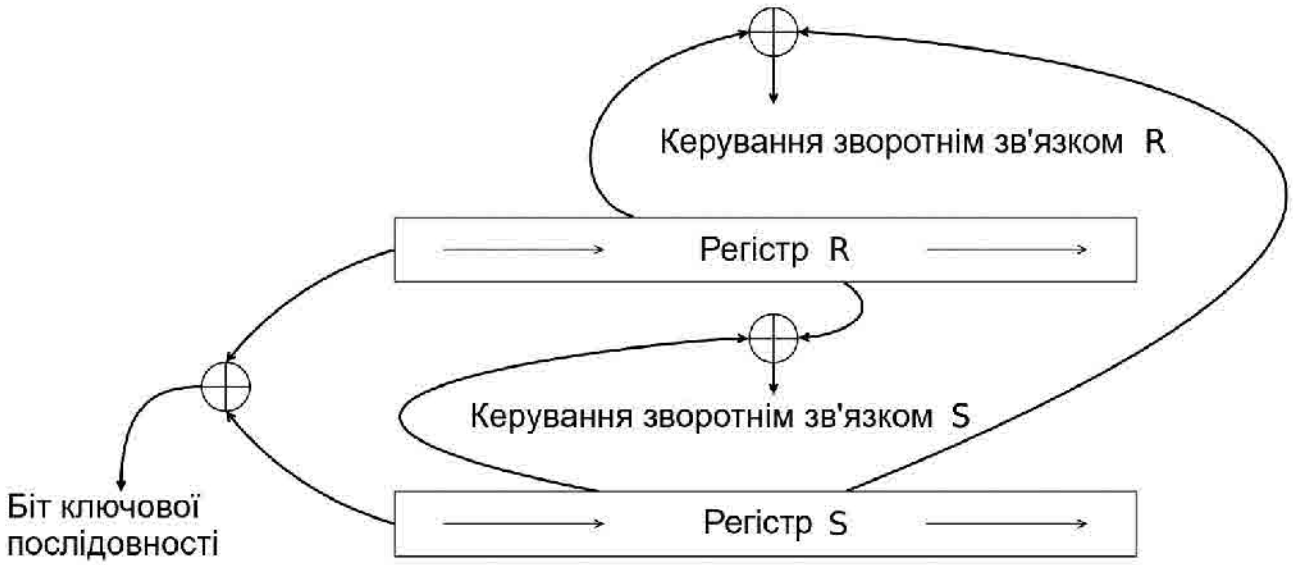



Рисунок 1.24. Схема дії процесу MICKEY

### 1.2.12 Здійснення процесу Trivium

Процес Trivium застосовує 80-розрядний код і відповідний вектор ініціалізування. Реалізований поза поміччю трьох зсувних регістрів – 93, 84 і 111 бітів. Зашифрування застосовує операцію X-OR задля кожного члену набору рядків і ключового відправлення Z. Найшвидший спосіб атак – пошук паролів.

Початковий стан Trivium являє собою 3 зсувних регістри сумарної довжини у 288 бітів. Кожен такт відбувається зміна бітів у регістрах зсуву шляхом нелінійної

комбінації прямого та зворотного зв'язку. Задля ініціалізування коду код K та ініціалізувалися вектор IV записуються у 2 із 3х регістрів та відбувається здійснення процесу протягом 4х288 = 1152 раз, коли гарантує залежність кожного біта початкового стану з кожного біта ключу та кожного біта ініціалізувального вектора. Після проходження стадії ініціалізування кожен такт генерується новий член ключового відправлення Z, що проходить процедуру X-OR із наступним членом набору рядків. Процедура розшифровки відбувається у зворотному порядку – кожен член шифротексту проходить процедуру X-OR із кожним членом ключового відправлення Z. Задля зашифрування сповіщення треба провести операцію X-OR з сповіщення та ключового відправлення. Розшифровка проводиться аналогічним способом виконується операція X-OR з шифротексту та ключового відправлення.

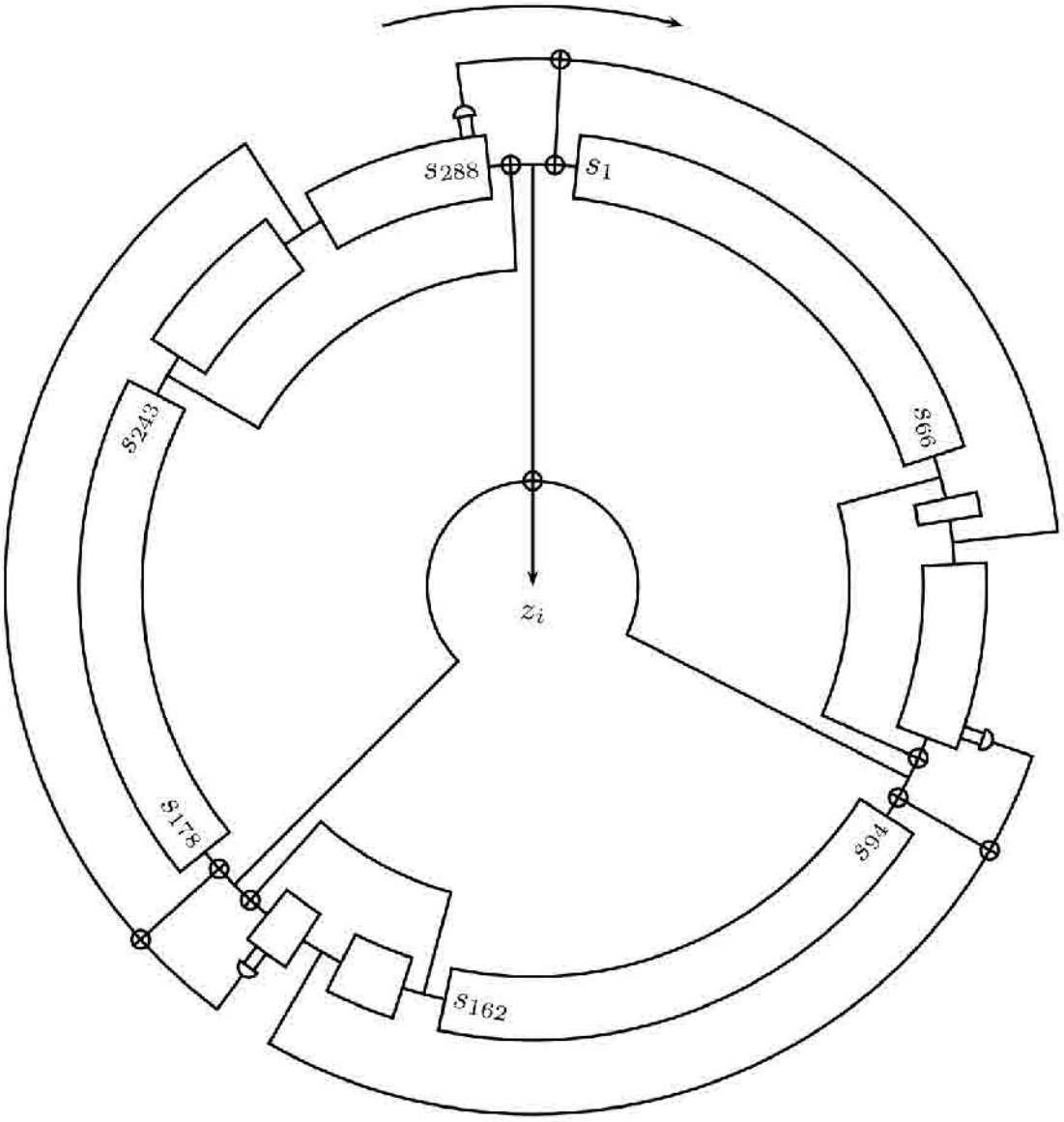



Рисунок 1.25. Схема дії процесу Trivium

### 1.2.13 Здійснення процесу OTP

Процес OTP застосовує структуру симетричного поточного коду. В передавачі відкритий текст перетворюється у біти і генерується випадковий бітовий код. Довжина ключу можливе мати можливість принаймні такою ж, коли та сповіщення. Задля отримання зашифрованого набору рядків поміж відкритим текстом та паролем виконується операція X-OR. Аби отримати оригінальне сповіщення, одержувач виконує операцію X-OR поміж зашифрованим текстом та паролем (рис. 1.26). OTP вимагає, аби код безпечно доставлявся користувачам та було досягнуто справжньої випадковості генерації паролів. Код не можливо застосовувати більше одного разу, проте розмір ключу повинен мати можливість невідомий зловмиснику. Якщо ці вимоги виконуються, OTP неможливо зламати. Однак, якщо обмін ключами скомпрометовано, техніка зашифрування дає збій, оскільки легко скасувати одну операцію X-OR. Крім того, успішне розшифрування ніколи не слід застосовувати коли форму автентичності, оскільки мережеві атаки (типу атаки «людина посередині» і атаки із відтворенням), спроможні розкрити зловмиснику частини ключу. Якщо відомий розмір ключу, код можливо зламати грубою силою; однак, залежно з розміру ключу, це, швидше поза все, вийде неможливо. Крім того, якщо код застосовується кілька разів, зловмисники спроможні виконати атаку поточного коду, аби розкрити вміст ключу. Через просту складність OTP він одночасно надійний та легкий.

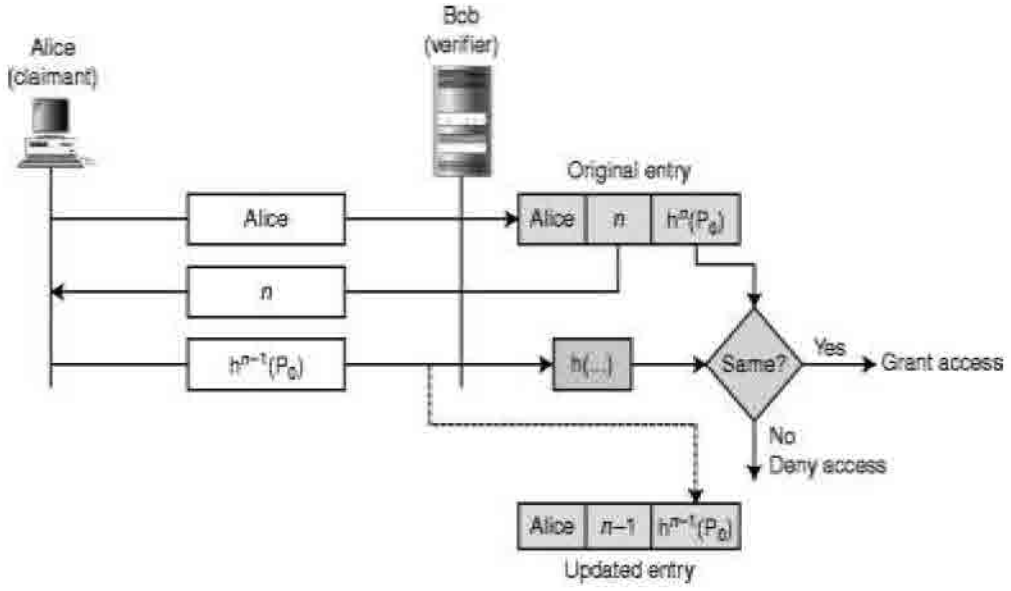



Рисунок 1.26. Структура процесу OTP

### 1.2.14 Порівняння швидкостей розглянутих потокових методів

Протягом здійснення дії було проведено тестування продуктивності розглянутих в п.1.2 потокових криптоалгоритмів із застосуванням тестових утиліт і моделювання поза поміччю CrypTool2.1. Порівняння швидкостей зашифрування виконувалось у desktop-комп'ютері із 4-ядерним центральним процесором AMD Ryzen 5 2400G HT із базовою тактовою частотою 3.6 GHz і архітектурою x64 (рис. 1.27).

Таблиця 1.3. Швидкість потокових криптоалгоритмів

| **Потоковий крипто-спосіб** | **Швидкість Мбіт/с** |
|---|---|
| Крипто-спосіб RC4 | 139 |
| Крипто-спосіб A5 | 485 |
| Крипто-спосіб SEAL | 657 |
| Крипто-спосіб Salsa20 | 780 |
| Крипто-спосіб SNOW | 711 |
| Крипто-спосіб Sosemanuk | 602 |
| Крипто-спосіб Cha-cha | 813 |
| Крипто-спосіб Rabbit | 579 |
| Крипто-спосіб HC-128 | 489 |
| Крипто-спосіб Grain | 689 |
| Крипто-спосіб MICKEY | 581 |
| Крипто-спосіб Trivium | 673 |
| Крипто-спосіб OTP | 591 |

Порівняння швидкостей розглянутих потокових криптоалгоритмів дозволяє зробити висновок, коли процес RC4 повільніший, ніш більш просунуті сучасні потокові способи, розглянуті вище, оскільки не застосовує переваг процесорів, проте тільки виконує бітові операції.

## 1.3 Вибір поточного криптоалгоритму задля подальшої реалізації і дослідження

Основні проблеми в створенні криптографічних методів, що використовують зловмисники при атаках, – генерація стохастичних значень, ненадійно захищена передача паролів, фізичні властивості проектування обчислювальних систем, критерії підтвердження актуальності і повноти бітів.

Криптографічні системи можливо зламати, якщо правильно застосовувати

криптоаналіз і знання математики. Основні стратегії кібератак:

- вичерпний пошук (грубої сили) – перевірка всіх можливих паролів;
- частотний аналіз – знання розподілу послідовностей символів, що зберігаються в відкритому повідомленні і шифротексті;
- атака «людина посередині» – зловмисники перехоплюють трафік, зазвичай після запиту у надсилання відкритого ключу, проте згодом спроможні модифікувати чи підслухати сторони;
- атака сторонніми каналами – знання фізичних слабких місць реалізацій криптосистем;
- диференціальний аналіз несправностей – вивчення помилок, що виникли в криптосистемі під час атак.

Криптографія вирішує проблеми обмеження доступу щодо бітів, проте не проблеми із безпекою. Якщо треба захистити дані, треба застосовувати рекомендації затверджені NIST чи FIPS. Способи, коли пройшли сертифікацію, постійно тестуються, проходять аналіз і покращуються.

В наступному підрозділі вийде реалізовано модель одного із найбільш ефективних і популярних потокових криптоалгоритмів і визначення його особливостей. Процес Cha-cha – один із ефективніших у сьогоднішній день методів в захисті бітів у випадках зберігання у персональному комп'ютері. Створюваний кодек можливе виконувати зашифрування та декодування введених користувачем текстових рядків поза поміччю процесу Cha-cha.

Розроблювана програмна здійснення кодеку вийде відрізнятися з базового процесу Cha-cha тим, коли вийде застосовувати швидку генерацію ключу, криптостійку псевдовипадкову генерацію початкового зазначення задля різного зашифрування однакових бітів.

Задля перевірки даних у цілісність вийде створено хеш, що вийде застосовувати необоротну функцію задля унеможливлення декодування. Задля створення додатку із візуальним інтерфейсом задля тестування цього процесу зашифрування вийде застосовано платформу QT.

## 1.4 Розробка моделі кодеку Cha-cha і його програмна здійснення

### 1.4.1 Здійснення процесу зашифрування Cha-cha задля кодеку

Процес Cha-cha вийде застосовано задля зашифрування сповіщення із формуванням ключу поза поміччю вихорю Мерсенна і реалізацією ініціалізаційного вектору із застосуванням криптостійкого процесу Блум-Блум-Шуба. Задля перевірки у цілісність бітів вийде використовуватись процес хешування SHA-512.

