НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
Навчально-науковий фізико-технічний інститут
Кафедра математичних методів захисту інформації

«На правах рукопису»
УДК 004.056.5

«До захисту допущено»
В.о. завідувача кафедри
_____ Сергій ЯКОВЛЄВ
«__» _____ 2023 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою
«Математичні методи криптографічного захисту інформації»

зі спеціальності: 113 Прикладна математика
на тему: **«Дослідження статистичних властивостей потокового алгоритму шифрування Kriр»**

Виконав:
студент IV курсу, групи ФІ-94
Зацаренко Анастасія Юріївна _____

Керівник:
д.т.н., проф. кафедри ММЗІ
Ковальчук Людмила Василівна _____

Рецензент:
посада, степінь, звання
Прізвище Ім'я По-батькові _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

Київ — 2023

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
Навчально-науковий фізико-технічний інститут
Кафедра математичних методів захисту інформації

Рівень вищої освіти — перший (бакалаврський)
Спеціальність — 113 Прикладна математика,
ОПП «Математичні методи криптографічного захисту інформації»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Сергій ЯКОВЛЄВ
«__» _____ 2023 р.

ЗАВДАННЯ
на дипломну роботу

Студент: Зацаренко Анастасія Юріївна

1. Тема роботи: «*Дослідження статистичних властивостей потокового алгоритму шифрування Krip*», науковий керівник роботи: д.т.н., проф. кафедри ММЗІ Ковальчук Людмила Василівна,

затверджені наказом по університету №__ від «__» _____ 2023 р.

2. Термін подання студентом роботи: «__» _____ 2023 р.

3. Об'єкт дослідження: *процес шифрування повідомлень з використанням потокових алгоритмів шифрування*

4. Предмет дослідження: *аналіз статистичних властивостей вихідних послідовностей потокового алгоритму шифрування Krip*

5. Перелік завдань:

– *вибрати тести для тестування алгоритму Krip та визначитися з методикою тестування;*

– *написати програму, яка реалізовує алгоритм шифрування Krip та тести, необхідні для проведення статистичних досліджень;*

– *отримати результат тестування;*

– *зробити висновки щодо криптографічної якості вихідних послідовностей з викоританням методити з NIST на базі отриманих*

результатів тестування.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: *презентація доповіді*

7. Орієнтовний перелік публікацій: *планується доповідь на всеукраїнській конференції*

8. Дата видачі завдання: 10 вересня 2022 р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання	Примітка
1	Узгодження теми роботи із науковим керівником	01-15 вересня 2022 р.	Виконано
2	Огляд опублікованих джерел за тематикою дослідження	Вересень-жовтень 2022 р.	Виконано
3	Вибір тести для проведення статистичного дослідження та методики тестування	Листопад-грудень 2022 р.	Виконано
4	Написання програмної реалізації алгоритму Krip та обраних тестів	Січень-березень 2023 р.	Виконано
5	Проведення тестування та аналіз отриманих результатів	Квітень-травень 2023 р.	Виконано
6	Оформлення дипломної роботи	Червень 2023 р.	Виконано

Студент _____ Анастасія ЗАЦАРЕНКО

Керівник _____ Людмила КОВАЛЬЧУК

РЕФЕРАТ

Кваліфікаційна робота містить: 61 стор., 1 рисунок, 5 таблиць, 10 джерел.

Темою роботи є дослідження статистичних властивостей потокового алгоритму шифрування Krip.

Метою дослідження є статистичний аналіз криптографічних властивостей вихідної гами потокового алгоритму шифрування Krip. Для досягнення мети необхідно розв'язати задачу дослідження, яка полягає у тестуванні достатньої кількості шифртекстів, отриманих з використанням алгоритму Krip та аналізу їх статистичних властивостей з використанням шести тестів і методики тестування NIST.

Об'єктом дослідження є процес шифрування повідомлень з використанням потокових алгоритмів шифрування.

Предметом дослідження є аналіз статистичних властивостей вихідних послідовностей потокового алгоритму шифрування Krip.

У цій роботі проведено дослідження криптографічної якості вихідних послідовностей потокового алгоритму Krip за допомогою шести статистичних тестів та методики тестування NIST SP 800-22. В результаті аналізу отриманих результатів зроблено висновки, що алгоритм Krip можна вважати генератором псевдовипадкових чисел.

ПОТОКОВИЙ АЛГОРИТМ ШИФРУВАННЯ, МАЛОРЕСУРСНА КРИПТОГРАФІЯ, ШИФР KRIP, СТАТИСТИЧНЕ ДОСЛІДЖЕННЯ

ABSTRACT

The thesis contains: 61 pages, 1 figures, 5 tables, 10 sourcesıo

The theme of this thesis is investigation of statistical properties of stream encryption algorithm Krip.

The aim of research is a statistical analysis of the cryptographic properties of the output gamma of the Krip stream encryption algorithm. To achieve the aim, it is necessary to solve the research problem, which consists in testing a sufficient number of ciphertexts obtained using the Krip algorithm and analyzing their statistical properties using six tests and the NIST testing methodology.

The object of research is the process of message encryption using stream encryption algorithms.

The subject of research is the analysis of statistical properties of the output sequences of the Krip stream encryption algorithm.

In this thesis the cryptographic quality of the output sequences of the Krip streaming algorithm was investigated using six statistical tests and the NIST SP 800-22 testing methodology. As a result of the analysis of the obtained results, it was concluded that the Krip algorithm can be considered a generator of pseudo-random numbers.

STREAM ENCRYPTION ALGORITHM, LIGHTWEIGHT CRYPTOGRAPHY, KRIP CIPHER, STATISTICAL INVESTIGATION

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

NIST – Національний інститут стандартів і технології

ВСТУП

Актуальність дослідження. У сучасному світі спостерігається суттєвий розвиток інформаційних технологій, а разом з ним виникає потреба у проєктуванні ефективних та безпечних систем шифрування. Рішенням цієї проблеми може стати малоресурсна криптографія, яка є компромісом між швидкістю та стійкістю алгоритмів.

Попри розмаїття існуючих алгоритмів потокового шифрування, створення апаратно реалізованих шифрів, що відповідають підвищеним вимогами до швидкодії, досі є актуальним завданням. Тому подальша робота буде сконцентрована на дослідженні алгоритму потокового шифрування Krip, що є одним з можливих підходів до розв'язання цієї задачі.

Метою дослідження є статистичний аналіз криптографічних властивостей вихідної гами потокового алгоритму шифрування Krip. Для досягнення мети необхідно розв'язати **задачу дослідження**, яка полягає у тестуванні достатньої кількості шифртекстів, отриманих з викоританням алгоритму Krip та аналізу їх статистичних властивостей з використанням шести тестів опублікованих в [1] і методики тестування NIST. Для розв'язання задачі необхідно вирішити такі завдання:

1) провести огляд опублікованих джерел за тематикою дослідження;

2) вибрати тести для тестування алгоритму Krip та визначитися з методикою тестування;

3) написати програму, яка реалізовує алгоритм шифрування Krip та тести, необхідні для проведення статистичних досліджень;

4) отримати результат тестування;

5) зробити висновки щодо криптографічної якості вихідних послідовностей з викоританням методити з NIST на базі отриманих результатів тестування.

Об'єктом дослідження є процес шифрування повідомлень з

використанням потокових алгоритмів шифрування.

Предметом дослідження є аналіз статистичних властивостей вихідних послідовностей потокового алгоритму шифрування Krip.

При розв'язанні поставлених завдань використовувались такі *методи дослідження*: методи криптографії, теорії булевих функцій, математичної статистики. Для програмної реалізації було використано мову програмування *C#*.

Наукова новизна отриманих результатів полягає у тому, що вперше було проведено статистичне дослідження алгоритму Krip тестами, відмінними від тестів пакету NIST.

Практичне значення результатів полягає у тому, що Krip – це новий алгоритм, і потребує більш глибоких статистичних досліджень, оскільки на сьогодні дослідження були зроблені лише для тестів пакету NIST.

1 ЗАГАЛЬНІ ТЕОРЕТИЧНІ ВІДОМОСТІ

У цьому розділі наведені основні теоретичні поняття, необхідні для подальшого дослідження теми. Також розглядаються переваги створення малоресурсних алгоритмів потокового шифрування.

1.1 Симетрична криптографія

Шифрування з симетричним ключем – це схема шифрування з використанням єдиного ключа для функції шифрування та розшифрування. Це відрізняє цей вид шифрування від асиметричного або шифрування з відкритим ключем, де застосовуються два різні ключі, які математично пов'язані.

Одним із недоліків систем шифрування з відкрити ключем є те, що для їх роботи потрібна відносно складна математика, що робить їх обчислювально складними та відносно повільними. Симетричне шифрування натомість вимагає лише одного ключа відносно короткої довжини, що дозволяє ефективно обробляти великі обсяги даних за менший час. Разом з тим, алгоритми з симетричним ключем є надійними, оскільки вони досі використовуються як стандартний метод шифрування компаніями та державними органами в усьому світі. Однак, складність використання цих шифрів полягає у необхідності ділитися ключем з іншими, так щоб його не отримав зловмисник.

У сучасних шифрах в якості вхідних даних приймається бітові рядки. Алгоритми симетричного шифрування поділяються на блокові та потокові шифри.

1.1.1 Блокові шифри

В блокових шифрах відкритий текст розбивається на блоки фіксованої довжини, а потім алгоритм перетворює кожен блок у шифртекст.

Означення 1.1. Блоковий шифр - це шифруюче відображення виду

$$E_k : V_n \times \mathcal{K} \to V_n,$$

де $V_n = \{0,1\}^n$ - множина текстів, $\mathcal{K} \equiv V_m = \{0,1\}^m$ - множина ключів.

Для створення безпечної системи шифрування необхідно вжити заходів, як-от розсіювання та перемішування, які були визначені Клодом Шенноном у 1945 році [2].

Механізм шифрування передбачає залежне від ключа перетворення блоку відкритого тексту в блок шифртексту. Принцип розсіювання спрямований на те, щоб відносини між відкритим і зашифрованим текстами були складені найбільш складним чином. Тобто очікується, що всі статистики у шифртексті повинні бути близькі до рівноймовірних. З іншого боку, принцип перемішування полягає у тому, щоб взаємозв'язок між статистикою зашифрованого тексту та значенням ключа шифрування був максимально заплутаним. Таким чином, перешкодивши спробам зловмисника виявити ключ.

1.1.2 Потокові шифри

В потокових шифрах дані обробляються посимвольно, незалежно один від одного.

