НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

«На правах рукопису»

УДК _____

«До захисту допущено»

Завідувач кафедри

_____ *Сергій СТІРЕНКО*
(підпис) (ім'я, прізвище)

"____" _____ 2022 р.

Магістерська дисертація

зі спеціальності: *123. Комп'ютерна інженерія*
(код та назва напряму підготовки або спеціальності)

Спеціалізація: *123. Комп'ютерні системи та мережі*

на тему: Метод підвищення ефективності генерації псевдовипадкових послідовностей для потокового шифрування даних

Виконав: студент _II_ курсу, групи ІО-01мн
(шифр групи)

Лисенко Дмитро Вадимович _____
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник доц., к.т.н., доц. Марковський О.П.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант проф., д.т.н., проф. Кулаков Ю.О.
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент декан ФПМ, д.т.н., проф. Дичка І.А.
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2022 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *Обчислювальної техніки*
(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою
Спеціальність *123. Комп'ютерна інженерія*
(код і назва)

Спеціалізація *123. Комп'ютерні системи та мережі*
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
Сергій СТІРЕНКО
(підпис) (ініціали, прізвище)
« » 2022 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Лисенко Дмитро Вадимович
(прізвище, ім'я, по батькові)

1. Тема дисертації Метод підвищення ефективності генерації псевдовипадкових послідовностей для потокового шифрування даних

Науковий керівник дисертації доц., к.т.н., доц. Марковський О.П.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від « 26» квітня 2022 р. № НС/88/2022

2. Строк подання студентом дисертації

3. Об'єкт дослідження процеси генерації псевдовипадкових двійкових послідовностей, орієнтованих на використання для захисту інформації в комп'ютерних системах і мережах, а також проектування програмно-апаратних засобів формування таких послідовностей

4. Предмет дослідження способи підвищення функціональної ефектив¬ності використання псевдовипадкових двійкових послідовностей для захисту інформації в комп'ютерних системах, а також методи проектування апаратно-програмних засобів генерації послідовностей, що мають покращені характеристики нелінійності

5. Перелік завдань, які потрібно розробити: аналіз використання генераторів псевдовипадкових послідовностей та огляд поточного стану генераторів, дослідження нелінійних функцій зворотного зв'язку для зсувного регістру, метод побудови генератора псевдовипадкових послідовностей на основі нелінійних функцій зворотного зв'язку, розробка програмного забезпечення методу побудови генератора псевдовипадкових послідовностей.

6. Консультанти розділів дисертації:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
нормоконтроль	Кулаков Ю.О., проф. д.т.н.		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів дисертації	Примітка
1	Затвердження теми роботи	31.01.2022	
2	Аналіз поставленого завдання	01.02.2022-13.02.2022	
3	Аналіз існуючих рішень	14.02.2022-27.02.2022	
4	Розробка нового методу	28.02.2022-27.03.2022	
5	Розробка програмного забезпечення	28.03.2022-24.04.2022	
6	Оформлення пояснювальної записки	25.04.2022-02.06.2022	
7	Захист	14.06.2022	

Студент _____ Д.В. Лисенко
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ О.П. Марковський
(підпис) (ініціали, прізвище)

РЕФЕРАТ

Робота складається із вступу та трьох розділів. Загальний обсяг роботи: 102 аркуші основного тексту, 10 ілюстрації, 4 таблиць. При підготовці використовувалася література з 80 джерел.

Тема магістерської дисертації: Метод підвищення ефективності генерації псевдовипадкових послідовностей для потокового шифрування даних

Актуальність теми дослідження. Генератори псевдовипадкових об'єктів (чисел, двійкових послідовностей та функцій) широко використовуються в сучасних системах захисту інформації. Особливо важливу роль в цій важливій сфері відіграють генератори псевдовипадкових двійкових послідовностей, які використовуються в якості базового елементу потокових алгоритмів захисту даних.

В сучасних умовах зростання продуктивності обчислювальних систем і можливостей об'єднання значної кількості комп'ютерів для порушення захисту даних, актуальною є проблема адекватного підвищення надійності захисту інформації, в тому числі засобами, в основі яких лежить використання псевдовипадкових послідовностей. Зростання швидкодії обчислювальних систем та швидкості передачі цифрових даних в лініях комп'ютерних мереж ставить більш жорсткі вимоги до продуктивності засобів захисту інформації, в тому числі, в основі яких лежить використання генераторів псевдовипадкових послідовностей: вони мають забезпечувати реалізацію формування елементів послідовності в темпі передачі даних.