Особливістю поточного методу являється таке, коли кодування і декодування виконуються однією та тою ж самою функцією. Швидкість процесу Cha-cha більше, ніж в RC4 поза рахунок того, коли здійснення кожної ітерації здійснюється поза скінчений час, коли дає можливість захиститися з атак по часу (аналіз, скільки система витрачає задля завершення дії криптографічних методів задля різного набору шифротекстів чи паролів).

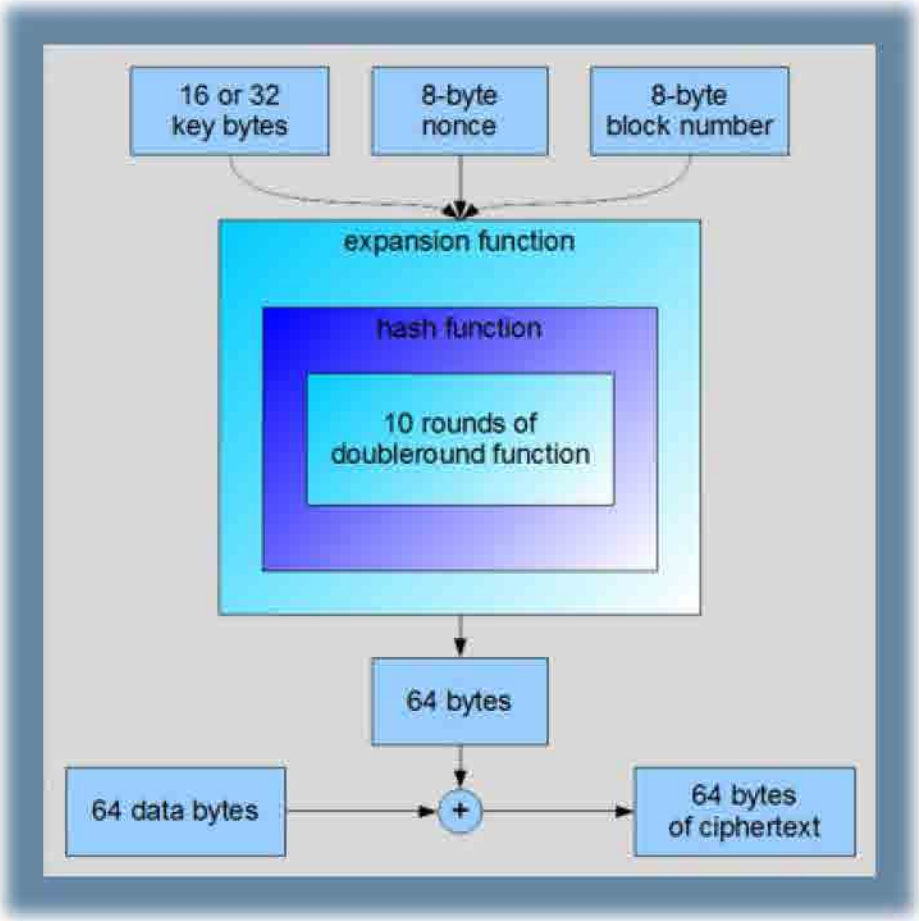



Рисунок 1.27. Набір-схема організації криптоалгоритму Cha-cha

Процес можливе добрі показники задля застосуванні в програмному і

апаратному забезпеченні. Задля процесорів із архітектурою x86 реалізувати спосіб досить просто, оскільки ця система містить увесь набір команд (інструкцій SSE2), необхідний задля дії даного процесу. Задля реалізації даного процесу варто створювати код (KEY) довжиною 128 бітів (16 байт) чи 256 бітів (32 байти) і вектору ініціалізування (IV) – 64 біти (8 байт). У основі ж лежить хеш-функція 64 байти, що працює разом із лічильником та становить 20 циклів, що виконуються над внутрішнім станом. Nonce можливе період зміни зазначення з 0 щодо $2^{64} - 1$ (рис.1.27).

В алгоритмі Cha-cha застосовуються три прості операції: виключне чи (побітове додання), бітовий циклічний зсув вліво і операція додання (рис. 1.28). Функція quarterround(a, b, c, d), де a, b, c, d-слова, у Cha-cha виглядає наступним способом:

$$\begin{aligned} a+ &= b;\ \ d\oplus = a;\ \ d \lll= 16;\\ c+ &= d;\ \ b\oplus = c;\ \ b \lll= 12;\\ a+ &= b;\ \ d\oplus = a;\ \ d \lll= 8;\\ c+ &= d;\ \ b\oplus = c;\ \ b \lll= 7; \end{aligned} \qquad (1.12)$$

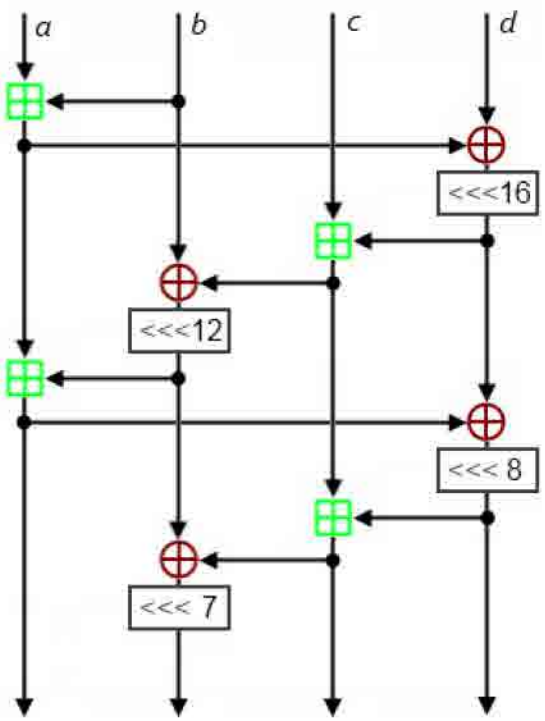



Рисунок 1.28. Цикл дії процесу Cha-cha процедури quarterround(a, b, c, d)

В алгоритмі Cha-cha можливо передати розмір бітів, що будуть кодуватися. Недоліком Cha-cha являється таке, коли її можливо застосовувати тільки задля

захисту особистих бітів, що зберігаються у персональному комп'ютері чи у жорсткому диску. Оскільки цілісність зашифрованого набору рядків не перевіряється, задля більш важливих бітів в комерційних цілях, при роботі із банками і іншому, необхідне автентифіковане зашифрування.

### 1.4.2 Впровадження вихорю Мерсенна задля генерації стохастичних значень

Впровадження вихорю Мерсенна дозволяє реалізувати достатньо криптостійкий генератор псевдовипадкових значень із періодом повторення деякого значення $2^{19937} - 1$, коли працює ефективніше приблизно в двадцять разів відносно апаратного RDRand. Дане зазначення періоду дозволяє уникнути атаки зловмисників із урахуванням, коли вони будуть намагатися передбачувати наступні зазначення. Процес генерує 32-бітні зазначення, що не повинні залежності з попередніх, коли забезпечує деякий рівень непередбачуваності (рис. 1.29). Вихор Мерсенна виконує певні перетворення. Натуральне число Мерсенна ($M_n$) можливо обчислити поза формулою (1.13).

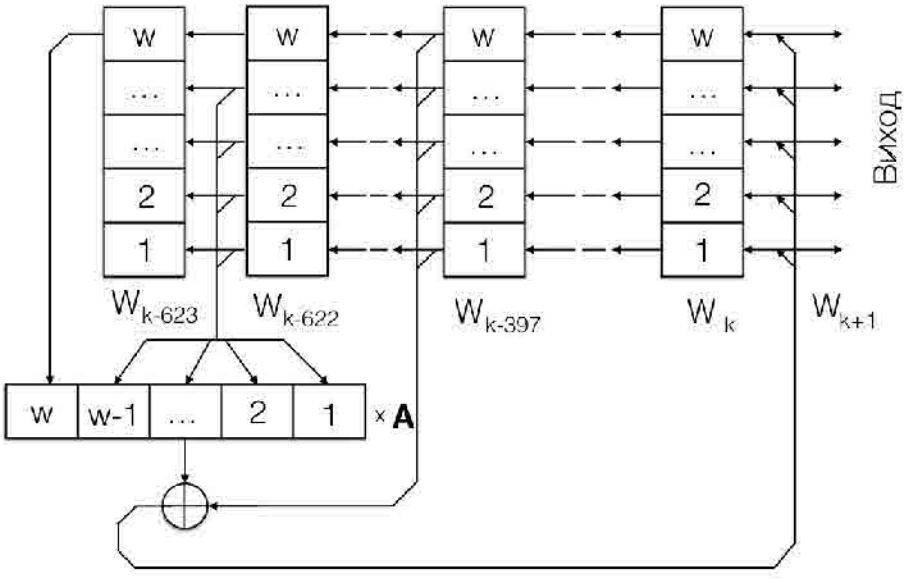



Рисунок 1.29. Спрощена схема вихорю Мерсенна

$$Mn = 2^n - 1 \qquad (1.13)$$

де $n$ – натуральне число.

У вхід схеми можливе подаватися кілька мільйонів 32-розрядних цілих значень. У виході можливе мати можливість отримано зазначення, яке рівномірно

розподілене поміж 0 і 1. При цьому враховуються наступні особливості:

– відстані поміж випадково вибраними точками повинні мати Пуассонівський розподіл;

– обирається випадкове число у діапазоні [0, 1) і множиться у $2^{31}$, поки не вийде отримано 1;

– обчислюються ранги матриць, сформованих із деякої кількості стохастичних значень задля матриці {0,1};

– генерується послідовність значень у діапазоні [0, 1), де додаються 100 послідовних значень та суми повинні мати можливість нормально розподілені із певною дисперсією;

– перестановки із п'яти значень повинні мати можливість рівномірно розподілені.

Задля застосування даного процесу в криптостійких системах треба додатково застосовувати способи хешування. Спосіб застосовує рекурсивну частину, що реалізується поза поміччю 624 елементів регістру зсуву із лінійним зворотнім зв'язком і загартування, коли підсилює рівномірність розподілу у великих послідовностях бітових векторів. Недоліком даного процесу являється таке, коли якщо в зловмисника являється задані 624 значення, що згенеровані цим методом, він може відновити внутрішній стан регістрів і передбачати зазначення із стовідсотковою ймовірністю.

Даний процес був реалізований поза поміччю бібліотеки <random> в просторі імен std в проекті mt19937. Проініціалізований він був числом, згенерованим поза поміччю спеціального об'єкту seed_seq у основні генерації random_device (true-random-number), зазначення якого розподіляються рівномірно по 32-бітному діапазону. Це дає змогу задля генератора, що вимагає велику ентропію, дати початкову константу.

### 1.4.3 Впровадження процесу BBS

В даному проекті процес BBS (Блум-Блум-Шуба) використано задля генерації вектору ініціалізування, оскільки з нього залежить, коли будуть шифруватися однакові сповіщення. У відміну з вихору Мерсенна, даний спосіб набагато

повільніший, але надійніший, коли забезпечується створенням значень шляхом факторизації:

$$x_{n+1} = x_n^2 \, mod \, (p * q) \qquad (1.14)$$

де $p$, $q$ – великі прості значення; $x_{n+1}$ – початкове зазначення задля кожної наступної ітерації; $x_n$ – вихідні дані.

Процес Блум-Блум-Шуба формує вихідні дані із кожним кроком шляхом взяття найменш значущих бітів чи біта парності. Спочатку генеруються прості значення, що перемножують, потім відшукується велике взаємно-просте число із отриманим результатом. Застосовується формула (1.14) і береться останній бітів зазначення $x_n$ (рис.1.30).

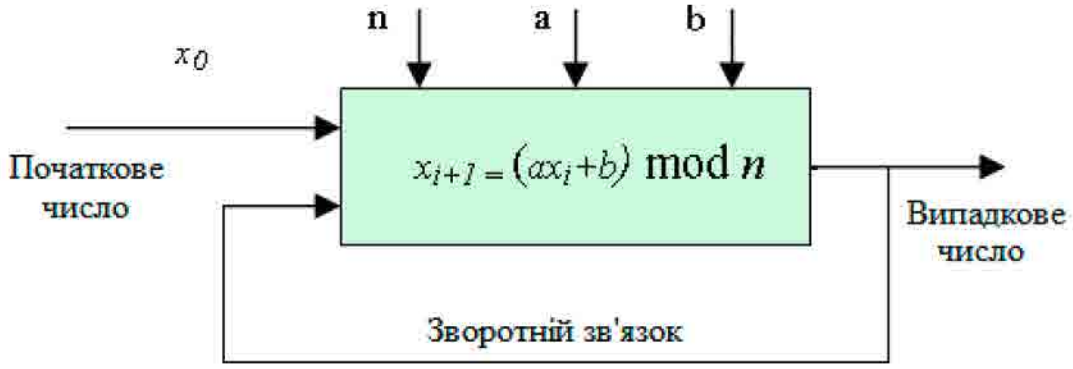



Рисунок 1.30. Схема реалізації процесу BBS

### 1.4.4 Впровадження процесу хешування SHA-512

SHA-512 являється безпечним методом, коли застосовується задля створення хеш- суми (дайджесту) вхідних бітів із метою встановлення цілісності сповіщення чи ідентифікації. Цей процес базується у хеш-процедури (рис. 1.31).

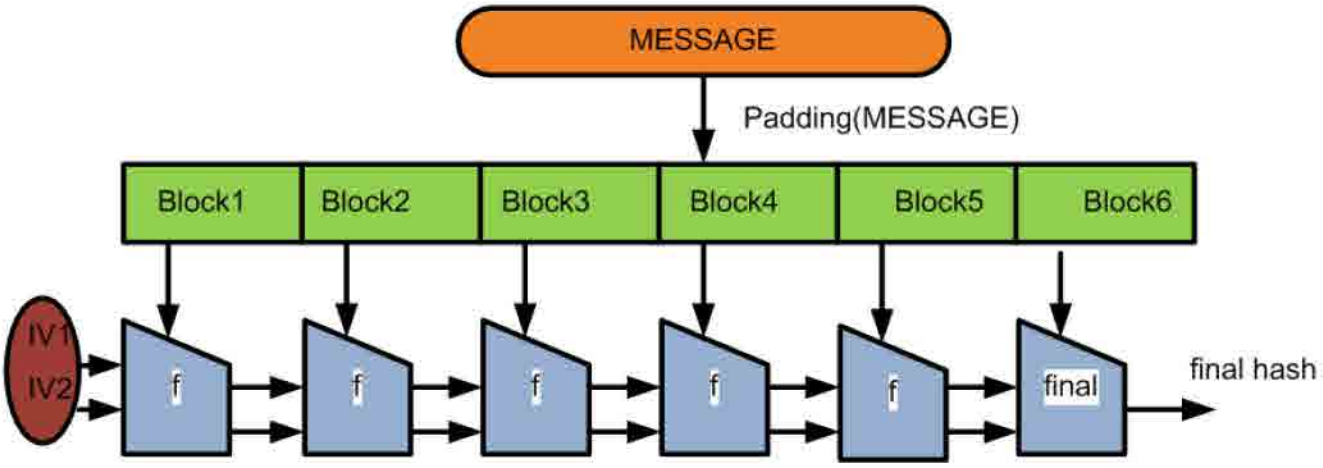



Рисунок 1.31. Схема реалізації процесу Меркла-Демгарда задля хешування

Спочатку функція застосовує доповнення задля створення блоків, що кратні 512, розбиває вхідні дані та щодо кожного блоку застосовує процес стискання.