Означення 1.2. Потоковий шифр - це криптосистема, яка складається з двох частин :

- генератора гами, що є математичним об'єктом, який визначається

відображеннями $Init$ (початкова ініціалізація), $Stream$ (вироблення символу гами) та $Next$ (оновлення стану шифрування);

• функції виходу Out, яка змішує символ відкритого тексту із символом гами та додатковою інформацією для одержання символу шифротексту. (Out^{-1} - обернена функція, яка використовується для розшифрування).

Генератор ключів потокового шифру породжує потік бітів ключа K, які будуть використовуватися як гама z_i, базуючись на внутрішньому стані S. Таким чином ці шифри поділяються на синхронні та самосинхронізовані.

Генератор гами із самосинхронізацією використовує l попередніх символів шифротексту для обчислення потоку ключа. Описується наступним чином :

1) $Init(K, IV) \to S, C_0, C_{-1}, C_{-2}, \ldots, C_{-l+1}$

2) $z_i = Stream(S, K, C_{i-1}, C_{i-2}, \ldots, C_{i-l})$

3) $C_i = Out(x_i, z_i, R_i)$

Функція $Next$ не використовується, оскільки немає внутрішнього стану, який оновлюється з кожним символом. Тому в цьому режимі, якщо у шифротексті відбулися певні збурення, то зазнають спотворення не більше ніж l символів відкритого тексту.

У синхронних шифрах оновлення внутрішнього стану та генерування потоку ключа відбувається незалежно від відкритого та зашифрованого текстів. Отже, синхронний генератор гами описується наступними функціями :

1) $S_0 = Init(K, IV)$

2) $z_i = Stream(K, S_i)$

3) $S_{i+1} = Next(K, S_i)$

На відміну від генератора із самосинхронізацією, синхронний потоковий шифр не має ефекту поширення помилок. Тобто спотворення одного символу шифротексту призведе до зміни лише цього біту, не впливаючи на інші елементи відкритого тексту.

1.2 Переваги потокових алгоритмів шифрування

На сьогодні немає чіткої думки, який алгоритм краще : потоковий чи блоковий. Кожен з них має свою сферу застосування, а також має переваги і недоліки. Зосередимося саме на перевагах використання потокових шифрів.

1. Швидкість шифрування очолює список переваг потокових алгоритмів, які перетворюють відкритий текст по бітах, а блочні шифри - по частинах фіксованої довжини. Це робить блокові алгоритми повільнішими, оскільки перед обробкою даних потрібно накопичити весь блок.

2. Друга перевага полягає в тому, що потокові алгоритми вимагають менше ресурсів для реалізації. Крім того, з цієї причини апаратна складність потокового шифру досить низька, що дозволяє легко використовувати ці алгоритми в сучасних програмах, і розробникам для цього не потрібне складне обладнання.

3. На відміну від блокових шифрів, які працюють з великими фрагментами тексту, потокові алгоритми обробляють дані посимвольно, тому вони не потребують значного обсягу пам'яті.

4. Зворотність – це ще одна перевага потокових алгоритмів, які використовують операцію побітового додавання для шифрування. У випадку блокових алгоритмів розшифрування є порівняно складною операцією через використання більшої кількості бітів.

5. Синхронні потокові алгоритми мають значну перевагу над блоковими шифрами, коли є вірогідність виникнення помилок передачі. Тому що спотворення одного біта шифротексту призведе до зміни лише цього символу у відкритому тексті, на відміну від блокових, де помилка може розповсюдитися.

1.3 Малоресурсна криптографія

У сучасному світі спостерігається суттєвий розвиток Інтернет речей, смартфонів та інших девайсів, а разом з тим підвищуються вимоги до їх безпеки. У 2018 році NIST розрочав роботу над процесом стандартизації малоресурсної криптографії. Метою даного проєкту було просування криптографічних алгоритмів, які підходять для використання в обмеженому середовищі [3].

Малоресурсна криптографія – це розділ криптографії, метою якого є створення алгоритмів, адаптованих для пристроїв з обмеженими ресурсами. Тобто ідея полягає у реалізації систем з кращим балансом між швидкістю, продуктивністю та безпекою.

Розгялнемо детальніше вимоги до малоресурсних алгоритмів шифрування.

1) Насамперед це ефективність, оскільки ці алгоритми мають швидко виконувати всі необхідні операції з мінімальним викоританням обчислювальних ресурсів.

2) Для забезпечення ефективної роботи необхідним є використання ключа меншої довжини, оскільки довші послідовності сповільнюють роботу всього алгоритму.

3) Ці алгоритми мають мінімально використовувати пам'ять пристрою, яка є одним з критичних ресурсів.

4) Раціональне використання ресурсів пристрою допомагає споживати менше енергії і подовжити час роботи заряду, тому наступна вимога полягає у економному енергоспоживанні.

5) Незважаючи на попередні критерії, які роблять ці алгоритми більш вразливими до атак, однієї з найважливіших вимог залишається забезпечення достатнього рівня безпеки.

Отже, головною перевагою малоресурсної криптографії є те, що при використанні менших обчислювальних ресурсів та меншого обсягу пам'яті,

ці алгоритми залишаються практично стійкими. Тому вони підходять для захисту бездротових сенсорних мереж, пристроїв Інтернету речей, смарт-карток та інших вбудованих систем.

За останнє десятиліття було запропоновано низку малоресурсних криптографічних примітивів, як-от геш-функції, блокові та потокові шифри, які пропонують перевагу в продуктивності порівняно з традиційними криптографічними стандартами [4]. Але важливо відзначити, що дана галузь продовжує розвиватися для знаходження більш ефективних та надійних алгоритмів.

Висновки до розділу 1

У цьому розділі були розглянуті основні теоретичні поняття симетричної криптографії, які знадобляться для подальшого досліження теми. Проаналізувавши різні види алгоритмів симетричного шифрування, були виділені преваги використання потокових шифрів. Також розглянута концепція малоресурсної критопграфії, виникнення якої пов'язано з розвитком пристроїв з обмеженою кількістю обчислювальних ресурсів.

2 ОГЛЯД ДЕЯКИХ МАЛОРЕСУРСНИХ ПОТОКОВИХ АЛГОРИТМІВ ШИФРУВАННЯ

У цьому розділі розглянуто загальний опис деяких сучасних малоресурсних потокових алгоритмів шифрування, як-от: Espresso, Strumok і Krip, та наведено порівняння їх характеристик. Також описано методику тестування, за допомогою якої в наступному розділі буде проведено дослідження статистичних властивостей шифру Krip.

2.1 Огляд сучасних малоресурсних потокових шифрів

На сьогоднішний день галузь малоресурсної криптографії має великий попит через розвиток пристроїв Інтернету речей та інших вбудованих програм. Попри розмаїття існуючих малоресурсних криптографічних примітивів, досі є потреба у створенні алгоритмів, придатних для використання в системах з обмеженими обчислювальними ресурсами. Одним із підходів до забезпечення конфіденційності даних у таких системах є потокові алгоритми. Розглянемо структуру деяких малоресурсних потокових шифрів, як-от Espresso, Strumok та новий алгоритм Krip.

2.1.1 Алгоритм Espresso

Одним з прикладів хорошого малорерсного потокового алгоритму є шифр Espresso. Він був розроблений для системи мобільного зв'язку п'ятого покоління [5]. Двома основними складовими цього агоритму є 256-бітний регістр зсуву з нелінійним зворотнім зв'язком та функція нелінійного виводу з 20 змінними. Генератор гами в шифрі Espresso використовує 96-бітний вектор ініціалізації та 128-бітний секретний ключ.

При цьому довжина слова цього алгоритму дорівнює 1 біту.

Позначимо функцію зворотного зв'язку нелінійного регістру зсуву в такті i як $g_i, i = \overline{0,255}$, тоді вона визначаються наступним чином:

$$
\begin{aligned}
g_{255}(x) &= x_0 \oplus x_{41}x_{70} \\
g_{251}(x) &= x_{252} \oplus x_{42}x_{83} \oplus x_8 \\
g_{247}(x) &= x_{248} \oplus x_{44}x_{102} \oplus x_{40} \\
g_{243}(x) &= x_{244} \oplus x_{43}x_{118} \oplus x_{103} \\
g_{239}(x) &= x_{240} \oplus x_{46}x_{141} \oplus x_{117} \\
g_{235}(x) &= x_{236} \oplus x_{67}x_{90}x_{110}x_{137} \\
g_{231}(x) &= x_{232} \oplus x_{50}x_{159} \oplus x_{189} \\
g_{217}(x) &= x_{218} \oplus x_3x_{32} \\
g_{213}(x) &= x_{214} \oplus x_4x_{45} \\
g_{209}(x) &= x_{210} \oplus x_6x_{64} \\
g_{205}(x) &= x_{206} \oplus x_5x_{80} \\
g_{201}(x) &= x_{202} \oplus x_8x_{103} \\
g_{197}(x) &= x_{198} \oplus x_{29}x_{52}x_{72}x_{99} \\
g_{193}(x) &= x_{194} \oplus x_{12}x_{121}.
\end{aligned}
$$

Для усіх інших тактів $g_i(x) = x_{i+1}$. Вихідна функція $z(x)$ визначається наступним чином:

$$
\begin{aligned}
z(x) = x_{80} \oplus x_{99} \oplus x_{137} \oplus x_{227} \oplus x_{222} \oplus x_{187} \oplus x_{243}x_{217} \\
\oplus x_{247}x_{231} \oplus x_{213}x_{235} \oplus x_{255}x_{251} \oplus x_{181}x_{239} \\
\oplus x_{174}x_{44} \oplus x_{164}x_{29} \oplus x_{255}x_{247}x_{243}x_{213}x_{181}x_{174}.
\end{aligned} \tag{2.1}
$$

2.1.2 Алгоритм Strumok

Одним із прикладів сучасних алгоритмів потокового шифрування є шифр Strumok, який є національним стандартом симетричного шифрування в Україні. Швидкість формування потоку ключа генератора Strumok може сягати понад 10 Гбіт/с, і це перевищує більшість відомих алгоритмів [6].

Генератор потоку ключа в шифрі Strumok використовує 256-бітний вектор ініціалізації та 256-бітний або 512-бітний секретний ключ K. Оскільки цей алгоритм орієнтований на 64-розрядні обчислювальні системи, тому розмір слова дорівнює 64 бітам. Для генерування гами використовується вісімнадцять 64-бітових блоків, які формують регістр зсуву з лінійним зворотнім зв'язком $s_i = (s_{15}^{(i)}, s_{14}^{(i}), \ldots, s_0^{(i)})$ та регістр скінченного автомату $r_i = (r_1^{(i)}, r_0^{(i)})$.