Таким чином, на сучасному етапі розвитку техніки захисту інформації в в системах та мережах обчислювальної техніки актуальною стає проблема розробки способів підвищення ефективності генерації псевдовипадкових двійкових послідовностей.

Мета досліджень полягає в підвищення ефективності застосування для захисту інформації в комп'ютерних системах псевдовипадкових двійкових послідовностей за рахунок вдосконалення програмно-апаратних засобів їх генерації

та поліпшення їх характеристик, що визначають здатність протидії несанкціонованого доступу до даних.

Основні задачі дослідження визначаються поставленою метою і полягають в наступному:

1. Аналіз сучасного стану та тенденцій використання псевдовипадкових двійкових послідовностей для захисту інформації в комп'ютерних системах і мережах. Обгрунтування на основі результатів аналізу вимог до псевдовипадкових послідовностей, орієнтованих на використання для захисту інформації, а також критеріїв ефективності програмно-апаратних засобів їх формування.

2. Огляд та аналіз з позицій виявлених критеріїв існуючих програмно-апаратних засобів генерації псевдовипадкових двійкових послідовностей для систем захисту інформації. Виявлення можливостей підвищення ефективності використання псевдовипадкових послідовностей та шляхів вдосконалення програмно-апаратних засобів їх генерації.

3. Теоретичні дослідження властивостей нелінійних булевих функцій зворотного зв'язку зсувного регістру, які забезпечують максимальний період повторення коду на ньому. Вивчення властивостей нелінійних функцій зворотного зв'язку, які забезпечують період повторення близький до максимального. Комбінаторний аналіз кодових кілець – груп кодів, які утворюються при циклічному зсуві.

4. Вдосконалення методу формування нелінійних булевих функцій зворотного зв'язку для NFSR шляхом організації рекурсивного знаходження шляхів Ейлера в графі переходів між станами зсувного регістру для зменшення витрат ресурсів пам'яті.

5. Розробка ітераційного методу отримання підмножини нелінійних булевих функцій зворотного зв'язку, що забезпечують максимальний період повторення його станів.

6. Розробка програмних засобів генерації псевдовипадкових двійкових послідовностей, які реалізують запропоновані методи побудови нелінійні функцій зворотного зв’язку зсувного регістру.

Об’єктом дослідження є процеси генерації псевдовипадкових двійкових послідовностей, орієнтованих на використання для захисту інформації в комп’ютерних системах і мережах, а також проектування програмно-апаратних засобів формування таких послідовностей.

Предмет дослідження – способи підвищення функціональної ефективності використання псевдовипадкових двійкових послідовностей для захисту інформації в комп’ютерних системах, а також методи проектування апаратно-програмних засобів генерації послідовностей, що мають покращені характеристики нелінійності.

Методи дослідження базуються на теорії ймовірності та математичної статистики, теорії булевих функцій та комбінаторики, теорії організації обчислювальних процесів, а також на використанні методів моделювання.

Наукова новизна полягає в наступному:

1. Розвинуті положення теорії зсувних регістрів з нелінійною функцією зворотного зв’язку: виявлені нові властивості нелінійних булевих функцій зворотного зв’язку, що забезпечують максимальний період повторення коду на зсувному регістрі, одержані аналітичні оцінки кількості кодових кілець, важливі для ефективного вирішення прикладних задач проектування засобів генерації псевдовипадкових двійкових послідовностей на зсувних регістрах.

2. Вдосконалено метод формування нелінійних булевих функцій зворотного зв’язку для NFSR шляхом організації рекурсивного знаходження шляхів Ейлера в графі переходів між станами зсувного регістру, запропоновані вдосконалення полягають в рекурсивній процедури переходу від циклів різних рівнів, що дозволяє не фіксувати в повному обсязі всі цикли попереднього рівня і таким чином досягти зменшення витрат ресурсів пам'яті.

3. Вперше запропоновано ітераційний методу отримання підмножини нелінійних булевих функцій зворотного зв’язку, що забезпечують максимальний період повторення його станів, який відрізняється тим, що спочатку допоміжний вектор заповнення, а потім з використанням цього вектору здійснюється побудова функції зворотного зв’язку, що дозволяє спростити і прискорити процес побудови NFSR у порівнянні з відомими методами .

Практичне значення одержаних результатів роботи визначається тим, що розроблені методи проектування схем формування псевдовипадкових двійкових послідовностей на зсувних регістрах з використанням нелінійних функцій зворотного зв’язку зсувних регістрів замість традиційних лінійних дозволяє значно знизити можливість екстраполяції послідовностей, тим самим підвищити ефективність систем захисту інформації на їх основі. Результати роботи можуть бути застосовані для підвищення ефективності захисту інформації в комп’ютерних системах, швидкісних кабельних та бездротових каналах передачі даних комп’ютерних мереж, системах мобільного зв’язку.