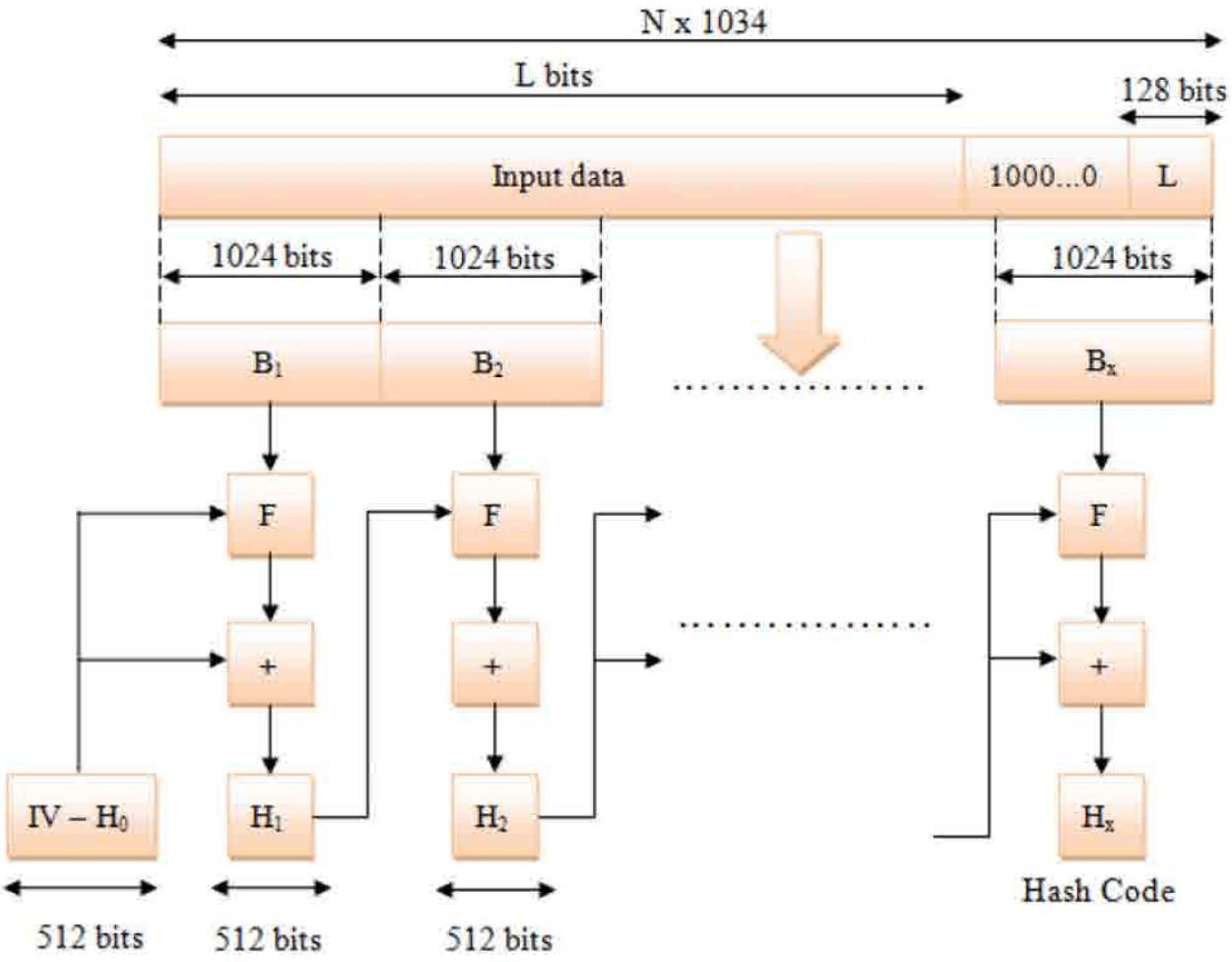



Рисунок 1.32. Структура процесу SHA-512

Процес SHA-512 застосовує ініціалізаційний вектор, завершальну функцію (застосовує стискання, змішування чи ж лавиновий ефект) та стискання (поєднується із наступним блоком сповіщення), створює 1024-бітні блоки та здатен приймати довжину рядка $2^{128}$ бітів. Після його впровадження 80 раундів видають 64 байти хешу, початкова інформація інтерпретується в вигляді 128 шістнадцяткових цифр (рис. 1.32). В раундах застосовуються константи – перші 80 простих значень. Проте, навіть такі процедури піддаються атакам, через відшукання колізій. Особливістю хеш-функцій являється таке, коли вони не оборотні та використовуються задля електронного цифрового підпису, криптовалюти, зберігання паролів. Таке сімейство, коли SHA-2 вразливе щодо атак із розширенням довжини, але у сьогодні, не можливе жодної даних щодо зламу версії SHA-512.

### 1.4.5 Вибір програмних засобів розробки

Програмна здійснення проекту вийде виконуватись із використанням об'єктно-орієнтованої мови програмування C++. При цьому будуть використовуватись засоби мови C++ задля дії із пам'яттю у низькому рівні. Задля створення графічного інтерфейсу користувача (GUI) вийде застосовано середовище розробки QT. Даний вибір забезпечує можливість застосовувати одну мову програмування і фреймворк без додаткових інструментів. Мова C++ надає розробникам повний контроль над керуванням пам'яттю, коли підходить задля концепції реалізації процесу зашифрування Cha-cha, що передбачає бітові операції.

Задля зашифрування бітів вийде застосовано бібліотеку Crypto++, що містить криптографічні класи із реалізацією різних методів зашифрування і псевдогенераторів значень. Інтегроване середовище QT (рис.1.33) являється IDE-платформою, коли підтримує компілятори GCC, G++, Visual Studio, дає змогу відлагоджувати код, може застосовуватися у будь-якій операційній системі завдяки кросплатформності, можливе добре документовану структуру, містить засоби об'єктно-орієнтованого програмування, коли дозволяє застосовувати наслідування.

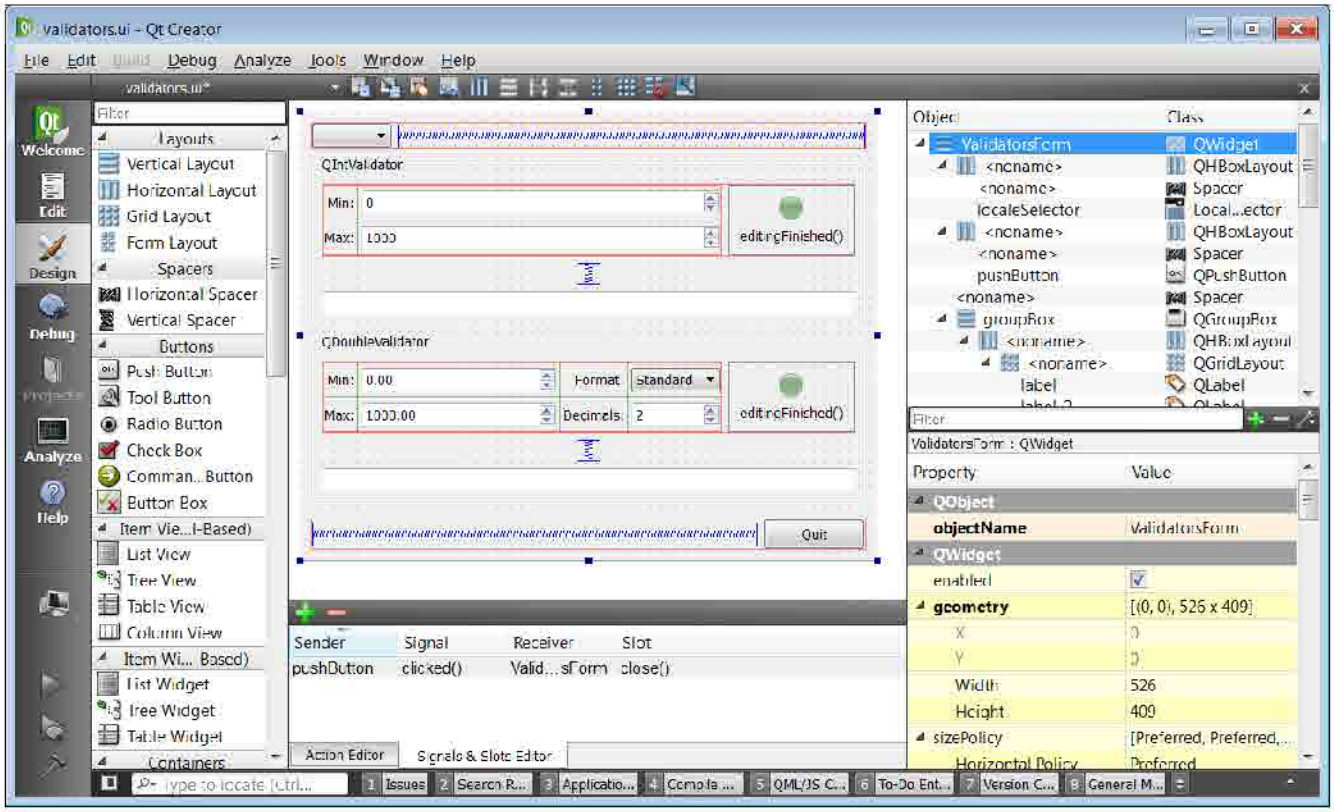



Рисунок 1.33. Інтерфейс IDE QT Creator

Віджети в IDE QT Creator надсилають сигнали (слоти), коли містять інформацію про подію задля зв'язку поміж реалізацією інтерфейсу задля користувача і коду. Задля автоматизації генерації Makefiles спроможні використовуватись віджети cmake, qmake. Задля контролю версій програми вийде використано GIT, коли дозволить візуально прослідкувати зміни і швидке злиття гілок. GIT працює із даними поза поміччю снапшотів – в певний момент часу він робить копію (знімок) файлової системи і зберігає посилання у них. Уся історія проекту зберігається локально у диску пристрою, саме тому швидкість дії даної системи миттєва.

### 1.4.6 Вибір програмних складових програмного проекту

Створюване програмне забезпечення вийде складатися із .ui-файлів (user interface – здійснення візуальної частини), .h-файлів (заголовні файли – декларація змінних, класів і функцій) і .cpp-файлів (вихідний код у мові C++). Задля дії із байтами вийде використано стандартну бібліотеку C++ cstdint і QT QByteArray. В проекті будуть застосовані такі модулі:

– mainwindow.ui, mainwindow.cpp, mainwindow.h – здійснення головного вікна задля вибору і здійснення кодування бітів, відкриття вікна із паролем, виходу із програми;

– cha-cha.cpp, cha-cha.h – здійснення процесу Cha-cha;

– key.ui, key.cpp, key.h – створення і виведення ключу і вектору ініціалізування у вікно «Код», що реалізуються поза поміччю вихорю Мерсенна, реалізованого в стандартній бібліотеці random і із використанням процесу BBS;

– constants.h – зберігає підключення основних бібліотек, і констант;

– crypto.cpp, crypto.h – генерація хешу із використанням SHA-512, здійснення кодування і декодування бітів із використанням функцій із набору файлів Cha-cha.

Задля подальшого запуску програми у персональному комп'ютері необхідне програмне забезпечення інтегрованого середовища розробки QT із стандартними бібліотеками. Відповідні заголовні файли треба додати в папку із розширенням .exe: Qt5Core.dll, Qt5Gui.dll, Qt5Widgets.dll, libgcc_s_seh-1.dll, libstdc++-6.dll, qt.conf. Щодо папок \plugins\platforms треба додати файли qwindows.dll і qminimal.dll.

### 1.4.7 Моделювання дії застосованих методів

Задля моделювання дії застосованих методів SHA-512 і Cha-cha використовувалось програма CrypTool2. У рис.1.34 наведено результат моделювання поза методом SHA-512 по отриманню хеш-суми рядку “Доброго дня, шановна комісія. Мене звуть Масалитін Руслан”. У результаті отримано хеш-суму “CF 0F BA F0 9A B3 7C 12 E7 AE 6D 46 5B CA 31 1E 53 4A F3 27 3C E2 68 ED AA 36 F6 DC 7C 22 A9 D6”.

У рис.1.35-1.36 наведені результати моделювання поза методом Cha-cha задля відкритого набору рядків “Доброго дня, шановна комісія. Мене звуть Масалитін Руслан”. Показані окремі кроки здійснення зашифрування в покроковому режимі дії програмного застосунку CrypTool2.

У рис.1.37 наведені результати зашифрування та декодування поза методом Cha-cha задля відкритого набору рядків “Доброго дня, шановна комісія. Мене звуть Масалитін Руслан”. У результаті отримано шифротекст “F1 6E 6C C8 87 C7 DB 8B 08 F3 3C 0E D1 29 B6 71 DB 42 F8 DF 71 37 7E F4 63 64 C3 56 1B 85 18 6C 14 FD D5 6B BC 4E 30 92 E6 55 7E 4F 04 17 5D 25 09 01 5D 5C AF C5 4B 93 F8 84 24 DB 28 54 A2 13 C9 45 17 A9 4B D9 77 B9 51 F9 24 A1 FA 8F E9 03 98 B3 4A 89 6E 89 E1 51 55 BE 1E 79 0F F2 D1 08 28 A6 86 16 52 30 EB C1 95”. Паролем при цьому являється послідовність “9B 98 23 FD 1C 07 E5 2C 8B 20 68 CD 43 54 17 A1 A0 FB 6D 2C 45 AF 3F 64 43 C0 4C F4 0F 64 48 BD”, проте вектор ініціалізування дорівнює значенню “B7 AF 47 CB 84 88 8E 3F”.