Схему ключового потоку алгоритму Strumok наведено на рисунку 2.1.

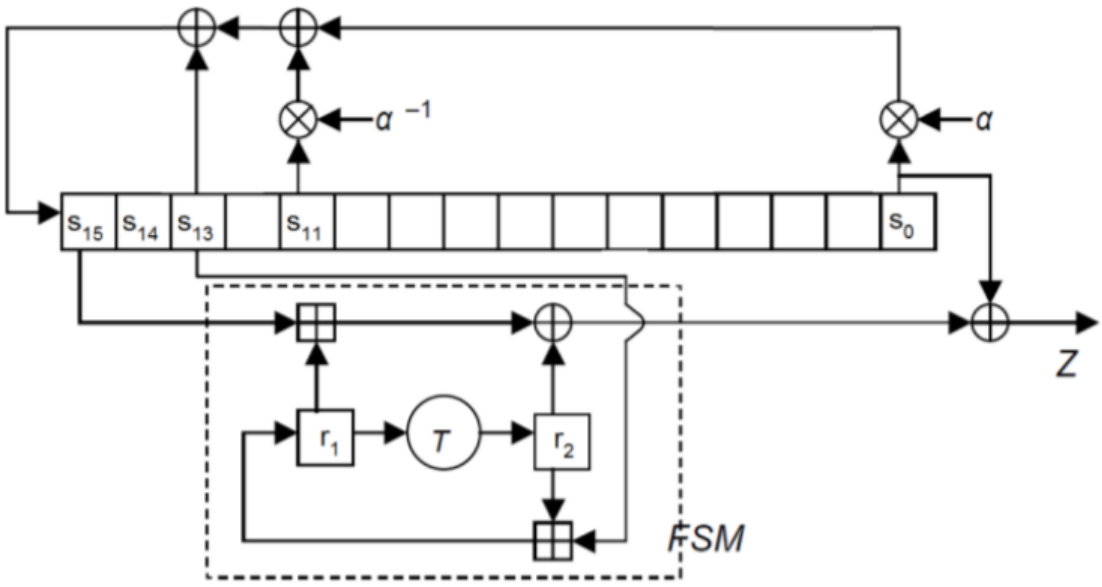

Рисунок 2.1 – Формування потоку ключа шифру Strumok

Функція виходу для алгоритму Strumok будується за примітивним

поліномом над полем $GF(2^{64})$

$$f(x) = x^{16} + x^{13} + \alpha^{-1}x^{11} + \alpha, \tag{2.2}$$

де α є коренем примітивного полінома над полем $GF(2^8)$

$$g(z) = z^8 + \beta^{170}z^7 + \beta^{166}z^6 + \beta^2z^5 + \beta^{224}z^4 + \beta^{70}z^3 + \beta^2. \tag{2.3}$$

2.1.3 Алгоритм Krip

Ще одним прикладом малоресурсного потокового алгоритму шифрування є новий шифр Krip, який відповідає сучасним вимогам до швидкодії та гарантує стійкість до відомих криптографічних атак [7].

Цей алгоритм побудований на основі нелінійного регістру зсуву над алфавітом X потужності 2^{256}. Однією з основних процедур алгоритму є формування 512-бітного початкового стану нелінійного регістру зсуву. Для його генерування на вхід подаються 256-бітний ключ та 128-бітний вектор ініціалізації. Наступні стани обчислюються за допомогою рекурентного співвідношення $(s_{i+2}, s_{i+1}) = h((s_{i+1}, s_i), x_i)$, h – функція переходу, яка обчислюється як

$$h((s_{i+1}, s_i), x_i) = (s_i \oplus \phi(x_i \oplus s_{i+1}), s_{i+1}), \tag{2.4}$$

де $x \epsilon X, s_i, s_{i+1} \epsilon V_{256}, \phi$ -– підстановки на множині V_{256}.

Шифрування та розшифрування повідомлень виконуються за допомогою функції виходів, яка обчислюється за формулою:

$$f((s_{i+1}, s_i), x_i) = x_i \oplus \psi_1(s_{i+1}, s_i) \oplus \psi_2(s_{i+1}, s_i), \tag{2.5}$$

де $x \epsilon X, s_1, s_2 \epsilon V_{256}, \psi = (\psi_1, \psi_2)$ -– підстановка, де $\psi_i : V_{512} \to V_{256}, i = 1,2$.

Варто зазначити, що підстановки ϕ та ψ співпадають з раундовими функціями шифру Kalyna [8].

2.2 Порівняльний аналіз алгоритмів

У табилці 2.1 наведені основні характеристики алгоритмів Krip, Espresso та Strumok, де дані щодо швидкості шифрування взяті з [7].

Таблиця 2.1 – Характеристика даних алгоритмів

Характеристика	Алгоритм		
	Krip	Espresso	Strumok
Розмір стану, біт	512	256	1024
Розмір ключа, біт	256	128	256 / 512
Розмір вектора ініціалізації	128	96	256 / 512
Довжина слова, біт	256	1	64
Швидкість шифрування, Гбіт/с	44.8	2.22	11.2

Порівнюючи алгоритми Espresso та Strumok з шифром Krip, можна зробити висновки, що :

1) Espresso потребує менший об'єм пам'яті, але виконує шифрування повідомлень по одному біту, що значно відображається на швидкості його роботи. Це дає алгоритму Krip, який за один крок шифрує слова довжиною 256 біт, двадцятикратну перевагу у швидкодії.

2) Krip використовує вдвічі менший розмір стану, ніж алгоритм Strumok, але при цьому шифрує вчетверо довші слова. Це дає алгоритму Krip чотирикратну перевагу у швидкодії.

Але варто відмітити, що шифр Krip має значно більші апаратні затрати, оскільки містить 256-бітну та 512-бітну раундові функції з шифру Kalyna. Тимчасом як в алгоритмі Strumok використовується 64-розрядна раундова функція, а в Espresso взагалі не реалізовано табличних елементів.

2.3 Методика тестування послідовностей

Безпека будь-якої криптосистеми залежить від якості її генератора. Тому однією з необхідних умов є випадковість та рівномірність символів вихідних послідовностей алгоритмів.

Для статистичного дослідження властивостей вихідних повідомлень шифру Krip було обрано методологію тестування NIST SP 800-22 [9]. Вона містить набір з 15 статистичних тестів для оцінки якості генераторів псевдовипадкових чисел. Але для дослідження алгоритму Krip буде використаний інший набір тестів, опублікований в [1].

Розглянемо один зі способів доведення незалежності набору тестів [10]. Нехай перевіряємо набір з N тестів при однаковому рівні значимості α. Використовуючи генератор псевдовипадкових послідовностей генеруємо n послідовностей. Рівень значимості для тесту перевірки незалежності дорівнює β. Обчислюємо наступні параметри :

$$a = (1 - \alpha)^N \tag{2.6}$$

$$\sigma^2 = \frac{a(1-a)}{n} \tag{2.7}$$

Для кожного тесту при заданому рівні значимості α важливо визначити максимально допустиме значення, за яким буде оцінюватися чи пройшла довільна послідовність цей тест. Шляхом проведення практичних експериментів обчислюємо значення :

$$\eta = \frac{1}{n} \sum_{j=1}^{n} \eta_j, \tag{2.8}$$

де η_j – індикатор того, чи пройшла j-та послідовність набір з N тестів. Далі обчислюємо функцію помилом $erfc\big(\frac{|\eta - a|}{\sigma\sqrt{2}}\big)$ і якщо це значення не менше за β, то даний набір тестів вважається незалежним.

З умови незалежності тестів, якщо для кожного тесту помилка першого роду дорівнює α, то можна знайти імовірність того, що

випадкова послідовність не пройде щонайменше один тест з набору. Ця імовірність є загальною помилкою першого роду і обчислюється за формулою :

$$A = 1 - (1 - \alpha)^N = 1 - a \tag{2.9}$$

Останнім етапом даної методики є оцінка криптографічної якості вихідних послідовностей на базі отриманих результатів тестування. Нехай k послідовностей з n згенерованих пройшли всі тести. Тоді вважатимемо алгоритм є генератором псевдовипадкових послідовностей, якщо виконується нерівність :

$$1 - A - 3\sqrt{\frac{A(1-A)}{N}} \leqslant \frac{k}{N} \leqslant 1 - A + 3\sqrt{\frac{A(1-A)}{N}}, \tag{2.10}$$

Висновки до розділу 2

У цьому розділі були розглянуті основні властивості сучасних малоресурсних потокових шифрів на приклаладі шифрів Espresso, Strumok і Krip. Також, шляхом порівняння їх характеристик, були виділені переваги та недоліки алгоритму Krip. Окрім цього, було обрано статистичні тести для тестування якостей вихідних послідовностей потокового шифру Krip. А також обрано і описано методику тестування для подальшого дослідження цього алгоритму.

3 ДОСЛІДЖЕННЯ ВИХІДНИХ ПОСЛІДОВНОСТЕЙ АЛГОРИТМУ KRIP СТАТИСТИЧНИМИ КРИТЕРІЯМИ

У цьому розділі наведений опис шести статистичних критеріїв, призначених для тестування коротких послідовностей, та доведена їх незалежність. Також проведено статистичне дослідження вихідних послідовностей алгоритму Krip за допомогою цих тестів та проаналізовано отримані результати.

3.1 Опис статистичних критеріїв

Визначимо шість критеріїв, які можна застосовувати для дослідження статистичних властивостей коротких послідовностей:

1) критерій χ^2 для знаків;

2) критерій серій максимальної довжини;

3) критерій χ^2 для біграм;

4) критерій кількості серій;

5) критерій місць знаків.

6) критерій інверсій;

Нехай $\{\xi_i\}_{i=1}^n$ – послідовність незалежних випадкових величин, рівноймовірно розподілених на алфавіті A об'єму N. Далі наведемо матиматичне обґрунтування роботи даних критеріїв.

3.1.1 Критерій χ^2

Випадкова величина

$$\xi^2 = \sum_{i=1}^{N} \frac{(M_i - np_i)^2}{np_i} = \sum_{i=1}^{N} \frac{M_i^2}{np_i} - n, \tag{3.1}$$

де M_i – кількість символів $a_i(a_i \epsilon A)$ у послідовності $\{\xi_i\}_{i=1}^n$.

Якщо $p_i = \frac{1}{N}$, тобто послідовність $\{\xi_i\}_{i=1}^n$ має рівномірний розподіл, то випадкова величина дорівнюватиме

$$\xi^2 = \frac{N}{n} \sum_{i=1}^{N} M_i^2 - n, \tag{3.2}$$

Отже, ξ^2 буде асимптотично мати χ^2-розподіл з $N-1$ степенем свободи.