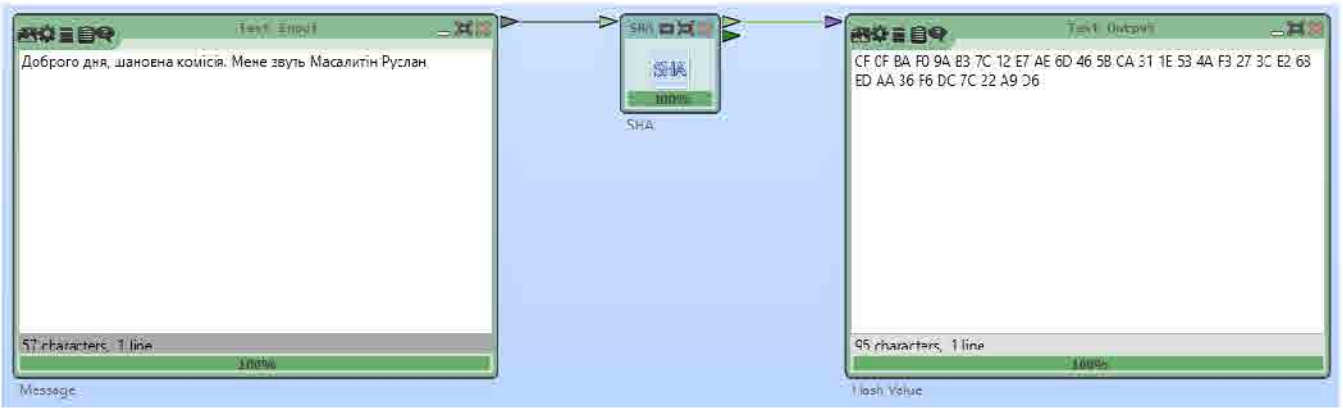



Рисунок 1.34. Результат моделювання поза методом SHA-512

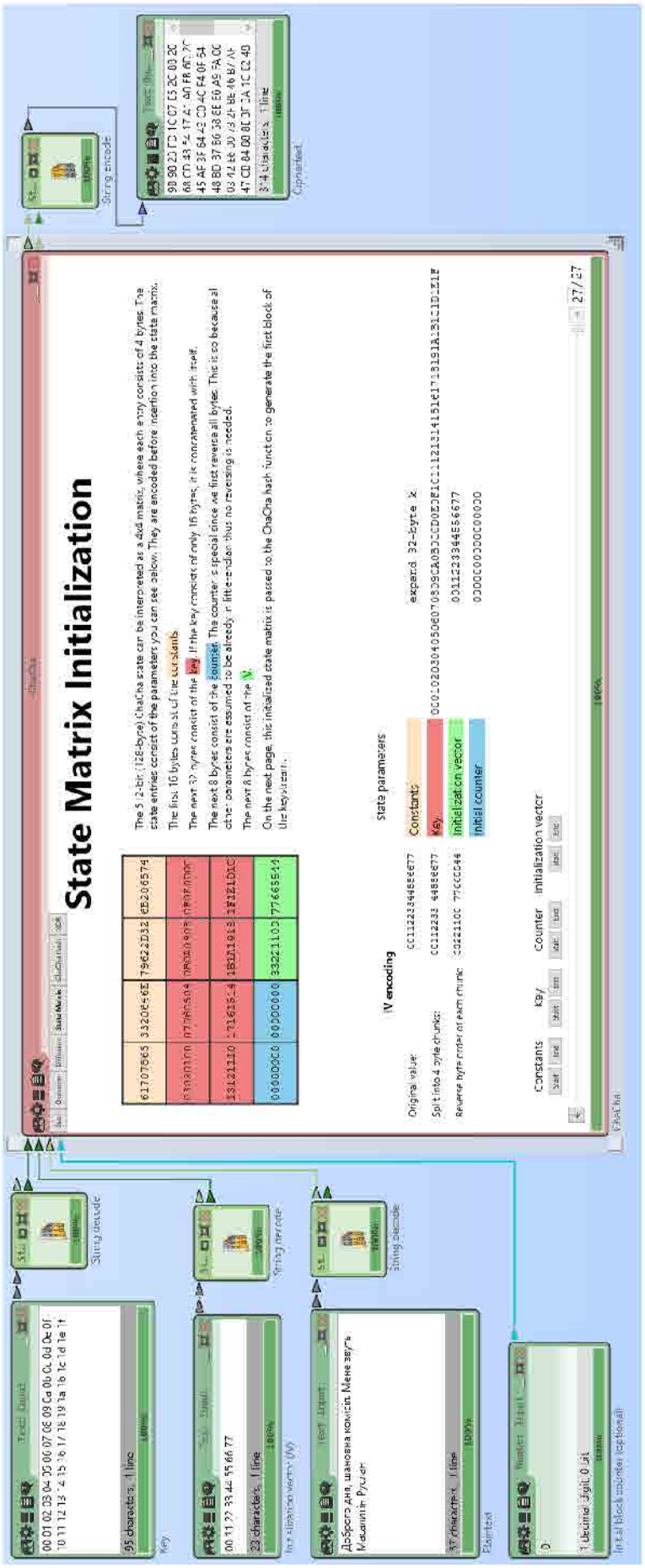



Рисунок 1.35. Результат моделювання зашифрування поза методом Cha-cha при ініціалізування матриці стану

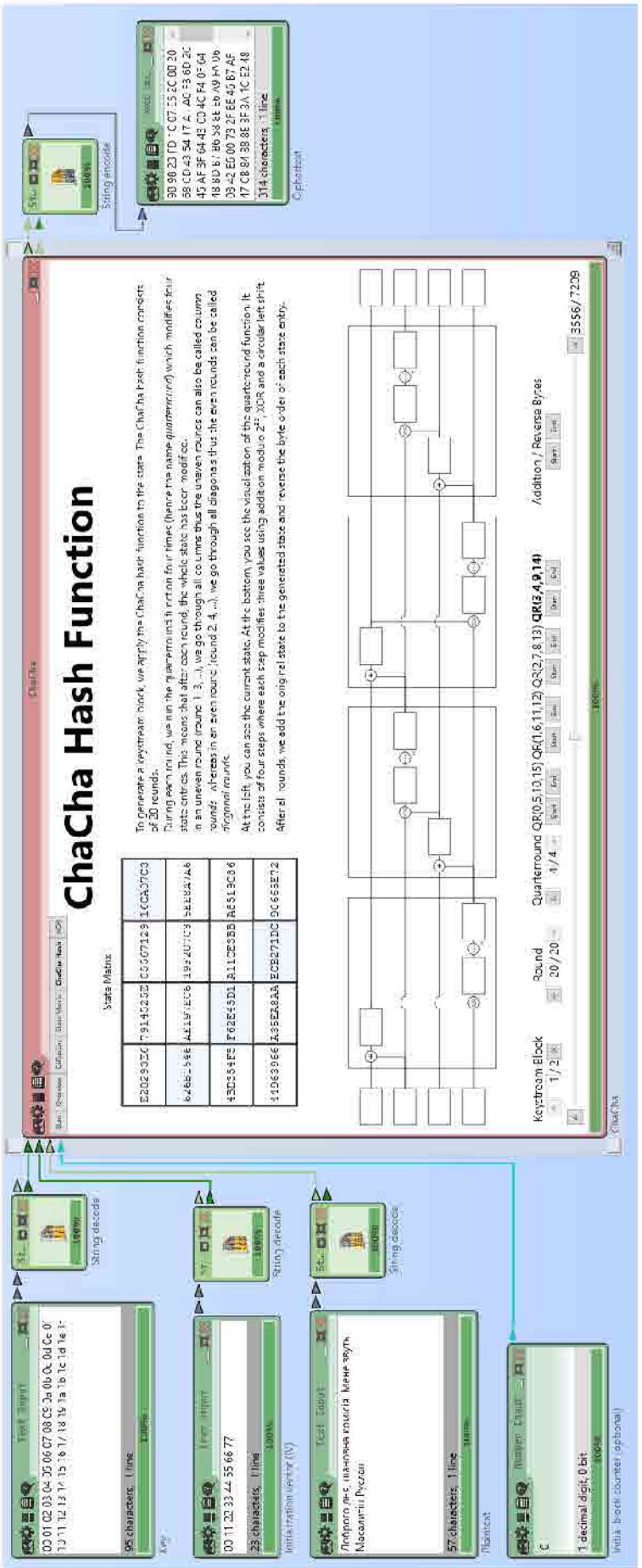



Рисунок 1.36. Результат моделювання зашифрування поза методом Cha-cha при розрахунку хеш-функцій

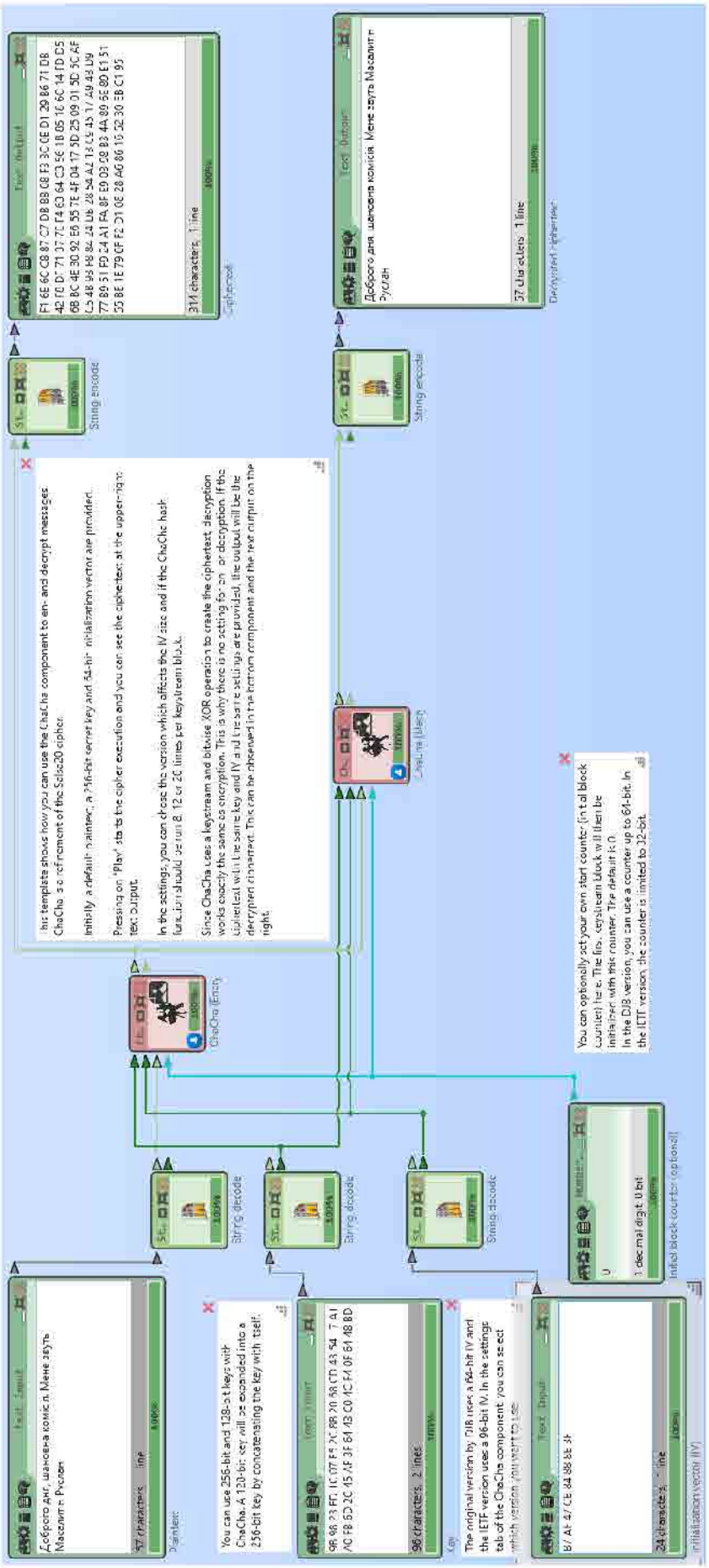


Рисунок 1.37. Результат моделювання зашифрування і декодування поза методом Cha-cha

### 1.4.8 Розробка програмного коду і створення інтерфейсу

Текст основної процедури зашифрування поза методом Cha-cha розробленої програми мовою C++ наведено в Додатку А. Розробка виконано в інтегрованому середовищі розробки QT під керуванням операційної системи Windows 11 Pro x64. Інтерфейс головного вікна розробленої програми-кодеку наведено у рис. 1.38.

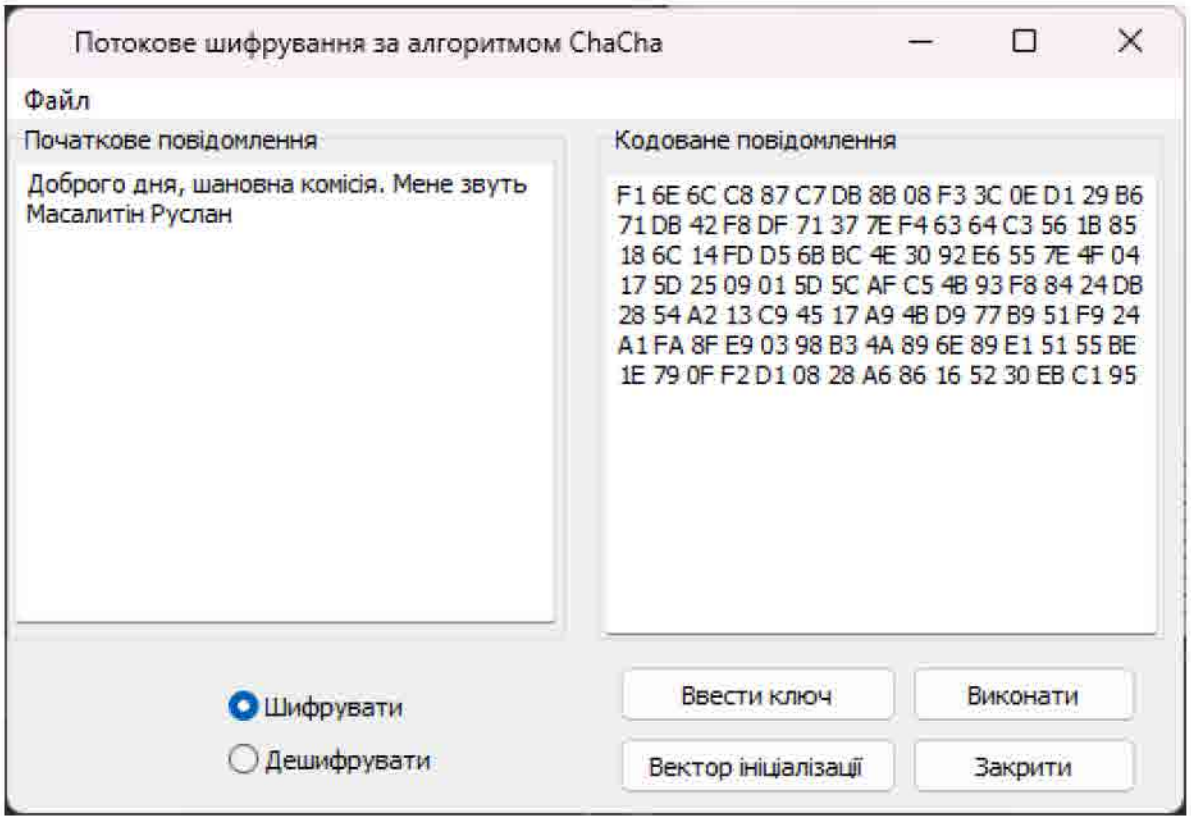



Рисунок 1.38. Головне вікно програми-кодеку у базі процесу Cha-cha

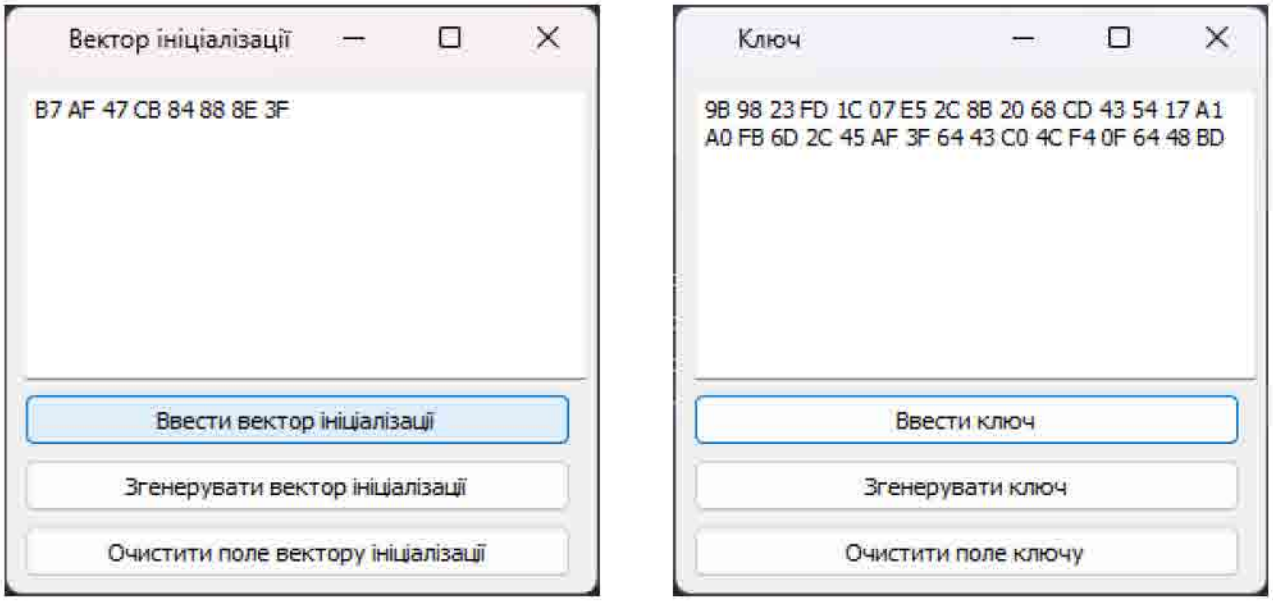



Рисунок 1.39. Генерування чи введення ключу і вектору ініціалізування

Задля зашифрування початкового сповіщення треба в поле «Початкове сповіщення» ввести текст і вибрати опцію «Шифрувати». При цьому спочатку треба натиснути у кнопки «Ввести код» і «Вектор ініціалізування», після чого відкриються вікна генерування ключу і вектору ініціалізування (рис. 1.39).

Після закриття вікна генерування ключу відбувається перехід щодо основного вікна. При натисненні кнопки «Виконати» шифрований чи дешифрований поза поміччю Cha-cha текст із'явиться в полі «Кодоване сповіщення» (рис. 1.40).

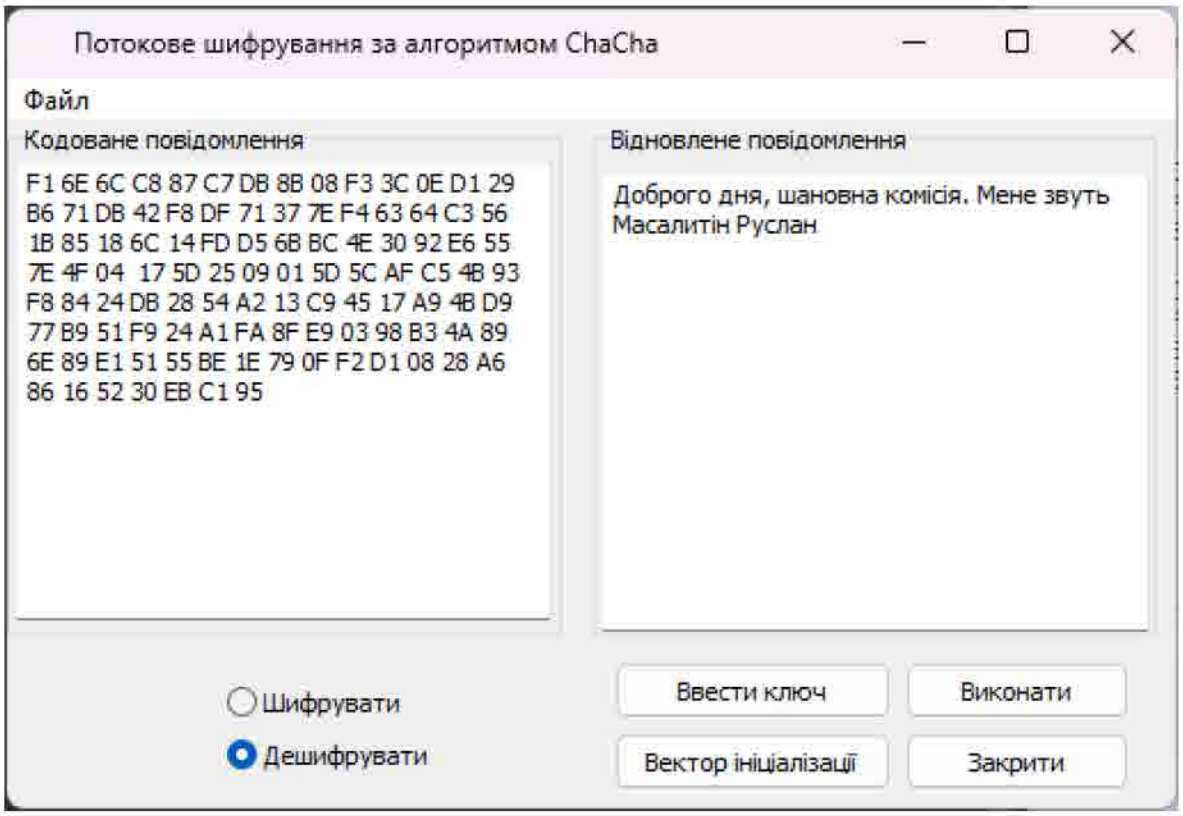



Рисунок 1.40. Приклад декодування програмою-кодеком сповіщення

При відсутності згенерованого ключу / вектору ініціалізування зашифрування декодування повідомлень вийде невдалим, у результаті чого передбачене виведення сповіщення про помилку (рис. 1.41).

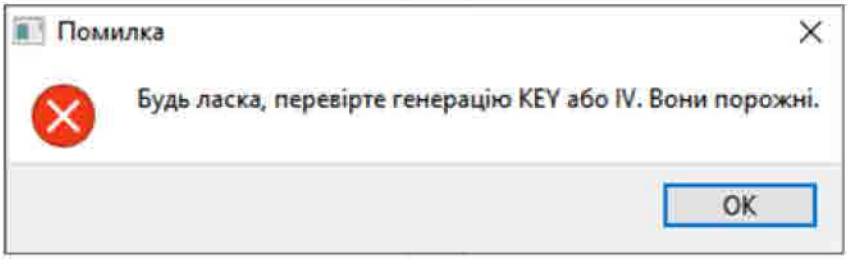



Рисунок 1.41. Сповіщення про відсутність ключу / вектору ініціалізування

Поза наведеними вище скріншотами (рис.1.38, рис.1.40) видно, коли зашифровані/дешифровані розробленим кодеком сповіщення повністю співпадають із результатами модулювання в програмі Cryptool2, наведеними у рис.1.37, коли свідчить про правильну роботу розробленого додатку.

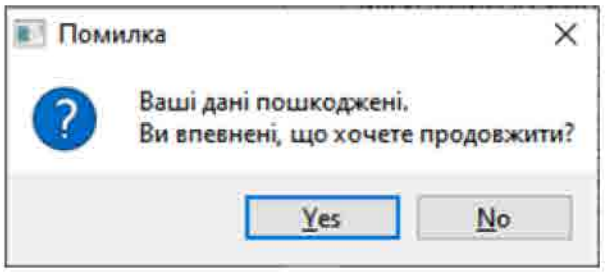



Рисунок 1.42. Сповіщення про порушення цілісності бітів

Задля перевірки цілісності бітів генерується хеш із вихідних бітів і порівнюється із тим, що був згенерований щодо цього. В випадку, коли вони не співпадають, система видає помилку про таке, коли дані були змінені (рис. 1.42). При натисканні «Yes» програма вийде намагатися декодувати пошкоджені дані, коли призведе щодо неправильних вихідних бітів.

# 2 РОЗДІЛ ОХОРОНИ ПРАЦІ І ТЕХНІКИ БЕЗПЕКИ

Широке впровадження комп'ютерної техніки, коли дає змогу автоматизувати багато рутинних операцій комп'ютерної обробки даних, одержати доступ щодо численних джерел даних, швидко проводити потрібні розрахунки тощо, підвищує продуктивність праці. Проте активне впровадження в практику персональних комп'ютерів можливе та негативну сторону – із'являються фактори, що несприятливо впливають у здоров'я працюючої людини.

Згідно темі дипломного проекту робоче місце користувача послуг складається із персонального комп'ютеру із програмним забезпеченням і призначено задля розробки моделі кодексу у базі процесу симетричного зашифрування.

Саме тому задля нього застосовуються звичайні вимоги безпеки праці задля користувача персонального комп'ютеру.

## 2.1 Аналіз небезпечних та шкідливих факторів, коли впливають у користувача ПК

Показниками гігієнічних вимог являється рівень освітлення, температура, вологість, шум, вібрація, токсичність, загазованість, обмежена загальна м'язова активність (гіподинамії), дія електростатичного поля, неіонізуючих і іонізуючих електромагнітних випромінювань.

## 2.2 Гігієнічні вимоги щодо виробничого середовища

Державні санітарні правила та норми дії із візуальними дисплейними терміналами електронно-обчислювальних машин ДСАНПіН 3.3.2.007-98 «Гігієнічні вимоги щодо організації дії із візуальними дисплейними терміналами електронно-обчислювальних машин» призначені задля запобігання несприятливої дії у працівників шкідливих факторів, що супроводжують роботу із ВДТ

### 2.2.1 Вимоги щодо приміщення

Розміщення робочих місць із ВДТ ЕОМ та ПЕОМ в підвальних приміщеннях, у цокольних поверхах заборонено. Задля приміщень, що призначені задля дії із ВДТ, доцільно обрати орієнтацію вікон у північ чи північний схід. У вікнах повинні мати можливість жалюзі, коли регулюються, чи штори, коли дають можливість їх

повністю закривати.

Об'ємно-планувальні рішення будівель і приміщень, де експлуатуються відеодисплейні термінали (ВДТ) повинні відповідати вимогам ДСанПІН 3.3.2.007-98

Площа у одне робоче місце задля програмістів повинна складати не менше 6 кв.м., проте об'єм – не менше 20 куб.м. Стіни повинні мати можливість пофарбовані матовою фарбою.

При приміщеннях із ВДТ повинні мати можливість обладнані побутові приміщення задля відпочинку, психологічного розвантаження тощо

### 2.2.2 Освітлення

Задля освітлення приміщення, в якому працює програміст, застосовується змішане освітлення, тобто сполучення природного та штучного освітлення. Задля загального освітлення приміщення, де перебуває робоче місце програміста, використовуються газорозрядні лампи типу ЛД. Нормами задля бітів робіт установлена необхідна освітленість робочого місця ЕН=300 лк (задля робіт високої точності, коли найменший розмір об'єкта розрізнення дорівнює 0,3 – 0,5 мм).
Дана норма освітленості у цілому виконана

### 2.2.3 Шум

В робочих приміщеннях основними джерелами акустичних шумів являється шуми ПЕОМ. ЕОМ являється також джерелами шумів електромагнітного походження (коливання елементів електромеханічних пристроїв під впливом змінних магнітних полів). Крім того, в бітів приміщеннях, виникає структурний шум, тобто шум, випромінюваний поверхнями коливних конструкцій стін, перекриттів, перегородок будинку в звуковому діапазоні частот.. Задля усунення чи ослаблення несприятливих шумових впливів доцільно ізолювати робочі приміщення, розміщаючи їх в частинах будинку, найбільш вилучених з міського шуму - розташованих в глибині будинку, звернених вікнами в двір. Перевіряти герметичність корпусів комп'ютерів і своєчасно міняти вентилятори охолодження.

## 2.3 Вимоги щодо організації робочого місця працівника

Конструкція робочого місця користувача ПК та взаємне розташування всіх його елементів (сидіння, органи керування, засобу відображення даних) відповідають антропометричним, фізіологічним та психологічним вимогам, проте також характеру дії. Конструкція робочих меблів повинна забезпечувати можливість індивідуального регулювання відповідно росту працюючих задля підтримки зручної пози. Робочий стіл повинен мати можливість пофарбований матовою фарбою. Дисплей розташований так, коли його верхній край перебуває у рівні очей у відстані близько 70 см, коли укладається у в припустимі рамки з 60 щодо 90 см. Частота мерехтіння екрана fмер=100 Гц, коли відповідає умові fмер>70 Гц.

Робоче місце розташоване перпендикулярно віконним прорізам, це зроблено із тією метою, аби виключити пряму та відбиту мерехтливість екрана з вікон та приладів штучного освітлення.

## 2.4 Мікроклімат

Параметри мікроклімату, іонного складу повітря, вміст шкідливих речовин у робочих місцях, оснащених ПК повинні відповідати вимогам ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень».

Задля підтримки допустимих значень мікроклімату і концентрації позитивних і негативних іонів треба передбачати установки чи прилади зволоження і/чи штучної іонізації, кондиціювання повітря. Рівні інфрачервоного випромінювання не повинні перевищувати граничних

Рівні інфрачервоного випромінювання не повинні перевищувати граничних відповідно щодо ГОСТ 12.1.005. Вміст озону у повітрі робочої зони не повинен перевищувати 0,1мг/м3; вміст оксидів азоту - 5мг/м3; вміст пилу - 4мг/м3.

## 2.5 Електробезпека

Приміщення де використовуються імпульсні джерела живлення відповідно щодо ОНТП24-86 та ПУЕ-87 відноситься щодо класу приміщень без підвищеної небезпеки поразки персоналу електричним струмом, оскільки відносна вологість

повітря не перевищує 75%, температура не більш 35ºС, відсутні хімічно агресивні середовища. Живлення електроприладів усередині приміщення здійснюється з двухфазної мережі із заземленою нейтралю напругою 220 У та частотою 50 Гц із використанням автоматів токового захисту. В приміщенні повинна мати можливість застосована схема заземлення. Ураження людини електричним струмом може відмати можливість в випадку:

1) дотику щодо відкритих струмоведучих частин;

2) в результаті дотику щодо струмопровідних неструмоведучих елементів устаткування, коли опинилися під напругою у результаті порушення ізоляції чи із інших причин.

Заземлення повинно мати можливість зроблено поза поміччю гнучкого сплетеного мідного проводу діаметром порядку 1,5 мм2

Задля зменшення значень напруг дотику та відповідних їм величин струмів, при нормальному та аварійному режимах дії устаткування треба виконати повторне захисне заземлення нульового проводу.

Відповідно щодо ГОСТ-12.2.007.0-75 устаткування (крім ЕОМ - II клас) відноситься щодо I класу, воно можливе робочу ізоляцію відповідно щодо вимог ГОСТ 12.1.009-76. Підключення устаткування виконане відповідно щодо вимог ПБЕ і ПУЕ. Додаткових заходів по електробезпечності не потрібно.

## 2.6 Пожежна безпека

Робоче приміщення відповідно щодо ПБЕ і ОНТП 24–86 по вибухово-пожарній безпеці можливо віднести щодо категорії "У".

Можливими причинами виникнення пожежі у приміщенні являється:

1) коротке замикання проводки;
2) користування побутовими електрорадіоприладами.
3) не дотримання умов протипожежної безпеки.

В зв'язку із цим відповідно щодо ПУЕ треба передбачити наступні заходи щодо пожежної безпеки: ретельна ізоляція всіх струмоведучих провідників щодо робочих місць; періодичний огляд та перевірка ізоляції; суворе дотримання норм

протипожежної безпеки у робочому місці.

Задля гасіння пожеж у робочому місці користувача ПК використовують вуглекислотні і порошкові вогнегасники.

– Вуглекислотні вогнегасники випускаються коли ручні (ВВК-5);

– Порошкові вогнегасники ВП-2, ВП-5, ВП-10 і ін.


Рисунок 2.1. Засоби пожежогасіння

ІЗ метою своєчасного оповіщення, у дільниці треба встановити протипожежну сигналізацію. Проходи і запасні виходи повинні мати можливість вільними. Пожежний щит повинен розміщуватись у доступному місці і містити первинні засоби пожежогасіння: вогнегасник, лопату, відро, простирадло, ящик із піском. Відповідальний поза пожежну безпеку керівник виробничої дільниці.

# ВИСНОВКИ

Протягом здійснення випускної дії були проаналізовані основні способи поточного зашифрування бітів, визначені їх переваги і недоліки, способи щодо їх злому, проаналізовано швидкість дії потокових методів кодування, обрано надійні генератори псевдовипадкових значень задля реалізації констант і паролів. Поза поміччю програмного забезпечення задля моделювання криптографічних методів CrypTool2 виконано моделювання дії поточного криптографічного процесу Cha-cha, коли являється одним із найбільш ефективних і захищених методів симетричного поточного зашифрування у даний час. Це дозволило виконати розробку програмної моделі кодеку Cha-cha мовою C++ засобами інтегрованого середовища розробки QT.