Якщо $N - 1 \leqslant 30$, то граничне значення статистики χ_α^2 для заданого рівня значимості α знаходять з формули $F(\chi_\alpha^2) = 1 - \alpha$, де $F()$ – функція χ-розподілу з $N-1$ степенем свободи.

Якщо $N - 1 > 30$, то розподіл $\sqrt{2\chi^2}$ наближається до нормального $N(x; \sqrt{2(N-1)+1}$,1), і граничне значення статистики χ_α^2 для заданого рівня значимості α знаходять з формули $\chi_\alpha^2 = \frac{1}{2}(\sqrt{2(N-1)-1} + z_\alpha)^2$, де $\Phi(z_\alpha) = 1 - \alpha$, $\Phi(x)$ – функція стандартного нормального розподілу.

Вважається, що при рівні значимості α послідовність пройшла тест, якщо $\chi^2 < \chi_\alpha^2$. Д

3.1.2 Критерій серій максимальної довжини

Нехай маємо послідовність, тоді серією називатимемо довільну підпослідовність, що складається з будь-яких однакових елементів алфавіту . В такому разі імовірність того, що на фіксованому місці у цій послідовності починається серія довжини не менше, ніж S, дорівнює $\frac{1}{N^S} N = N^{1-S}$.

Нехай подія $U(i, S)$ полягає у тому, що з i-го місця починається серія довжини $\geqslant S$. Тоді імовірність виникнення в послідовності серії довжиною не менше, ніж S, дорівнює

$$P(\cup_{i=1}^{n-S+1} U(i,S)) \leqslant \sum_{i=1}^{n-S+1} N^{1-S} = (n - S + 1)N^{1-S}. \tag{3.3}$$

Так як при натуральних S, n, N та $S \leqslant n$ функція $(n - S + 1)N^{1-S}$ спадає з ростом S, то для знаходження максимально допустимого значення S_α потрібно визначити значення S, для яких ця імовірність буде менше деякого заданого рівня α. Для цього розв'язуємо рівняння: $(n - S_\alpha + 1)N^{1-S_\alpha} = \alpha$. При $n > 50$ можна скористатися наближеною формулою $nN^{1-S_\alpha} = \alpha$, звідки знаходимо $S_\alpha = \frac{lg\frac{nN}{\alpha}}{lgN}$; інакше можна знайти значення S_α перебором по цілим числам до 50. Оскільки S_α повинно бути цілим числом, то можна вважати $S_\alpha = \lfloor\frac{lg\frac{nN}{\alpha}}{lgN}\rfloor$.

Вважається, що при рівні значимості α послідовність пройшла тест, якщо $S \leqslant S_\alpha$, де $S = max\{l : \xi_i = \xi_{i+1} = \cdots = \xi_{i+l-1}, 1 \leqslant i \leqslant i + l - 1 \leqslant n\}$.

3.1.3 Критерій χ^2 для біграм

Послідовність $\nu = \{\nu_j\}_{j=1}^{n/2}$, де $\nu_j = (\xi_{2j-1}, \xi_{2j})$, яка складається з біграм, що не перекриваються, буде послідовністю незалежних випадкових величин, які приймають значення в алфавіті $A \times A$ об'єму N^2. Тоді випадкова величина

$$X^2 = \sum_{i,j=1}^{N} \frac{(M_{ij} - kp_{ij})^2}{kp_{ij}} = \sum_{i,j=1}^{N} \frac{M_{ij}^2}{kp_{ij}} - k, \tag{3.4}$$

де $k = \frac{n}{2}$, M_{ij} – кількість символів $a_i a_j (a_i, a_j \epsilon A)$ у послідовності ν.

Якщо $p_{ij} = \frac{1}{N^2}$, тобто послідовність $\nu = \{\nu_j\}_{j=1}^{n/2}$ має рівномірний розподіл, то випадкова величина дорівнюватиме

$$X^2 = \frac{N^2}{k} \sum_{i,j=1}^{N} M_{ij}^2 - k, \tag{3.5}$$

Отже, X^2 буде асимптотично мати χ^2-розподіл з $N^2 - 1$ степенем свободи.

Якщо $N^2 - 1 \leqslant 30$, то граничне значення статистики X_α^2 для заданого рівня значимості α знаходять з формули $F(X_\alpha^2) = 1 - \alpha$, де $F()$ – функція

χ-розподілу з N^2-1 степенем свободи.

Якщо $N^2-1>30$, то розподіл $\sqrt{2\chi^2}$ наближається до нормального $N(x;\sqrt{2(N^2-1)+1},1)$, і граничне значення статистики X_α^2 для заданого рівня значимості α знаходять з формули $X_\alpha^2=\frac{1}{2}(\sqrt{2(N^2-1)-1}+z_\alpha)^2$, де $\Phi(z_\alpha)=1-\alpha$, $\Phi(x)$ – функція стандартного нормального розподілу.

Вважається, що при рівні значимості α послідовність пройшла тест, якщо $X^2<X_\alpha^2$.

3.1.4 Критерій кількості серій

Визначимо випадкову величину $G_n=\sum_{i=1}^{n-1}I(\xi_{i+1}\neq\xi_i)+1$, що позначає кількість серій у послідовності $\{\xi_i\}_{i=1}^n$). Тоді $MG_n=\sum_{i=1}^{n-1}MI+1=(n-1)(1-\frac{1}{N}+1)$ та $DG_n=\sum_{i=1}^{n-1}DI=(n-1)(1-\frac{N-1}{N^2})$.

Оскільки G_n є сумою однаково розподілених незалежних випадкових величин, то величина

$$G=\frac{G_n-MG_n}{\sqrt{DG_n}} \tag{3.6}$$

має асимптотично стандартний нормальний розподіл, тобто $G=\frac{G_n-MG_n}{\sqrt{DG_n}}\to N(0,1)$ при $b\to\infty$. Тому для обраного рівня значимості α: $\alpha=P(|\frac{G_n-MG_n}{\sqrt{DG_n}}|>G_\alpha)=2-2F(G_\alpha)$, тобто G_α знаходиться з рівняння $F(G_\alpha)=1-\frac{1}{\alpha}$, де $F(x)$ -– функція стандартного нормального розподілу.

Вважається, що при рівні значимості α послідовність пройшла тест, якщо $|G|<G_\alpha$.

3.1.5 Критерій місць знаків

Визначимо випадкову величину $R_\nu=\sum_{i=1}^{n-1}iI(\xi_i=\nu)$, що позначає суму місць знаку ν ($\nu\epsilon$) у послідовності $\{\xi_i\}_{i=1}^n$). Оскільки $I(\xi_i=\nu)$ -– незалежні випадкові величини, тоді $MR_\nu=\sum_{i=1}^{n}iP(\xi_i=\nu)=\frac{n(n+1)}{2N}$ та $DR_\nu=\sum_{i+1}^{n}i^2DI(\xi_i=\nu)=\frac{N-1}{N^2}\frac{n^3}{3}$.

Тоді $(\frac{R_\nu - MR_\nu}{\sqrt{DR_\nu}})^2 = \frac{(R_\nu - \frac{n(n+1)}{2N})^2}{\frac{N-1}{N^2}\frac{n^3}{3}} \approx \frac{3N}{n}(\frac{R_\nu}{n} - \frac{n+1}{2N})^2$. Оскільки величини $R_\nu = \sum_{i=1}^{n-1} iI(\xi_i = \nu)$ $(\nu \epsilon A)$ є незалежними, розподілом величини $\frac{R_\nu - MR_\nu}{\sqrt{DR_\nu}}$ для досить великих n можна вважати $N(0,1)$. Отже, можна вважати, що величина

$$T = \frac{3N}{n}\sum_{\nu=0}^{N-1}(\frac{R_\nu}{n} - \frac{n+1}{2N})^2 \tag{3.7}$$

має розподіл χ^2. Тому для обраного рівня значимості α повинно виконуватись $\alpha = P(T > \chi^2_\alpha) = 1 - F(\chi^2_\alpha)$, тобто χ^2_α знаходиться з рівняння $F(\chi^2_\alpha) = 1 - \alpha$, де $F(x)$ – функція розподілу χ^2.

Вважається, що при рівні значимості α послідовність пройшла тест, якщо $T < \chi^2_\alpha$.

3.1.6 Критерій інверсії

При $n = 2l$ побудуємо допоміжну послідовність $\sigma_i = I\{\xi_{2i-1} < \xi_{2i}\}$, $i = \overline{1,l}$. Тоді: $q = P(\sigma_i = 0) = \frac{1}{2} + \frac{1}{2N}$, $p = P(\sigma_i = 1) = \frac{1}{2} - \frac{1}{2N}$, $M\sigma_i = \frac{1}{2} - \frac{1}{2N}$, $D\sigma_i = \frac{1}{4} - \frac{1}{4N^2}, \forall i = \overline{1,l}$.

Визначимо випадкову величину $I_n = \sum_{i=1}^{l} \sigma_i$, що позначає кількість інверсій без перекриття у послідовності $\{\xi_i\}_{i=1}^n$. Тоді $MI_n = l(\frac{1}{2} - \frac{1}{2N})$ та $DI_n = l(\frac{1}{4} - \frac{1}{4N^2})$.

Оскільки I_n є сумою однаково розподілених незалежних випадкових величин, то величина

$$I = \frac{I_n - MI_n}{\sqrt{DL_n}} \tag{3.8}$$

має асимптотично стандартний нормальний розподіл. Тому для обраного рівня значимості α повинно виконуватись: $\alpha = P(|\frac{I_n - MI_n}{\sqrt{DL_n}}| > I_\alpha) = 2 - 2F(I_\alpha)$, тобто I_α знаходиться з рівняння $F(I_\alpha) = 1 - \frac{\alpha}{2}$, де $F(x)$ – функція стандартного нормального розподілу.

Вважається, що при рівні значимості α послідовність пройшла тест, якщо $|\frac{I_n - MI_n}{\sqrt{DL_n}}| < I_\alpha$.

3.2 Оцінка запропонованих тестів

Дані критерії призначенні для тестування коротких послідовностей. Над шістнадцятковим алфавітом критерії χ^2 для знаків та серій максимальної довжини розраховані на послідовності, довжина яких перевищує 100 символів. Тоді як для інших тестів довжина послідовностей має бути не менше 1000. Тому у подальшій роботі будуть розглядатися послідовності довжини 1024, що задовольняє обидві умови. Як генератор псевдовипадкових послідовностей було обрано генератор байтів Блюма–Блюма–Шуба (BBS), бо він є одним із найефективніших відомих генераторів псевдовипадкових чисел.