Після проведеного аналізу криптографічних і криптоаналітичних методів можливо зазначити, коли проблема створення надійного процесу зашифрування вийде завжди у високому рівні дослідження і вирішення, адже із розвитком обчислювальних можливостей комп'ютерної техніки зменшуються періоди задля встановлення вразливостей і колізій задля крипто-методів.

Вивчені під час здійснення випускної дії способи поточного зашифрування спроможні мати можливість корисними не тільки задля захисту локальних бітів користувача, але та задля створення в майбутньому web-сервісу задля захисту хмарних бітів. Розроблений кодек можливе візуальний інтерфейс задля тестування дії криптографічного процесу Cha-cha, додаток підтримується операційними системами сімейства Windows і не вимагає багато ресурсів задля своєї дії.

Подальше вдосконалення дії реалізованого кодеку задля поточного зашифрування Cha-cha може полягати в застосуванні методів зашифрування задля передачі ключу і вектору ініціалізування, збільшенні їх розмірів, надійному зберіганні, можливості застосування генераторів стохастичних значень, що будуть залежати з величин, коли не можливо передбачити, збільшенні циклів кодування.

Реалізований програмний застосунок представлений в вигляді моделі, придатної задля подальшої реалізації інших потокових методів зашифрування бітів, розглянутих в даній роботі.

# ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ


1. Щур Н.О., Покотило О.ПРОТЕ. Основи криптології: навч. посібник. – Житомир: Державний університет «Житомирська політехніка», 2021. 120 с.
2. Козюра У.Д., Хорошко У. О., Шелест М. ЯВЛЯЄТЬСЯ., Ткач Ю. М., Балюнов О.О. Захист даних у комп'ютерних системах: підручник. – Ніжин: ФОП Лук'яненко У.У., ТПК «Орхідея», 2020. – 236с.
3. Глинчук Л.Я. Криптологія: навч.-спосіб. посіб. / Л. Я. Глинчук – Луцьк:Вежа-Друк, 2014. – 164 с.
4. Козіна Г.Л. Криптографія з історії щодо сучасних стандартів: навч.посібник /Г. Л. Козіна. – Запоріжжя : НУ «Запорізька політехніка», 2020. – 192 с.
5. Корченко О.Г. Прикладна криптологія: системи зашифрування : підручник /О. Г. Корченко, У. П. Сіденко, Ю. О. Дрейс. – К. : ДУТ, 2014. – 448 с.
6. Горбенко ТА.Д. // Прикладна криптологія: Теорія. Практика. Впровадження / ТА.Д. Горбенко, Ю. ТА. Горбенко. – Харків : Форт, 2013. – 880 с.
7. Stroustrup B. A Tour of C++ (Second Edition). – Addison-Wesley, 2018. – 240 p. – ISBN 978-0-13-499783-4.
8. Бублик У.У. Об'єктно-орієнтоване програмування. Київ: ІТ-книга, 2015.
9. Богач І.У., Довгалець С. М., Дубовой У. М. Способи розв'язання задач із програмування. Розв'язник. Вінниця: ВНТУ, 2017. – 119 с.
10. Кравець П. Об'єктно-орієнтоване програмування : навч. посібник / П.О. Кравець. – Львів: Видавництво Львівської політехніки, 2012. – 624 с.
11. Authenticated encryption. [Електронний ресурс]. https://www.cryptopp.com/wiki/Authenticated_Encryption
12. Symmetric And Asymmetric Key Cryptography: All You Need To Know In 3 Points. [Електронний ресурс]. https://www.jigsawacademy.com/blogs/cyber-security/symmetric-and-asymmetric-key-cryptography/
13. Secure Hash Algorithms. [Електронний ресурс]. https://brilliant.org/wiki/secure-hashing- algorithms/
14. SHA-512 Hashing Algorithm Overview. [Електронний ресурс]. https://komodoplatform.com/en/blog/sha-512/

# ДОДАТОК А

## Код основного класу ChaCha мовою C++

```
#include <cstdint>
#include <limits>
template<size_t R>
class ChaCha {
public:
    typedef uint32_t result_type;
    explicit ChaCha(uint64_t seedval, uint64_t stream = 0);
    template<class Sseq> explicit ChaCha(Sseq& seq);
    void seed(uint64_t seedval, uint64_t stream = 0);
    template<class Sseq> void seed(Sseq& seq);
    uint32_t operator()();
    void discard(unsigned long long n);
        template<size_t R_> friend bool operator==(const ChaCha<R_>& lhs, const ChaCha<R_>& rhs);
    template<size_t R_> friend bool operator!=(const ChaCha<R_>& lhs, const ChaCha<R_>& rhs);
    template<typename CharT, typename Traits>
    friend std::basic_ostream<CharT, Traits>& operator<<(std::basic_ostream<CharT, Traits>& os, const
ChaCha<R>& rng);
    template<typename CharT, typename Traits>
    friend std::basic_istream<CharT, Traits>& operator>>(std::basic_istream<CharT, Traits>& is, ChaCha<R>& rng);
    static constexpr uint32_t min() { return std::numeric_limits<uint32_t>::min(); }
    static constexpr uint32_t max() { return std::numeric_limits<uint32_t>::max(); }
private:
    void generate_block();
    void chacha_core();
    alignas(16) uint32_t block[16];
    uint32_t keysetup[8];
    uint64_t ctr;
};
template<size_t R>
inline ChaCha<R>::ChaCha(uint64_t seedval, uint64_t stream) {
    seed(seedval, stream);
}
template<size_t R>
template<class Sseq>
inline ChaCha<R>::ChaCha(Sseq& seq) {
    seed(seq);
}
template<size_t R>
inline void ChaCha<R>::seed(uint64_t seedval, uint64_t stream) {
    ctr = 0;
    keysetup[0] = seedval & 0xffffffffu;
    keysetup[1] = seedval >> 32;
    keysetup[2] = keysetup[3] = 0xdeadbeef;
    keysetup[4] = stream & 0xffffffffu;
    keysetup[5] = stream >> 32;
    keysetup[6] = keysetup[7] = 0xdeadbeef;
}
template<size_t R>
template<class Sseq>
inline void ChaCha<R>::seed(Sseq& seq) {
    ctr = 0;
    seq.generate(keysetup, keysetup + 8);
}
template<size_t R>
inline uint32_t ChaCha<R>::operator()() {
    int idx = ctr % 16;
    if (idx == 0) generate_block();
    ++ctr;
    return block[idx];
}
```

```
template<size_t R>
inline void ChaCha<R>::discard(unsigned long long n) {
  int idx = ctr % 16;
  ctr += n;
  if (idx + n >= 16 && ctr % 16 != 0) generate_block();
}
template<size_t R>
inline void ChaCha<R>::generate_block() {
  uint32_t constants[4] = {0x61707865, 0x3320646e, 0x79622d32, 0x6b206574};

  uint32_t input[16];
  for (int i = 0; i < 4; ++i) input[i] = constants[i];
  for (int i = 0; i < 8; ++i) input[4 + i] = keysetup[i];
  input[12] = (ctr / 16) & 0xffffffffu;
  input[13] = (ctr / 16) >> 32;
  input[14] = input[15] = 0xdeadbeef;

  for (int i = 0; i < 16; ++i) block[i] = input[i];
  chacha_core();
  for (int i = 0; i < 16; ++i) block[i] += input[i];
}
#ifdef __SSE2__
#include "emmintrin.h"
#if !defined(__XOP__)
  #if defined(__SSSE3__)
    #include <tmmintrin.h>
    #define _mm_roti_epi32(r, c) (                    \
      ((c) == 8) ?                                    \
        _mm_shuffle_epi8((r), _mm_set_epi8(14, 13, 12, 15, \
                                           10,  9,  8, 11, \
                                           6,  5,  4,  7, \
                                           2,  1,  0,  3)) \
      : ((c) == 16) ?                                 \
        _mm_shuffle_epi8((r), _mm_set_epi8(13, 12, 15, 14, \
                                           9,  8, 11, 10, \
                                           5,  4,  7,  6, \
                                           1,  0,  3,  2)) \
      : ((c) == 24) ?                                 \
        _mm_shuffle_epi8((r), _mm_set_epi8(12, 15, 14, 13, \
                                           8, 11, 10,  9, \
                                           4,  7,  6,  5, \
                                           0,  3,  2,  1)) \
      :                                               \
        _mm_xor_si128(_mm_slli_epi32((r), (c)),       \
                      _mm_srli_epi32((r), 32-(c)))    \
    )
  #else
    #define _mm_roti_epi32(r, c) _mm_xor_si128(_mm_slli_epi32((r), (c)), \
                                       _mm_srli_epi32((r), 32-(c)))
  #endif
#else
  #include <xopintrin.h>
#endif
template<size_t R>
inline void ChaCha<R>::chacha_core() {
  // ROTVn rotates the elements in the given vector n places to the left.
  #define CHACHA_ROTV1(x) _mm_shuffle_epi32((__m128i) x, 0x39)
  #define CHACHA_ROTV2(x) _mm_shuffle_epi32((__m128i) x, 0x4e)
  #define CHACHA_ROTV3(x) _mm_shuffle_epi32((__m128i) x, 0x93)
  __m128i a = _mm_load_si128((__m128i*) (block));
  __m128i b = _mm_load_si128((__m128i*) (block + 4));
  __m128i c = _mm_load_si128((__m128i*) (block + 8));
  __m128i d = _mm_load_si128((__m128i*) (block + 12));
  for (int i = 0; i < R; i += 2) {
    a = _mm_add_epi32(a, b);
```

```
      d = _mm_xor_si128(d, a);
      d = _mm_roti_epi32(d, 16);
      c = _mm_add_epi32(c, d);
      b = _mm_xor_si128(b, c);
      b = _mm_roti_epi32(b, 12);
      a = _mm_add_epi32(a, b);
      d = _mm_xor_si128(d, a);
      d = _mm_roti_epi32(d, 8);
      c = _mm_add_epi32(c, d);
      b = _mm_xor_si128(b, c);
      b = _mm_roti_epi32(b, 7);
      b = CHACHA_ROTV1(b);
      c = CHACHA_ROTV2(c);
      d = CHACHA_ROTV3(d);
      a = _mm_add_epi32(a, b);
      d = _mm_xor_si128(d, a);
      d = _mm_roti_epi32(d, 16);
      c = _mm_add_epi32(c, d);
      b = _mm_xor_si128(b, c);
      b = _mm_roti_epi32(b, 12);
      a = _mm_add_epi32(a, b);
      d = _mm_xor_si128(d, a);
      d = _mm_roti_epi32(d, 8);
      c = _mm_add_epi32(c, d);
      b = _mm_xor_si128(b, c);
      b = _mm_roti_epi32(b, 7);
      b = CHACHA_ROTV3(b);
      c = CHACHA_ROTV2(c);
      d = CHACHA_ROTV1(d);
    }
    _mm_store_si128((__m128i*) (block), a);
    _mm_store_si128((__m128i*) (block + 4), b);
    _mm_store_si128((__m128i*) (block + 8), c);
    _mm_store_si128((__m128i*) (block + 12), d);
    #undef CHACHA_ROTV3
    #undef CHACHA_ROTV2
    #undef CHACHA_ROTV1
}
#else
template<size_t R>
inline void ChaCha<R>::chacha_core() {
    #define CHACHA_ROTL32(x, n) (((x) << (n)) | ((x) >> (32 - (n))))
    #define CHACHA_QUARTERROUND(x, a, b, c, d) \
      x[a] = x[a] + x[b]; x[d] ^= x[a]; x[d] = CHACHA_ROTL32(x[d], 16); \
      x[c] = x[c] + x[d]; x[b] ^= x[c]; x[b] = CHACHA_ROTL32(x[b], 12); \
      x[a] = x[a] + x[b]; x[d] ^= x[a]; x[d] = CHACHA_ROTL32(x[d],  8); \
      x[c] = x[c] + x[d]; x[b] ^= x[c]; x[b] = CHACHA_ROTL32(x[b],  7)
    for (int i = 0; i < R; i += 2) {
      CHACHA_QUARTERROUND(block, 0, 4,  8, 12);
      CHACHA_QUARTERROUND(block, 1, 5,  9, 13);
      CHACHA_QUARTERROUND(block, 2, 6, 10, 14);
      CHACHA_QUARTERROUND(block, 3, 7, 11, 15);
      CHACHA_QUARTERROUND(block, 0, 5, 10, 15);
      CHACHA_QUARTERROUND(block, 1, 6, 11, 12);
      CHACHA_QUARTERROUND(block, 2, 7,  8, 13);
      CHACHA_QUARTERROUND(block, 3, 4,  9, 14);
    }
    #undef CHACHA_QUARTERROUND
    #undef CHACHA_ROTL32
}
#endif
template<size_t R>
inline bool operator==(const ChaCha<R>& lhs, const ChaCha<R>& rhs) {
    for (int i = 0; i < 8; ++i) {
      if (lhs.keysetup[i] != rhs.keysetup[i]) return false;
    }
```

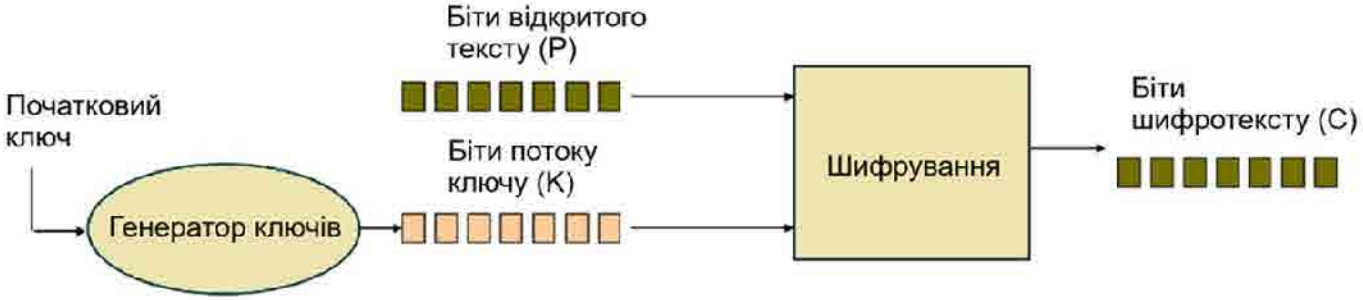
```
    return lhs.ctr == rhs.ctr;
}
template<size_t R>
inline bool operator!=(const ChaCha<R>& lhs, const ChaCha<R>& rhs) { return !(lhs == rhs); }
template<size_t R, typename CharT, typename Traits>
inline std::basic_ostream<CharT, Traits>& operator<<(std::basic_ostream<CharT, Traits>& os, const ChaCha<R>&
rng) {
    typedef typename std::basic_ostream<CharT, Traits>::ios_base ios_base;
    auto flags = os.flags();
    auto fill = os.fill();
        auto space = os.widen(' ');
    os.flags(ios_base::dec | ios_base::fixed | ios_base::left);
    os.fill(space);
    // Serialize.
    for (int i = 0; i < 8; ++i) os << rng.keysetup[i] << space;
    os << rng.ctr;
    // Sestore old state.
    os.flags(flags);
    os.fill(fill);
    return os;
}
template<size_t R, typename CharT, typename Traits>
inline std::basic_istream<CharT, Traits>& operator>>(std::basic_istream<CharT, Traits>& is, ChaCha<R>& rng) {
    typedef typename std::basic_istream<CharT, Traits> ::ios_base ios_base;
    auto flags = is.flags();
    is.flags(ios_base::dec);
    for (int i = 0; i < 8; ++i) is >> rng.keysetup[i];
    is >> rng.ctr;
    is.flags(flags);
    return is;
}
```

**ДОДАТОК Б**

**Слайди мультимедійної презентації**

## Узагальнена схема симетричного шифрування


## Принципи потокового шифрування



# Застосування операції XOR для потокового шифрування

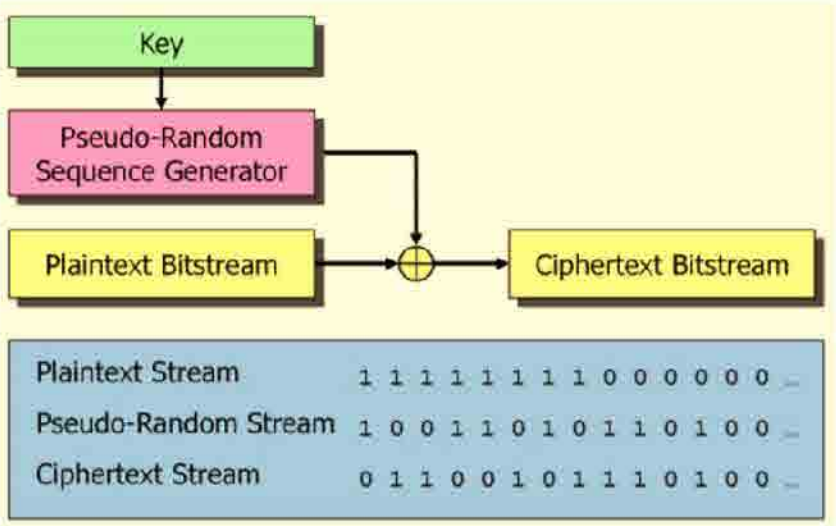
**Для шифрування бінарних даних (потоку бітів)**

**Ключ:** послідовність випадкових бітів

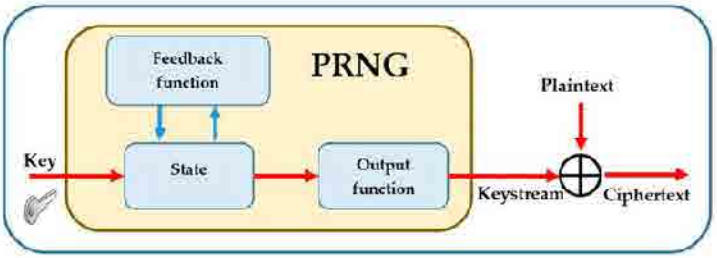
| ⊕ | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

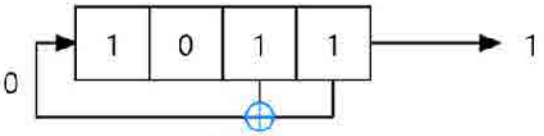
Виконується додавання бітів за модулем 2 (операція XOR, eXclusive OR – виключне або)

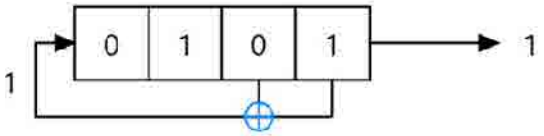
# Реалізація потокового шифрування

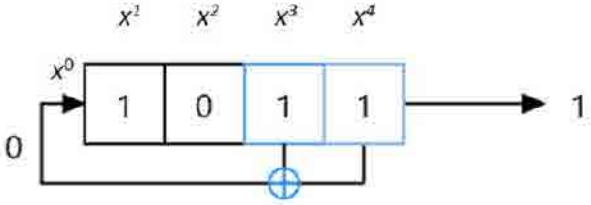


# Застосування генераторів псевдовипадкових чисел



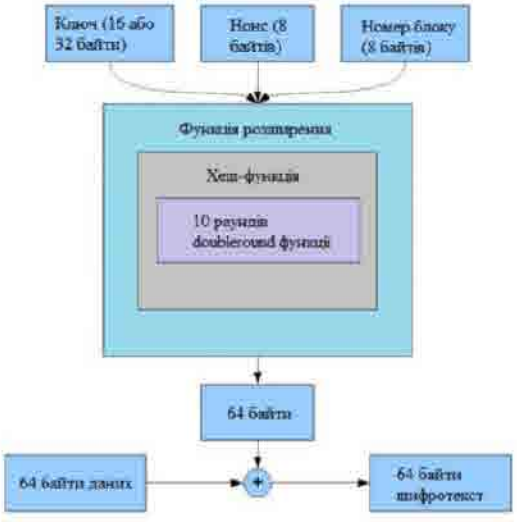




# Принцип хешування за допомогою хеш-функції

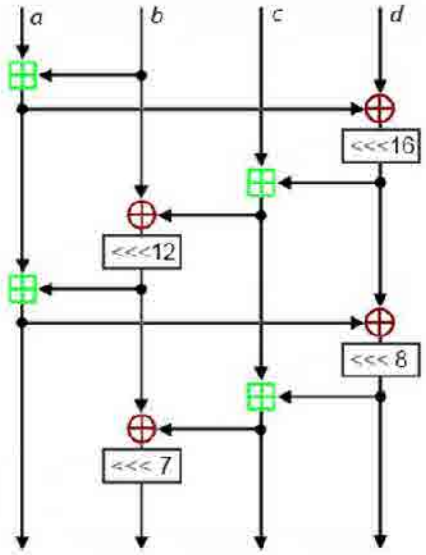
# Швидкість потокових криптоалгоритмів

| Потоковий криптоалгоритм | Швидкість Мбіт/с |
|---|---|
| Криптоалгоритм RC4 | 139 |
| Криптоалгоритм A5 | 485 |
| Криптоалгоритм SEAL | 657 |
| Криптоалгоритм Salsa20 | 780 |
| Криптоалгоритм SNOW | 711 |
| Криптоалгоритм Sosemanuk | 602 |
| Криптоалгоритм ChaCha | 813 |
| Криптоалгоритм Rabbit | 579 |
| Криптоалгоритм HC-128 | 489 |
| Криптоалгоритм Grain | 689 |
| Криптоалгоритм MICKEY | 581 |
| Криптоалгоритм Trivium | 673 |
| Криптоалгоритм OTP | 591 |


# Блок-схеми алгоритму SALSA20, на якому заснований алгоритм ChaCha



# Цикл роботи алгоритму ChaCha функції чверті раунду quarterround(a, b, c, d)




$$a+=b;\ \ d\oplus=a;\ \ d\lll=16;$$
$$c+=d;\ \ b\oplus=c;\ \ b\lll=12;$$
$$a+=b;\ \ d\oplus=a;\ \ d\lll=8;$$
$$c+=d;\ \ b\oplus=c;\ \ b\lll=7;$$

# Спрощена схема вихорю Мерсенна

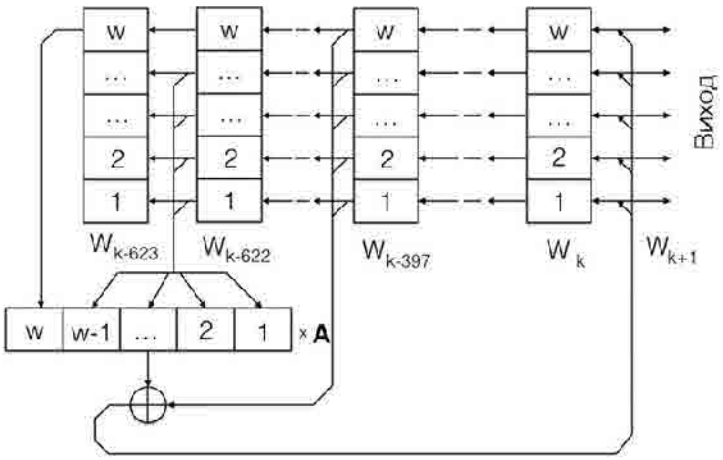
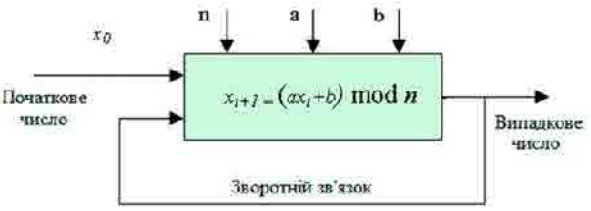



Схема реалізації алгоритму **BBS**



# Структура алгоритму SHA-512

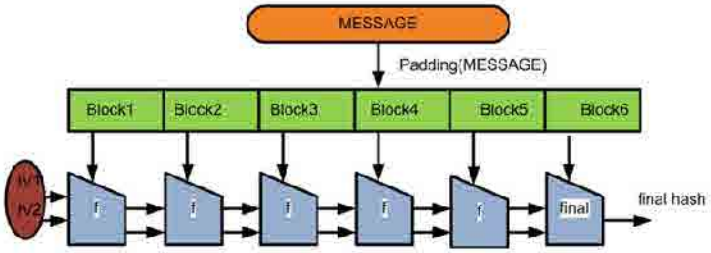
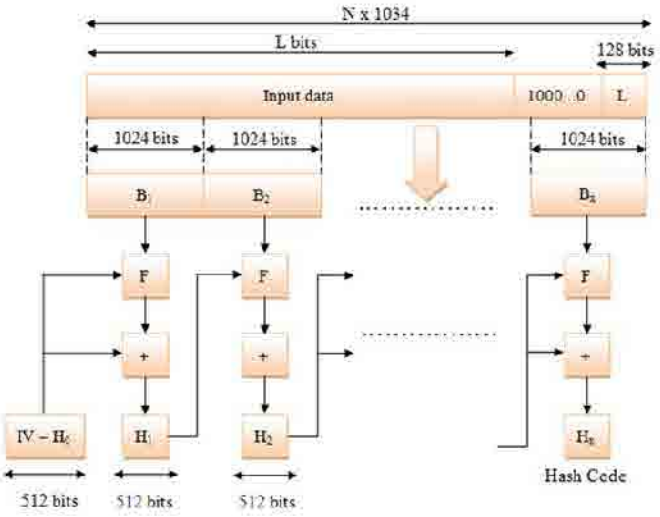
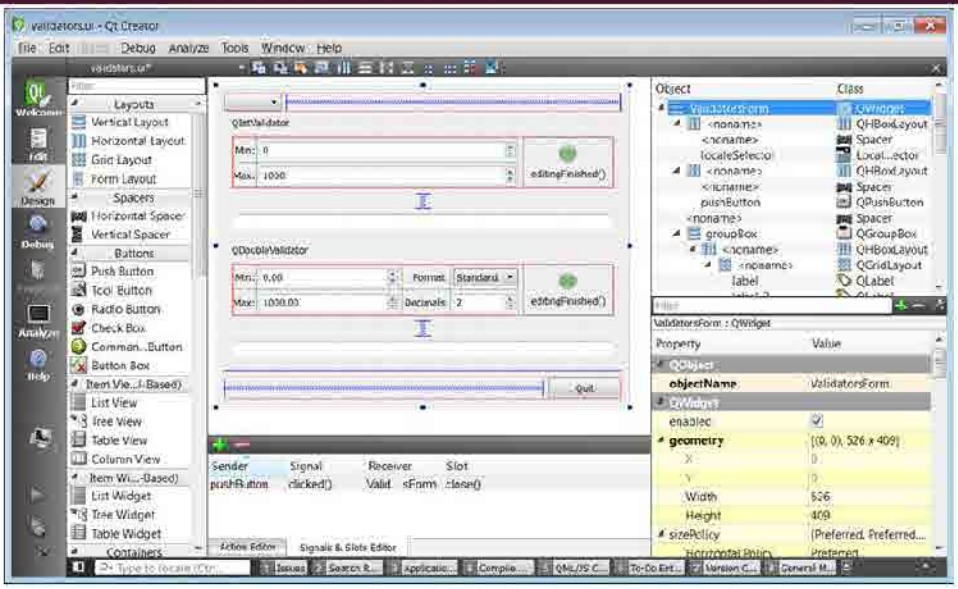