У таблиці 3.1 наведені граничні значення статистик для кожного тесту при однаковому рівні значимості $\alpha = 0.01$.

Таблиця 3.1 – Теоретичні границі для шести тестів

Назва критерію	Границя
χ^2	$\chi^2_\alpha = 30.6$
Серій максимальної довжини	$S_\alpha = 5$
χ^2 для біграм	$X^2_\alpha = 309.781$
Кількості серій	$G_\alpha = 2.58$
Місць знаків	$\chi^2_\alpha = 30.6$
Інверсій	$I_\alpha = 2.58$

Перевіримо незалежність набору з $N = 6$ тестів, для кожного з яких $\alpha = 0.01$. Рівень значимості для тесту перевірки незалежності обрано $\beta = 0.01$. За формулою 2.6 отримано значення $a = 0.9415$. У таблиці 3.2 наведено значення параметрів, які були обчислені згідно методики, описаної у розділі 2.3.

Таблиця 3.2 – Перевірка незалежності тестів

Кількість послідовностей	σ	η	$erfc$	$erfc \geqslant \beta$
500	0.010	0.95	0.417	✓
1000	0.007	0.955	0.068	✓
1500	0.006	0.9567	0.012	✓
2000	0.005	0.9505	0.086	✓

Отже, згідно даної методики, тести слід вважати незалежними.

3.3 Перевірка статистичних властивостей алгоритму Krip

Статистичне дослідження потокового алгоритму шифрування Krip було проведено згідно методики, описаної у розділі 2.3. З умови незалежності цього набору з шести тестів за формулою 2.9 обчислили значення загальної помилки першого роду $A = 1 - 0.9415 = 0.0585$ при рівні значимості $\alpha = 0.01$. Нехай гіпотеза H_0 потягає у тому, що вихідні послідовності потокового шифру Krip є псевдовипадковими відносно даного набору статистичних тестів. У таблиці 3.3 наведені границі інтервалів Δ, які складаються із значень при яких гіпотеза H_0 приймається, для заданої кількості згенерованих повідомлень, .

Таблиця 3.3 – Інтервал Δ при різній кількості послідовностей

Кількість послідовностей	A	$\sigma(A,n)$	Δ
100	0.0585	0.023	(0.871; 1)
2000		0.005	(0.926; 0.957)

Всього було проведено 20 експериментів для 20 різних згенерованих ключів. Для кожного екперименту було зашифровано по 100 довільних послідовностей довжиною 1024 бітів за допомогою алгоритму Krip. Далі було проведено тестування кожного шифротексту допомогою шести

критеріїв. У таблиці 3.4 наведені відсотки послідовностей, які пройшли всі тести.

Таблиця 3.4 – Дослідження статистичних властивостей послідовностей, які були зашифровані алгоритмом Krip

Номер експерименту	Відсоток	Номер експерименту	Відсоток
1	0.92	11	0.97
2	0.97	12	0.96
3	0.97	13	1
4	0.94	14	0.92
5	0.94	15	0.97
6	0.99	16	0.93
7	0.96	17	0.94
8	0.94	18	0.99
9	0.95	19	0.96
10	0.95	20	0.95
Вибіркове математичне сподівання			0.956

Результати кожного з двадцяти екпериментів належать відповідному проміжку $(0{,}871; 1)$. Також частка послідовностей з 2000 шифротекстів, що пройшли тести, належить проміжку $(0.926; 0.957)$. Тому гіпотеза H_0 приймається. Отже, алгоритм Krip можна вважати генератором псевдовипадкових послідовностей.

Висновки до розділу 3

У цьому розділі наведений детальний опис шести статистичних критеріїв, призначених для тестування коротких послідовностей. Було обчислено максимальне допустиме значення для кожного критерію при рівні значимості $\alpha = 0.01$ та доведено незалежність цього набору тестів.

Також за допомогою цих тестів було проведено тестування 2000 шифротекстів, отриманих з використанням потокового алгоритму Krip, в результаті якого отримано 0.956 – відсоток послідовностей, які пройшли всі тести. Оскільки це значення належить інтервалу $(0.926; 0.957)$, то згідно методики криптографічної оцінки якості послідовностей з NIST зроблено висновок, що алгоритм Krip можна вважати генератором псевдовипадкових чисел.

ВИСНОВКИ

У ході виконання даної роботи був проведений аналіз опублікованих джерел за тематикою дослідження, яка полягає у статистичного аналізу криптографічних властивостей вихідних послідовностей алгоритмів шифрування.

Проаналізувавши теоретичний матеріал, було обрано шість статистичних критеріїв для тестування властивостей шифротекстів потокового алгоритму Krip, призначених для тестування коротких послідовностей. Також наведено їх детальний опис, обчислено максимальне допустиме значення для кожного критерію при рівні значимості $\alpha = 0.01$ та доведено незалежність цього набору тестів. Методикою тестування криптографічної оцінки якості вихідної гами потокового алгоритму шифрування Krip обрано NIST SP 800-22.

На мові програмування *C#* було написано програмну реалізацію алгоритму шифрування Krip та обраних шести тестів, необхідних для проведення статистичних досліджень.

За допомогою цих тестів проведено тестування достатньої кількості шифротекстів, отриманих з використанням потокового алгоритму Krip, в результаті якого отримано значення 0.956, що є відсотком послідовностей, які пройшли всі тести.

Зроблено висновки щодо криптографічної якості вихідних послідовностей з викоританням методити з NIST на базі отриманих результатів тестування, що алгоритм Krip можна вважати генератором псевдовипадкових чисел.

Напрямом подальшої роботи є продовжити дослідження властивостей потокового шифру Krip, оскільки цей алгоритм є новим і ще маловивченим.

ПЕРЕЛІК ПОСИЛАНЬ

[1] Roman Kochana та ін. *Statistical tests independence verification methods.* T. 192. 2021. DOI: 10.1016/j.procs.2021.09.038.

[2] C. E. Shannon. *Mathematical Theory of Cryptography.* T. 379. 1945.

[3] *Lightweight Cryptography.* National Institute of Standards та Technology. 2023. URL: https://csrc.nist.gov/Projects/Lightweight-Cryptography.

[4] Kerry A McKay та ін. *Report on lightweight cryptography.* T. NISTIR 811. 2017.

[5] Elena Dubrova та Martin Hell. *Espresso: A stream cipher for 5G wireless communication systems.* T. 9. 2017. DOI: 10.1007/s12095-015-0173-2.

[6] Olexandr Kuznetsov, Mariya Lutsenko та Dmytro Ivanenko. *Strumok stream cipher: Specification and basic properties.* 2017. DOI: 10.1109/INFOCOMMST.2016.7905335.

[7] L. V. Kovalchuk, I. V. Koriakov та A. N. Alekseychuk. *Krip: High-Speed Hardware-Oriented Stream Cipher Based on a Non-Autonomous Nonlinear Shift Register.* Springer Science та Business Media LLC, лют. 2023. DOI: 10.1007/s10559-023-00538-6. URL: https://doi.org/10.1007/s10559-023-00538-6.

[8] Roman Oliynykov та ін. *A New Encryption Standard of Ukraine : The Kalyna Block Cipher.* 2015.

[9] Andrew Rukhin та ін. *SP800-22:A statistical test suite for random and pseudorandom number generators for cryptographic applications.* T. 800. 2010.

[10] L. Kovalchuk та V. Bezditnyi. *A statistical tests independence checking intended for cryptographic properties of RNG estimation.* T. 18-23. Ukrainian Information Security Journal, 2006.

ДОДАТОК А ТЕКСТИ ПРОГРАМ

Код програми, яка реалізовує потоковий алгоритм шифрування Krip та тести, необхідні для проведення статистичного дослідження.

A.1 Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Krip
{
    class Program
    {
        static void Main(string[] args)
        {
            var f = new Algorithm();
            bool prog = true;
            int r = 0;
            while (prog == true)
            {
                Console.WriteLine("Press <<0>> - to form the initial state;" +
                    "\npress <<1>> - to encrypt the message;" +
                    "\npress <<2>> - to decrypt the message;" +
                    "\npress <<3>> - to test the algorithm;" +
                    "\npress <<4>> - to check whether the tests are independent;" +
                    "\npress any other key - to exit.");
                r = int.Parse(Console.ReadLine());
                switch (r)
                {
                    case 0:
                        f.GenerateInitialState();
                        break;
                    case 1:
```

```
                    Console.WriteLine("Enter the message to encrypt: ");
                    f.Crypt(false);
                    break;
                case 2:
                    Console.WriteLine("Enter the ciphertext to decrypt: ");
                    f.Crypt(true);
                    break;
                case 3:
                    f.Test();
                    break;
                case 4:
                    f.CheckTests();
                    break;
                default:
                    prog = false;
                    break;
            }
        }
    }
}
```

A.2 Algorithm.cs

```
class Algorithm
{
    LongOperation op = new LongOperation();
    Functions f = new Functions();
    Test t = new Test();
    List<int> s0 = new List<int>();
    List<int> s1 = new List<int>();
    List<int> x = new List<int>();
    List<int> y = new List<int>();

    public void Test()
    {
        int count = 0;
```

```
        double n = 0;
        for (int i = 0; i < 20; i++)
        {
            count = 0;
            List<int> k = f.Generators(32);
            for (int j = 0; j < 100; j++)
            {
                GenerateInitialState(k);
                x = f.Generators(512);
                y = Crypt(x.ToList(), s0.ToList(), s1.ToList());
                string seq = op.Output(y);
                if (seq.Length % 2 != 0)
                    seq = "0" + seq;
                if (t.X2sign(seq) && t.SeriesMaxLength(seq)
                    && t.X2bigram(y, seq) && t.NumberSeries(seq)
                    && t.PlaceSign(seq) && t.Inversion(seq))
                    count++;
            }
            Console.WriteLine("Fraction = {0}", count / 100.0);
            n += count;
        }
        Console.WriteLine("Average = {0}", n / 2000.0);
    }

    public void CheckTests()
    {
        int count = 0;
        int i = 0;
        for (int n = 500; n <= 2000; n += 500)
        {
            for (; i < n; i++)
            {
                x = f.Generators(512);
                string seq = op.Output(x);
                if (seq.Length % 2 != 0)
                    seq = "0" + seq;
                if (t.X2sign(seq) && t.SeriesMaxLength(seq) && t.X2bigram(x, seq)
                    && t.NumberSeries(seq) && t.PlaceSign(seq) && t.Inversion(seq))
                    count++;
```

```
        }
        Console.WriteLine("Fraction = {0}", count / Convert.ToDouble(n));
    }
}

public void Crypt(bool flag)
{
    if(flag)
    {
        op.Print(y);
        List<int> t = new List<int>();
        t = Crypt(y.ToList(), s0.ToList(), s1.ToList());
        Console.WriteLine("Decrypted message: ");
        op.Print(t);
        if (op.LongCmp(x, t) == 0)
            Console.WriteLine("YES");
        else Console.WriteLine("NO");
    }
    else
    {
        x = f.Generators(512);
        op.Print(x);
        y = Crypt(x.ToList(), s0.ToList(), s1.ToList());
        Console.WriteLine("Encrypted message: ");
        op.Print(y);
    }
}

public List<int> Crypt(List<int> c, List<int> a0, List<int> b1)
{
    List<int> s = new List<int>();
    List<int> w = new List<int>();
    for (int i = 0; i < c.Count; i += 32)
    {
        List<int> a = a0.ToList();
        List<int> b = b1.ToList();
        w.AddRange(f.f(b, a, c.GetRange(i, 32)));
        s = b.ToList();
        b = f.h(b, a, c.GetRange(i, 32));
```

```
                a = s.ToList();
            }
            return w;
        }

        public void GenerateInitialState()
        {
            GenerateInitialState(f.Generators(32));
        }

        public void GenerateInitialState(List<int> k)
        {
            List<int> C1 = op.InputRev("A09E667F3BCC908B2FB1366EA957D3E3
␣␣␣␣␣␣␣␣␣␣␣␣ADEC17512775099DA2F590B0667322A9");
            List<int> C2 = op.InputRev("B67AE8584CAA73B25742D7078B83B892
␣␣␣␣␣␣␣␣␣␣␣␣5D834CC53DA4798C720A6486E45A6E24");
            List<int> C3 = op.InputRev("C6EF372FE94F82BE73980C0B9DB90682
␣␣␣␣␣␣␣␣␣␣␣␣1044ED7E744E4A3F0D8D423A1831D2A4");
            List<int> C4 = op.InputRev("54FF53A5F1D36F1CEA7E61FC37A20D54
␣␣␣␣␣␣␣␣␣␣␣␣A77FE7B78415DFC8E34A6FE8E2DF92A4");
            List<int> C5 = op.InputRev("10E527FADE682D1DE49E330E42B4CBB2
␣␣␣␣␣␣␣␣␣␣␣␣9BA5A455316E0C65507CD18E9E51E694");
            List<int> C6 = op.InputRev("B05688C2B3E6C1FDBD99E6FF3C90BDC4
␣␣␣␣␣␣␣␣␣␣␣␣DBC64712A5BB168767E27C3CF76C8E72");

            List<int> c = f.Generators(16);
            string k1 = "", k2 = "", c1 = "", c2 = "";
            for (int i = 0, j = 16; i < 16; i++, j++)
            {
                k1 += f.AddNull(Convert.ToString(k[i], 2));
                k2 += f.AddNull(Convert.ToString(k[j], 2));
                c1 += f.AddNull(Convert.ToString(c[i], 2));
                c2 += f.Eor(c1.Substring(8 * i, 8), "11111111");
            }

            List<int> Kl = f.bittobyte(f.Eor(k1, c2) + f.Eor(f.Eor(k1, k2), c1));
            List<int> Kr = f.bittobyte(f.Eor(f.Eor(k1, k2), c2) + f.Eor(k2, c2));

            List<int> U1 = f.h(Kl, Kr, C1);
```

```
List<int> U2 = Kl.ToList();
List<int> U = U1.ToList();
U1 = f.h(U1, U2, C2);
U2 = U.ToList();

U = Kr.GetRange(16, 16);
U.AddRange(Kl.GetRange(0, 16));
U1 = f.Xor(U1, U);
U = Kl.GetRange(16, 16);
U.AddRange(Kr.GetRange(0, 16));
U2 = f.Xor(U2, U);

List<int> Kal = f.h(U1, U2, C3);
List<int> Kar = U1.ToList();
U = Kal.ToList();
Kal = f.h(Kal, Kar, C4);
Kar = U.ToList();

U = U2.GetRange(16, 16);
U.AddRange(U1.GetRange(0, 16));
List<int> U0 = U1.GetRange(16, 16);
U0.AddRange(U2.GetRange(0, 16));
U1 = f.Xor(Kal, U);
U2 = f.Xor(Kar, U0);

List<int> Kbl = f.h(U1, U2, C5);
List<int> Kbr = U1.ToList();
U = Kbl.ToList();
Kbl = f.h(Kbl, Kbr, C6);
Kbr = U.ToList();

U1 = Kl.ToList();
U2 = Kr.ToList();

for (int i = 0; i < 32; i++)
{
    U = U1.ToList();
    U1 = f.h(U1, U2, f.Const(i, Kl, Kr, Kal, Kar, Kbl, Kbr));
    U2 = U.ToList();
```

```
        }
        s0 = U1.ToList();
        s1 = U2.ToList();
    }
}
```

A.3 Test.cs

```
    class Test
{
    int N = 16;

    public bool X2sign(string s)
    {
        int n = s.Length;
        double Xa2 = 30.6; //alpha = 0.01

        double X2 = 0;
        List<int> M = new List<int>(new int[N]);
        for (int i = 0; i < n; i++)
            M[Convert.ToInt32(s[i].ToString(), 16)]++;
        for (int i = 0; i < N; i++)
            X2 += Math.Pow(M[i], 2);
        X2 = X2 * N / n - n;

        if (X2 < Xa2) return true;
        return false;
    }

    public bool SeriesMaxLength(string s)
    {
        double alpha = 0.01;
        int n = s.Length;
        int Sa = Convert.ToInt32((Math.Log10(n * N / alpha) / Math.Log10(N)));

        int S = 0;
        for (int i = 0, j = 1; i < n - 1; i++, j = 1)
```

```
        {
            while (s[i] == s[i + j])
            {
                j++;
                if (i + j == n)
                    break;
            }
            if (S < j) S = j;
        }

        if (S <= Sa) return true;
        return false;
    }

    public bool X2bigram(List<int> l, string s)
    {
        int N2 = (int)Math.Pow(N, 2);
        int k = l.Count;
        double za = 2.33; // alpha = 0.01
        double Xa2 = Math.Pow(Math.Sqrt(2 * (N2 - 1) - 1) + za, 2) / 2;

        double X2 = 0;
        List<int> M = new List<int>(new int[N2]);
        for (int i = 0; i < l.Count; i++)
            M[l[i]]++;
        for (int i = 0; i < N2; i++)
            X2 += Math.Pow(M[i], 2);
        X2 = X2 * N2 / k - k;

        if (X2 < Xa2) return true;
        return false;
    }

    public bool NumberSeries(string s)
    {
        int n = s.Length;
        double Ga = 2.58; //alpha = 0.01

        int Gn = 1;
```

```
    for (int i = 0; i < n - 1; i++)
        if (s[i] != s[i + 1]) Gn++;
    double MGn = (n - 1) * (1 - 1 / Convert.ToDouble(N)) + 1;
    double DGn = (n - 1) * (N - 1) / Math.Pow(N, 2);
    double G = Math.Abs((Gn - MGn) / Math.Sqrt(DGn));

    if (G < Ga) return true;
    return false;
}

public bool PlaceSign(string s)
{
    int n = s.Length;
    double Xa2 = 30.6; //alpha = 0.01

    double T = 0;
    List<int> Rv = new List<int>(new int[N]);
    for (int j = 0; j < N; j++)
    {
        char v = Convert.ToChar(Convert.ToString(j, 16).ToUpper());
        for (int i = 0; i < n; i++)
            if (s[i] == v) Rv[j] += i;
    }
    for (int i = 0; i < N; i++)
        T += Math.Pow(Rv[i] / Convert.ToDouble(n) - (n + 1) / Convert.ToDouble(2 * N
    T *= 3 * N / Convert.ToDouble(n);

    if (T < Xa2) return true;
    return false;
}

public bool Inversion(string s)
{
    int n = s.Length;
    double Ia = 2.58; //alpha = 0.01

    int In = 0;
    for (int i = 0; i < n; i += 2)
        if (s[i] < s[i + 1]) In++;
```

```
            double MIn = n / 2 * (0.5 - 1 / Convert.ToDouble(2 * N));
            double DIn = n / 2 * (0.25 - 1 / (4 * Math.Pow(N, 2)));
            double I = Math.Abs((In - MIn) / Math.Sqrt(DIn));

            if (I < Ia) return true;
            return false;
        }
    }
```

A.4 Functions.cs

```
class Functions
{
    LongOperation op = new LongOperation();
    List<int> p0 = new List<int> { 168, 67, 95, 6, 107, 117, 108, 89, 113, 223,
    135, 149, 23, 240, 216, 9, 109, 243, 29, 203, 201, 77, 44, 175, 121, 224,
    151, 253, 111, 75, 69, 57, 62, 221, 163, 79, 180, 182, 154, 14, 31, 191, 21,
    225, 73, 210, 147, 198, 146, 114, 158, 97, 209, 99, 250, 238, 244, 25, 213,
    173, 88, 164, 187, 161, 220, 242, 131, 55, 66, 228, 122, 50, 156, 204, 171,
    74, 143, 110, 4, 39, 46, 231, 226, 90, 150, 22, 35, 43, 194, 101, 102, 15,
    188, 169, 71, 65, 52, 72, 252, 183, 106, 136, 165, 83, 134, 249, 91, 219,
    56, 123, 195, 30, 34, 51, 36, 40, 54, 199, 178, 59, 142, 119, 186, 245, 20,
    159, 8, 85, 155, 76, 254, 96, 92, 218, 24, 70, 205, 125, 33, 176, 63, 27,
    137, 255, 235, 132, 105, 58, 157, 215, 211, 112, 103, 64, 181, 222, 93, 48,
    145, 177, 120, 17, 1, 229, 0, 104, 152, 160, 197, 2, 166, 116, 45, 11, 162,
    118, 179, 190, 206, 189, 174, 233, 138, 49, 28, 236, 241, 153, 148, 170,
    246, 38, 47, 239, 232, 140, 53, 3, 212, 127, 251, 5, 193, 94, 144, 32, 61,
    130, 247, 234, 10, 13, 126, 248, 80, 26, 196, 7, 87, 184, 60, 98, 227, 200,
    172, 82, 100, 16, 208, 217, 19, 12, 18, 41, 81, 185, 207, 214, 115, 141,
    129, 84, 192, 237, 78, 68, 167, 42, 133, 37, 230, 202, 124, 139, 86, 128 };
    List<int> p1 = new List<int> { 206, 187, 235, 146, 234, 203, 19, 193, 233,
    58, 214, 178, 210, 144, 23, 248, 66, 21, 86, 180, 101, 28, 136, 67, 197, 92,
    54, 186, 245, 87, 103, 141, 49, 246, 100, 88, 158, 244, 34, 170, 117, 15, 2,
    177, 223, 109, 115, 77, 124, 38, 46, 247, 8, 93, 68, 62, 159, 20, 200, 174,
    84, 16, 216, 188, 26, 107, 105, 243, 189, 51, 171, 250, 209, 155, 104, 78,
    22, 149, 145, 238, 76, 99, 142, 91, 204, 60, 25, 161, 129, 73, 123, 217,
    111, 55, 96, 202, 231, 43, 72, 253, 150, 69, 252, 65, 18, 13, 121, 229, 137,
```