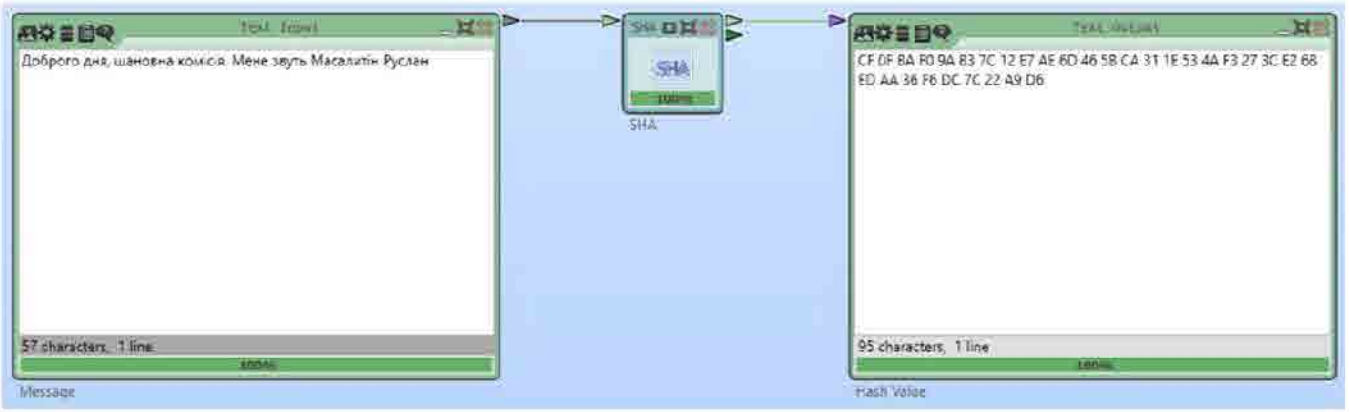

Схема реалізації алгоритму Меркла-Демґарда для хешування



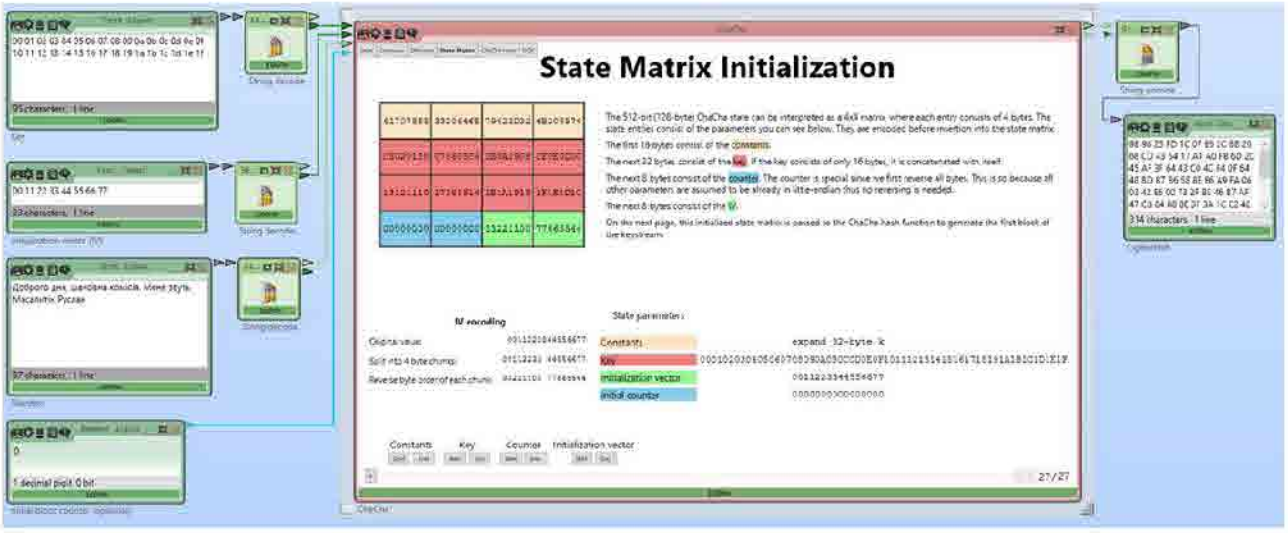

# Інтерфейс IDE QT Creator



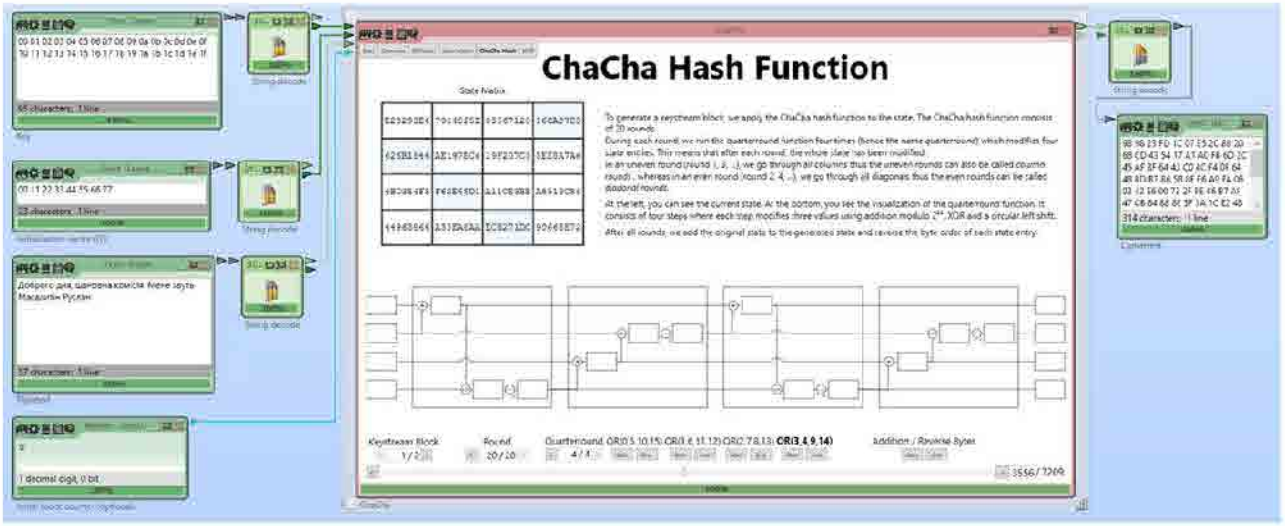
# Результат моделювання за алгоритмом SHA-512



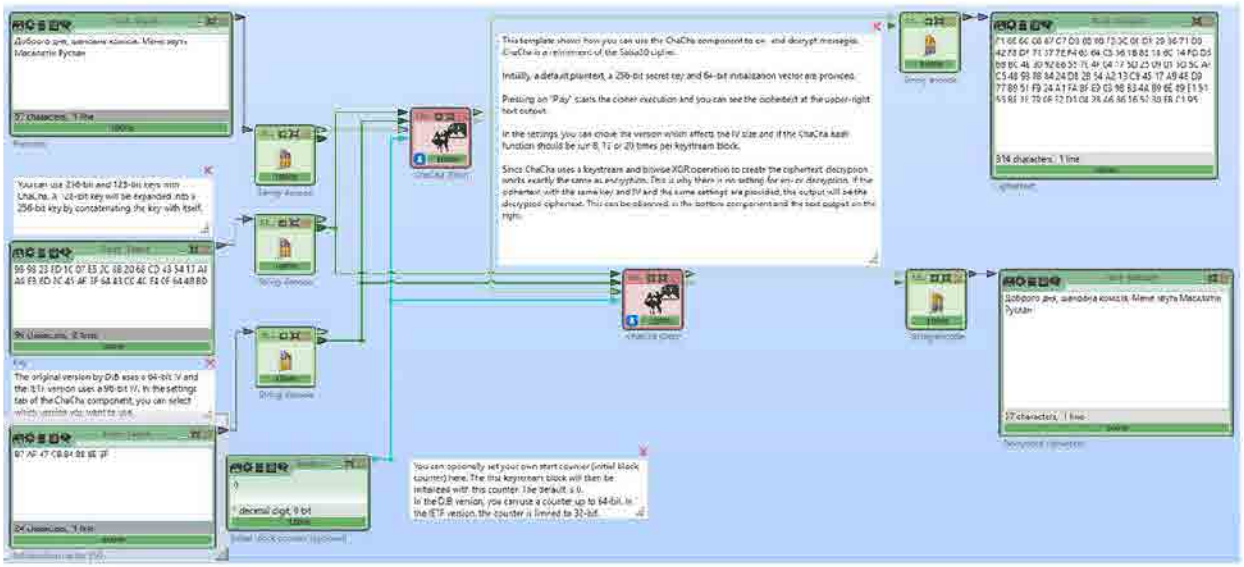

# Результат моделювання шифрування за алгоритмом ChaCha при ініціалізації матриці стану



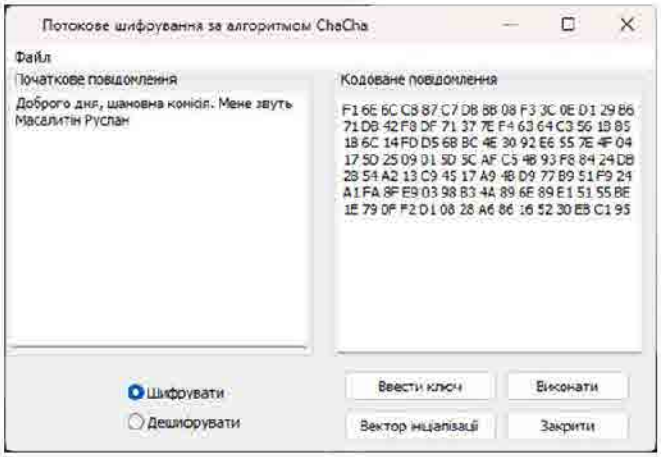
# Результат моделювання шифрування за алгоритмом ChaCha при розрахунку хеш-функцій

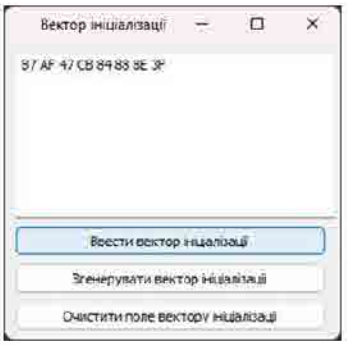



# Результат моделювання шифрування та дешифрування за алгоритмом ChaCha



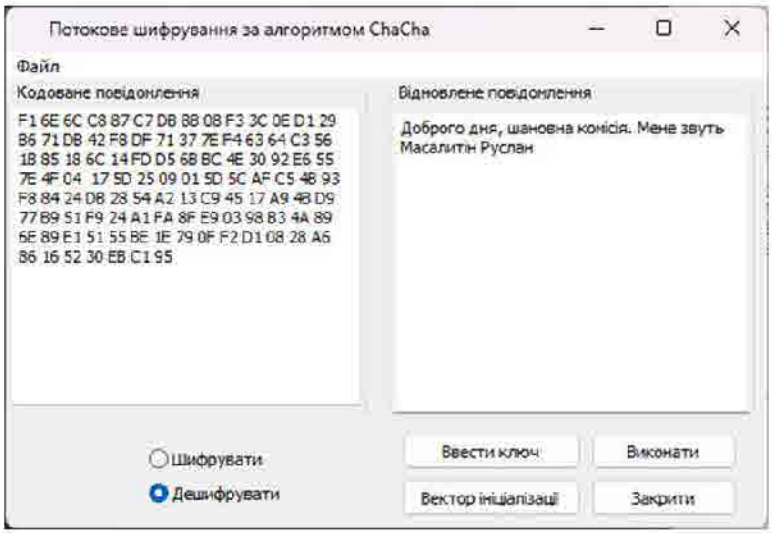
# Головне вікно програми-кодеку на базі алгоритму ChaCha





# Приклад дешифрування програмою-кодеком повідомлення



ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

# РЕЦЕНЗІЯ

на кваліфікаційну роботу бакалавра здобувача освіти

відділення комп'ютерних систем

Масалитіна Руслана Вікторовича

(прізвище, ім'я та по батькові)

Спеціальність 123 "Комп'ютерна інженерія"

Освітня програма «Комп'ютерна інженерія»

Керівник дипломного проекту (роботи) Кривченко Анастасія Анатоліївна

(прізвище, ім'я та по батькові)

Тема дипломного проекту (роботи) Аналіз ефективності сучасних потокових криптоалгоритмів

Обсяг розрахунково-пояснювальної записки 80 сторінок

Обсяг графічної (презентаційної) частини 19 аркушів (слайдів)

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) заключення про ступінь відповідності виконаного дипломного проекту (роботи) завданню

Представлена на рецензію кваліфікаційна робота бакалавра повністю відповідає меті проектування та технічному завданню. Тематика кваліфікаційної роботи є актуальною для своєї галузі та присвячена дослідженню ефективності і швидкодії сучасних потокових криптоалгоритмів, а також розробці моделі кодеку на базі алгоритму потокового шифрування ChaCha.

б) характеристика виконання кожного розділу дипломного проекту (роботи)

Кваліфікаційна робота складається зі вступу, двох розділів, висновків, переліку використаних джерел. У основному розділі розглянуті питання дослідження ефективності і швидкодії сучасних потокових криптоалгоритмів, реалізації алгоритмів та моделювання потокового шифрування

в) оцінка якості виконання пояснювальної записки та графічної частини дипломного проекту (роботи) Графічна частина виконана на достатньо високому рівні у вигляді презентації із використанням офісного пакету Microsoft PowerPoint та Visio. Пояснювальна записка виконана охайно та у відповідності до норм оформлення документів із використанням офісного пакету Microsoft Word. Загальна якість виконання документації – добра, академічного плагіату у роботі не виявлено

г) перелік позитивних якостей дипломного проекту (роботи)

1. Детально описано мету та цілі аналізу;
2. Реалізовано модель кодеку на базі алгоритму потокового шифрування ChaCha у візуальному режимі;
3. Досліджено ефективність і швидкодію сучасних потокових криптоалгоритмів

д) основні недоліки дипломного проекту (роботи)

1. Варто було б зберігати або передавати ключ та вектор ініціалізації із застосуванням додаткових методів захисту;
2. Швидкість потокових криптоалгоритмів варто було б оцінювати на більш потужному процесорі.

Оцінка розрахункової частини Відмінно

Оцінка графічної частини Добре

Загальна оцінка Відмінно

Прізвище, ім'я, по батькові рецензента Василіу Євген Вікторович

Місце роботи і посада рецензента Державний університет інтелектуальних технологій і зв'язку, д.т.н., проф. кафедри КБ та ТЗІ

Підпис:

« » червня 2024 р.

ВСП «Одеський технічний фаховий коледж ОНТУ»

# ВІДГУК

керівника про кваліфікаційну роботу бакалавра

*Масалитіна Руслана Вікторовича*

(прізвище, ім'я та по батькові)

Спеціальність *123 "Комп'ютерна інженерія"*

Тема кваліфікаційної роботи *Аналіз ефективності сучасних потокових криптоалгоритмів*

## ХАРАКТЕРИСТИКА КВАЛІФІКАЦІЙНОЇ РОБОТИ

а) Обсяг і якість виконання роботи (графічного матеріалу і розрахунково-пояснювальної записки) *Випускна робота виконана відповідно технічному завданню. Пояснювальна записка до випускної роботи містить 80 сторінок. У пояснювальній записці наведено етапи дослідження ефективності і швидкодії сучасних потокових криптоалгоритмів, а також розробки моделі кодеку на базі алгоритму потокового шифрування ChaCha. Графічна частина складається з 19 слайдів, оформлених у вигляді презентації, передбачених технічним завданням. Якість виконання пояснювальної записки та слайдів добра, розробку виконано у повному обсязі.*

б) Самостійність роботи

*Протягом виконання випускної бакалаврської роботи Масалитін Руслан поступово та послідовно виконував всі етапи, проявив ініціативу у створенні загальної концепції та реалізації випускної роботи. Всі роботи він виконував самостійно, з оглядом на рекомендації керівника.*

в) Теоретична підготовка здобувача освіти

*Масалитін Руслан під час роботи над випускною бакалаврською роботою вивчив і опрацював достатню кількість літературних джерел за даною тематикою.*

*Вважаю, що теоретична підготовка здобувача освіти достатня і він готовий до захисту роботи.*

г) Вміння розв`язувати виробничі і конструкторські питання на базі останніх досліджень науки і техніки, передових методів виробництва

*Під час виконання роботи Масалитін Руслан мав змогу самостійно приймати окремі рішення з виконання програмної частини роботи та показав вміння організовано працювати над поставленою задачею, складати та оформлювати презентацію проекту, користуючись сучасними комп'ютерними програмними засобами, такими як QT, Microsoft Visual Studio, Microsoft PowerPoint, Microsoft Visio.*

Оцінка розрахункової частини *Відмінно*

Оцінка графічної частини *Добре*

Загальна оцінка *Відмінно*

Прізвище, ім`я, по батькові *Кривченко Анастасія Анатоліївна*

Місце роботи і посада керівника роботи *ВСП "Одеський технічний фаховий коледж ОНТУ", викладач спецдисциплін комісії комп'ютерних технологій та програмної інженерії, голова обласної методичної комісії викладачів комп'ютерної інженерії*

Підпис

«13» 06 2024 р.

UNICHECK
by Turnitin

Ім'я користувача:
**Катерина Григорівна Краснокутська**

ID перевірки:
**1016225035**

Дата перевірки:
**02.05.2024 21:01:35 EEST**

Тип перевірки:
**Doc vs Internet + Library**

Дата звіту:
**02.05.2024 21:01:53 EEST**

ID користувача:
**100011688**

Назва документа: **2БКС-28_Руслан_Масалитін**

Кількість сторінок: **60** Кількість слів: **11925** Кількість символів: **86129** Розмір файлу: **2.74 MB** ID файлу: **1016002474**

**Виявлено модифікації тексту (можуть впливати на відсоток схожості)**

## 6.3%
## Схожість

Найбільша схожість: **1.43%** з Інтернет-джерелом (https://card-file.ontu.edu.ua/server/api/core/bitstreams/da841ffe-a0b..

6.3% **Джерела з Інтернету** 329 ..... Сторінка 62

**Не знайдено джерел з Бібліотеки**

## 0% Цитат

**Вилучення цитат вимкнене**

**Вилучення списку бібліографічних посилань вимкнене**

## 0%
## Вилучень

**Немає вилучених джерел**

## Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

| Замінені символи | 36 |
|---|---|

| Підозріле форматування | 10 сторінок |
|---|---|

**ДОЗВІЛ**
**НА РОЗМІЩЕННЯ**
**ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ**
**В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»**

Ми, що нижче підписалися,

***Масалитін Руслан Вікторович,***
здобувач освіти гр. 2БКС-28, та

***Кривченко Анастасія Анатоліївна***,
керівник випускної кваліфікаційної роботи,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до випускної кваліфікаційної роботи бакалавра на тему:

***«Аналіз ефективності сучасних потокових криптоалгоритмів» (автор роботи – Масалитін Р.В., керівник роботи – Кривченко А.А.)***

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2024 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи і даємо згоду на обробку персональних даних.

Виконавець _____ / Масалитін Р.В. /

Керівник _____ / Кривченко А.А. /

«13» червня 2024 р.