```
140, 227, 32,48, 220, 183, 108, 74, 181, 63, 151, 212, 98, 45, 6, 164, 165,
131, 95, 42, 218, 201, 0, 126, 162, 85, 191, 17, 213, 156, 207, 14, 10, 61,
81, 125, 147, 27, 254, 196, 71, 9, 134, 11, 143, 157, 106, 7, 185, 176, 152,
24, 50, 113, 75,239, 59, 112, 160, 228, 64, 255, 195, 169, 230, 120, 249,
139, 70, 128, 30, 56, 225, 184, 168, 224, 12, 35, 118, 29, 37, 36, 5, 241,
110, 148, 40, 154, 132, 232, 163, 79, 119, 211, 133, 226, 82, 242, 130, 80,
122, 47, 116, 83, 179, 97,175, 57, 53, 222, 205, 31, 153, 172, 173, 114, 44,
221, 208, 135, 190, 94, 166,236, 4, 198, 3, 52, 251, 219, 89, 182, 194, 1,
240, 90, 237, 167, 102, 33, 127,138, 39, 199, 192, 41, 215 };
List<int> p2 = new List<int> { 147, 217, 154, 181, 152, 34, 69, 252, 186,
106, 223, 2, 159, 220, 81, 89, 74, 23, 43, 194, 148, 244, 187, 163, 98, 228,
113, 212, 205, 112, 22, 225, 73, 60, 192, 216, 92, 155, 173, 133, 83, 161,
122, 200,45, 224, 209, 114, 166, 44, 196, 227, 118, 120, 183, 180, 9, 59,
14, 65, 76, 222, 178, 144, 37, 165, 215, 3, 17, 0, 195, 46, 146, 239, 78,
18, 157, 125, 203, 53, 16, 213, 79, 158, 77, 169, 85, 198, 208, 123, 24,
151, 211, 54, 230, 72, 86, 129, 143, 119, 204, 156, 185, 226, 172, 184, 47,
21, 164, 124, 218, 56,30, 11, 5, 214, 20, 110, 108, 126, 102, 253, 177, 229,
96, 175, 94, 51, 135, 201, 240, 93, 109, 63, 136, 141, 199, 247, 29, 233,
236, 237, 128, 41, 39, 207,153, 168, 80, 15, 55, 36, 40, 48, 149, 210, 62,
91, 64, 131, 179, 105, 87, 31, 7, 28, 138, 188, 32, 235, 206, 142, 171, 238,
49, 162, 115, 249, 202, 58, 26, 251, 13, 193, 254, 250, 242, 111, 189, 150,
221, 67, 82, 182, 8, 243, 174, 190, 25, 137, 50, 38, 176, 234, 75, 100, 132,
130, 107, 245, 121, 191, 1, 95, 117, 99, 27, 35, 61, 104, 42, 101, 232, 145,
246, 255, 19, 88, 241, 71, 10, 127, 197, 167, 231, 97, 90, 6, 70, 68, 66, 4,
160, 219, 57, 134, 84, 170, 140, 52, 33, 139, 248, 12, 116, 103 };
List<int> p3 = new List<int> { 104, 141, 202, 77, 115, 75, 78, 42, 212, 82,
38, 179, 84, 30, 25, 31, 34, 3, 70, 61, 45, 74, 83, 131, 19, 138, 183, 213,
37, 121, 245, 189, 88, 47, 13, 2, 237, 81, 158, 17, 242, 62, 85, 94, 209,
22, 60, 102, 112, 93, 243, 69, 64, 204, 232, 148, 86, 8, 206, 26, 58, 210,
225, 223, 181, 56, 110, 14, 229, 244, 249, 134, 233, 79, 214, 133, 35, 207,
50, 153, 49, 20, 174, 238, 200, 72, 211, 48, 161, 146, 65, 177, 24, 196, 44,
113, 114, 68, 21, 253, 55, 190, 95, 170, 155, 136, 216, 171, 137, 156, 250,
96, 234, 188, 98, 12, 36, 166, 168, 236, 103, 32, 219, 124, 40, 221, 172,
91, 52, 126, 16, 241, 123, 143, 99, 160, 5, 154, 67, 119, 33, 191, 39, 9,
195, 159, 182, 215, 41, 194, 235, 192, 164, 139, 140, 29, 251, 255, 193,
178, 151, 46, 248, 101, 246, 117, 7, 4, 73, 51, 228, 217, 185, 208, 66, 199,
108, 144, 0, 142, 111, 80, 1, 197, 218, 71, 63, 205, 105, 162, 226, 122,
167, 198, 147, 15, 10, 6, 230, 43, 150, 163, 28, 175, 106, 18, 132, 57, 231,
176, 130, 247, 254, 157, 135, 92, 129, 53, 222, 180, 165, 252, 128, 239,
```

```
203, 187, 107, 118, 186, 90, 125, 120, 11, 149, 227, 173, 116, 152, 59, 54,
100, 109, 220, 240, 89, 169, 76, 23, 127, 145, 184, 201, 87, 27, 224, 97 };

public List<int> h(List<int> s1, List<int> s2, List<int> x)
{
    List<int> res = Phi(Xor(x, s1));
    while (res.Count < s2.Count)
        res.Insert(0, 0);
    res = Xor(s2, res);
    return res;
}

public List<int> f(List<int> s1, List<int> s2, List<int> x)
{
    List<int> res = s1.ToList();
    res.AddRange(s2);
    res = Xor(x, Psi(res));
    return res;
}

public List<int> Phi(List<int> u)
{
    List<List<int>> x = new List<List<int>>();
    List<int> v = new List<int>();
    for (int i = 0; i < 8; i++)
    {
        List<int> y = new List<int>();
        List<int> p = Substitution(i % 4);
        for (int j = 0; j < 4; j++)
            y.Add(p[u[8 * j + i]]);
        switch (i / 2)
        {
            case 0: break;
            case 1:
                y.Add(y[0]);
                y.RemoveAt(0);
                break;
            case 2:
                for (int l = 0; l < 2; l++)
```

```
                {
                    y.Add(y[0]);
                    y.RemoveAt(0);
                }
                break;
            default:
                for (int l = 0; l < 3; l++)
                {
                    y.Add(y[0]);
                    y.RemoveAt(0);
                }
                break;
        }
        x.Add(y);
    }
    List<int> d = new List<int> { 1, 1, 5, 1, 8, 6, 7, 4 };
    string mod = "100011101";
    for (int j = 0; j < 4; j++)
    {
        for (int i = 0; i < 8; i++)
        {
            string l = Convert.ToString(x[i][j], 2);
            string w = Convert.ToString(d[i], 2);
            string s0, s = "", nul = "";
            bool flag = false;
            for (int r = w.Length - 1; r >= 0; r--)
            {
                if (w[r] == '1')
                {
                    s0 = l + nul;
                    if (flag)
                    {
                        while (s.Length < s0.Length)
                            s = "0" + s;
                        s = Eor(s0, s);
                        s = s.TrimStart('0');
                        if (s.Length == 0)
                            s = "0";
                    }
```

```
                    else s = s0;
                    flag = true;
                }
                nul += "0";
            }
            while (s.Length > 8)
            {
                string m = mod;
                while (m.Length < s.Length)
                    m += "0";
                s = Eor(s, m);
                s = s.TrimStart('0');
                if (s.Length == 0)
                    s = "0";
            }
            v.Add(Convert.ToInt32(s, 2));
        }
        d.Add(d[0]);
        d.RemoveAt(0);
    }
    return v;
}

public List<int> Psi(List<int> u)
{
    List<List<int>> x = new List<List<int>>();
    List<int> v = new List<int>();
    List<int> p = new List<int>();
    for (int i = 0; i < 8; i++)
    {
        List<int> y = new List<int>();
        p = Substitution(i % 4).ToList();
        for (int j = 0; j < 8; j++)
            y.Add(p[u[8 * j + i]]);
        for (int t = 0; t < i; t++)
        {
            y.Insert(0, y.Last());
            y.RemoveAt(y.Count - 1);
        }
```

```
        x.Add(y);
    }
    List<int> d = new List<int> { 1, 1, 5, 1, 8, 6, 7, 4 };
    string mod = "100011101";
    for (int j = 0; j < 8; j++)
    {
        for (int i = 0; i < 8; i++)
        {
            string l = Convert.ToString(x[i][j], 2);
            string w = Convert.ToString(d[i], 2);
            string s0, s = "", nul = "";
            bool flag = false;
            for (int r = w.Length - 1; r >= 0; r--)
            {
                if (w[r] == '1')
                {
                    s0 = l + nul;
                    if (flag)
                    {
                        while (s.Length < s0.Length)
                            s = "0" + s;
                        s = Eor(s0, s);
                        s = s.TrimStart('0');
                        if (s.Length == 0)
                            s = "0";
                    }
                    else s = s0;
                    flag = true;
                }
                nul += "0";
            }
            while (s.Length > 8)
            {
                string m = mod;
                while (m.Length < s.Length)
                    m += "0";
                s = Eor(s, m);
                s = s.TrimStart('0');
                if (s.Length == 0)
```

```
                    s = "0";
                }
                v.Add(Convert.ToInt32(s, 2));
            }
            d.Add(d[0]);
            d.RemoveAt(0);
        }
        for (int i = 0; i < v.Count; i++)
        {
            p = Substitution(i % 4).ToList();
            v[i] = p[v[i]];
        }
        for (int i = 0; i < 32; i++)
        {
            v[32] = Convert.ToInt32(Eor(AddNull(Convert.ToString(v[0], 2)),
            AddNull(Convert.ToString(v[32], 2))), 2);
            v.RemoveAt(0);
        }
        return v;
    }

    public List<int> Const(int i, List<int> Kl, List<int> Kr, List<int> Kal,
    List<int> Kar, List<int> Kbl, List<int> Kbr)
    {
        List<int> K = new List<int>();
        switch(i+1)
        {
            case 1: K = Kl.ToList();break;
            case 2: K = Kr.ToList();break;
            case 3: K = Kbl.ToList();break;
            case 4: K = Kbr.ToList();break;
            case 5: K = Shift(Kal, 15);break;
            case 6: K = Shift(Kar, 15);break;
            case 7: K = Xor(Kl, Shift(Kbl, 15));break;
            case 8: K = Xor(Kr, Shift(Kbr, 15));break;
            case 9: K = Shift(Kal, 30);break;
            case 10: K = Shift(Kar, 30);break;
            case 11: K = Shift(Kbl, 30);break;
            case 12: K = Shift(Kbr, 30);break;
```

```
            case 13: K = Xor(Kl, Shift(Kal, 45));break;
            case 14: K = Xor(Kr, Shift(Kar, 45));break;
            case 15: K = Shift(Kbl, 45);break;
            case 16: K = Shift(Kbr, 45);break;
            case 17: K = Shift(Kal, 60);break;
            case 18: K = Shift(Kar, 60);break;
            case 19: K = Shift(Kl, 60);break;
            case 20: K = Shift(Kr, 60);break;
            case 21: K = Xor(Kal, Shift(Kbl, 77));break;
            case 22: K = Xor(Kar, Shift(Kbr, 77));break;
            case 23: K = Shift(Kbl, 77);break;
            case 24: K = Shift(Kbr, 77);break;
            case 25: K = Shift(Kal, 94);break;
            case 26: K = Shift(Kar, 94);break;
            case 27: K = Xor(Kl, Shift(Kr, 94));break;
            case 28: K = Xor(Kr, Shift(Kl, 94));break;
            case 29: K = Xor(Kal, Shift(Kar, 111));break;
            case 30: K = Xor(Kar, Shift(Kal, 111));break;
            case 31: K = Xor(Kbl, Shift(Kbr, 111));break;
            default: K = Xor(Kbr, Shift(Kbl, 111));break;
        }
        return K;
    }

    private List<int> Shift(List<int> U, int n)
    {
        string u = "";
        for (int i = 0; i < U.Count; i++)
            u += AddNull(Convert.ToString(U[i], 2));
        u = u.Substring(u.Length - n, n) + u.Substring(0, u.Length - n);
        U = bittobyte(u);
        return U.ToList();
    }

    public List<int> Substitution(int n)
    {
        switch(n)
        {
            case 0: return p0;
```

```
            case 1: return p1;
            case 2: return p2;
            default: return p3;
        }
    }

    public List<int> Generators(int size)
    {
        return BBS(size);
    }

    private List<int> BBS(int size)
    {
        Random R = new Random();
        List<int> p = op.Input("D5BBB96D30086EC484EBA3D7F9CAEB07");
        List<int> q = op.Input("425D2B9BFDB25B9CF6C416CC6E37B59C1F");
        List<int> n = op.LongMul(p, q);
        List<int> m = op.LongM(n);
        List<int> x = new List<int>();
        List<int> r = op.Rand(R.Next(2, p.Count / 2));
        r.Add(1);
        for (int i = 0; i < size; i++)
        {
            r = op.LongMod(op.LongMul(r, r), n, m);
            x.Add(r[0]);
            if (i == size - 1 && x.Last() == 0)
                i -= 1;
        }
        if (x.Count != size)
            x.RemoveAt(size);
        return x;
    }

    public string AddNull(string s)
    {
        while (s.Length < 8)
            s = "0" + s;
        return s;
    }
```

```
    public string Eor(string a, string b)
    {
        string c = "";
        for (int i = 0; i < a.Length; i++)
        {
            if (a[i] == b[i]) c += "0";
            else c += "1";
        }
        return c;
    }

    public List<int> Xor(List<int> a, List<int> b)
    {
        string s;
        List<int> c = new List<int>();
        for (int i = 0; i < a.Count; i++)
        {
            s = Eor(AddNull(Convert.ToString(a[i], 2)),
            AddNull(Convert.ToString(b[i], 2)));
            c.Add(Convert.ToInt32(s, 2));
        }
        return c;
    }

    public List<int> bittobyte(string s)
    {
        List<int> res = new List<int>();
        for (int i = 0; i < s.Length; i += 8)
            res.Add(Convert.ToInt32(s.Substring(i, 8), 2));
        return res;
    }
}
```

A.5 LongOperation.cs

```
class LongOperation
```

```
{
    public void Print(List<int> l)
    {
        for (int i = 0; i < l.Count; i++)
            Console.Write(l[i] + "␣");
        Console.WriteLine();
    }

    public List<int> Input(string a)
    {
        List<int> l = new List<int>();
        if (a.Length % 2 != 0)
            a = "0" + a;
        for (int i = a.Length - 1; i > 0; i -= 2)
            l.Add(Convert.ToInt32(Convert.ToString(a[i - 1]), 16) * 16 +
            Convert.ToInt32(Convert.ToString(a[i]), 16));
        return l;
    }

    public List<int> InputRev(string a)
    {
        List<int> l = new List<int>();
        if (a.Length % 2 != 0)
            a = "0" + a;
        for (int i = 0; i < a.Length; i += 2)
            l.Add(Convert.ToInt32(Convert.ToString(a[i]), 16) * 16 +
            Convert.ToInt32(Convert.ToString(a[i + 1]), 16));
        return l;
    }

    public string Output(List<int> a)
    {
        string s = "";
        for (int i = 0; i < a.Count; i++)
        {
            string h = Convert.ToString(a[i], 16);
            if (h.Length != 2) h = "0" + h;
            s = h + s;
        }
```

```
    s = s.TrimStart('0');
    return s.ToUpper();
}

public List<int> LongAdd(List<int> a1, List<int> b1)
{
    List<int> a = new List<int>();
    List<int> b = new List<int>();
    if (a1.Count < b1.Count)
    {
        a = b1.ToList();
        b = a1.ToList();
    }
    else
    {
        b = b1.ToList();
        a = a1.ToList();
    }
    List<int> c = new List<int>();
    int carry = 0, temp = 0;
    int i = 0;
    for (i = 0; i < b.Count; i++)
    {
        temp = a[i] + b[i] + carry;
        c.Add(temp & 255);
        carry = temp >> 8;
    }
    for (; i < a.Count; i++)
    {
        temp = a[i] + carry;
        c.Add(temp & 255);
        carry = temp >> 8;
    }
    if (carry != 0)
        c.Add(carry);
    return c;
}

public List<int> LongSub(List<int> a, List<int> b)
```

```
{
    a = LongSameSize(a, b.Count);
    b = LongSameSize(b, a.Count);
    int size = a.Count;
    List<int> c = new List<int>();
    int borrow = 0, temp = 0;
    for (int i = 0; i < size; i++)
    {
        temp = a[i] - b[i] - borrow;
        if (temp >= 0)
        {
            c.Add(temp);
            borrow = 0;
        }
        else
        {
            c.Add(256 + temp);
            borrow = 1;
        }
    }
    a = TrueSize(a);
    b = TrueSize(b);
    return TrueSize(c);
}

public List<int> LongMul(List<int> p, List<int> q)
{
    p = LongSameSize(p, q.Count);
    q = LongSameSize(q, p.Count);
    List<int> c = new List<int>();
    List<int> temp = new List<int>();
    for (int i = 0; i < p.Count; i++)
    {
        temp = LongMulOneDigit(p, q[i]);
        temp = LongShiftDigitsToHight(temp, i);
        c = LongSameSize(c, temp.Count);
        temp = LongSameSize(temp, c.Count);
        c = LongAdd(c, temp);
    }
```

```
    p = TrueSize(p);
    q = TrueSize(q);
    return TrueSize(c);
}

public List<int> LongDivMod(List<int> a, List<int> b)
{
    List<int> c = new List<int>();
    int t, k = b.Count;
    List<int> q = new List<int>(new int[a.Count]);
    while (LongCmp(a, b) != -1)
    {
        t = a.Count;
        c = LongShiftDigitsToHight(b, t - k);
        if (LongCmp(a, c) == -1)
        {
            t--;
            c = LongShiftDigitsToHight(b, t - k);
        }
        a = LongSub(a, c);
        q[t - k]++;
    }
    return TrueSize(q);
}

public List<int> LongMod(List<int> c, List<int> n, List<int> m)
{
    List<int> q = KillLastDigits(c, n.Count - 1);
    q = LongMul(q, m);
    q = KillLastDigits(q, n.Count + 1);
    q = LongMul(q, n);
    c = LongSub(c, q);
    while (LongCmp(c, n) != -1)
        c = LongSub(c, n);
    return c;
}

private List<int> LongMulOneDigit(List<int> a, int b)
{
```

```
    int size = a.Count;
    List<int> c = new List<int>();
    int temp = 0, carry = 0;
    for (int i = 0; i < size; i++)
    {
        temp = a[i] * b + carry;
        c.Add(temp & 255);
        carry = temp >> 8;
    }
    c.Add(carry);
    return TrueSize(c);
}

public int LongCmp(List<int> a, List<int> b)
{
    if (a.Count > b.Count)
        return 1;
    else if (b.Count > a.Count)
        return -1;
    int i = a.Count - 1;
    while (a[i] == b[i])
    {
        if (i == 0)
            return 0;
        i -= 1;
    }
    if (a[i] > b[i])
        return 1;
    else
        return -1;
}

public List<int> LongM(List<int> n)
{
    int k = n.Count;
    List<int> temp = new List<int> { 1 };
    List<int> m = LongShiftDigitsToHight(temp, 2 * k);
    m = LongDivMod(m, n);
    return m;
```

```
}

private List<int> KillLastDigits(List<int> a, int i)
{
    List<int> p = a.ToList();
    if (p.Count > i)
        p.RemoveRange(0, i);
    else
    {
        p.Clear();
        p.Add(0);
    }
    return p;
}

private List<int> TrueSize(List<int> c)
{
    c.Reverse();
    while (c.Count != 1 && c[0] == 0)
        c.RemoveAt(0);
    c.Reverse();
    return c;
}

private List<int> LongShiftDigitsToHight(List<int> b, int i)
{
    List<int> c = b.ToList();
    for (int j = 0; j < i; j++)
        c.Insert(0, 0);
    return c;
}

private List<int> LongSameSize(List<int> a, int size)
{
    for (int i = a.Count; i < size; i++)
        a.Add(0);
    return a;
}
```

```
    public List<int> Rand(double n)
    {
        Random R = new Random();
        List<int> l = new List<int>();
        for (int i = 0; i < n; i++)
            l.Add(R.Next(0, 256));
        return l;
    }
}
```