Особистий внесок здобувача полягає в теоретичному обґрунтуванні одержаних результатів, їх експериментальній перевірці та дослідженні, а також у створенні програмних продуктів для практичного використання одержаних результатів.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ГПБР - генератор псевдовипадкових бітових рядків

КГ – конгруентний генератор

ЛВМ – лінійна відтворююча модель

ПБР - псевдовипадкові бітові рядки

LFSR (Linear Feedback Shift Register) – зсувний регістр з лінійною функцією зворотного зв'язку

NFSR (Nonlinear Feedback Shift Register) – зсувний регістр із нелінійною функцією зворотного зв'язку

RFSR (Random Feedback Shift Register) – зсувний регістр із випадковою зворотного зв'язку

ВСТУП

З плином часу збільшується питома вага обчислювальних систем реального часу. Прикладами є системи моніторингу, відеоспостереження, керування дорожнім трафіком, тощо. Зазвичай у таких розподілених системах використовується велика кількість вузлів, які мають автономне живлення, що обмежує складність мікросхем. Ще одним прикладом систем з передачею даних у реальному часі є чіпи, які імплантуються у органи людини і живляться від температури тіла. В описаних вище випадках, доступні апаратні ресурси є обмеженими.

Передача даних переважно відбувається по відкритим каналам зв’язку і потребує захисту. Для шифрування у реальному часі зазвичай використовуються потокові шифри, в основі яких лежать генератори псевдовипадкових двійкових послідовностей.

Проте сучасні криптографічно стійкі алгоритми мають складну реалізацію, що потребує більше апаратних ресурсів. Це також збільшує час, який необхідний для обрахування наступного біту.

Оскільки у вбудованих системах використовуються прості комбінаційні схеми, то така складність алгоритмів призводить до небажаних наслідків. А саме, використання менш надійних алгоритмів або їх програмного обчислення на універсальних процесорах. При чому другий спосіб значно сповільнює генерацію.

Таким чином, виникає необхідність створення нових методів розробки алгоритмів генерації псевдовипадкових двійкових послідовностей, які будуть працювати у системах реального часу найбільш ефективно.

Виділимо критерії, за якими будемо виконувати оцінку генераторів. По-перше, необхідний високий рівень захисту інформації. Тобто отримана послідовність практично не піддається криптоаналізу. По-друге, реалізація має

бути проста, щоб використовувалось якнайменше апаратних ресурсів. І по-третє, обрахунки повинні виконуватись якомога швидше.

РОЗДІЛ 1
АНАЛІЗ ВИКОРИСТАННЯ ГЕНЕРАТОРІВ ПСЕВДОВИПАДКОВИХ ПОСЛІДОВНОСТЕЙ ТА ОГЛЯД ПОТОЧНОГО СТАНУ ГЕНЕРАТОРІВ

1.1 Аналіз використання генераторів псевдовипадкових послідовностей у сучасних умовах

Генератор псевдовипадкових послідовностей у класичному вигляді представляє собою деякий повністю математичний алгоритм, який на виході видає числа або двійкові послідовності, які схожі на дійсно випадкові числа та послідовності. Основними елементами у таких генераторах є пам'ять розміром n, яка зберігає стан генератора, і деяке перетворення, за допомогою якого обраховується наступний стан генератора та послідовність, яка подається на вихід. Такий алгоритм завжди може бути відображений у еквівалентний автомат.

Вихідні двійкова послідовність алгоритму не є дійсно випадковою. По суті, для всіх послідовностей довжиною k, загальне число всі можливих послідовностей становить $\frac{2^n}{2^k}$. Послідовність вважається псевдовипадковою, якщо не має такого алгоритму, який би при поставлених обмеженнях по обчислювальним ресурсам і часу, зміг би відрізнити дану послідовність від дійсно випадкової.

Головною особливістю генератора псевдовипадкових послідовностей є те, що при однаковому початковому векторі IV, на виході він видасть однакові послідовності, тоді як дійсно випадковий генератор видасть, скоріш за все, іншу послідовність. Хоча все ще існує невеликий шанс того, що ці послідовності співпадуть.

Класичний генератор зазвичай використовує попереднє значення стану для того, щоб обчислити наступний стан. Схема такого генератора наведена на рисунку 1.1.

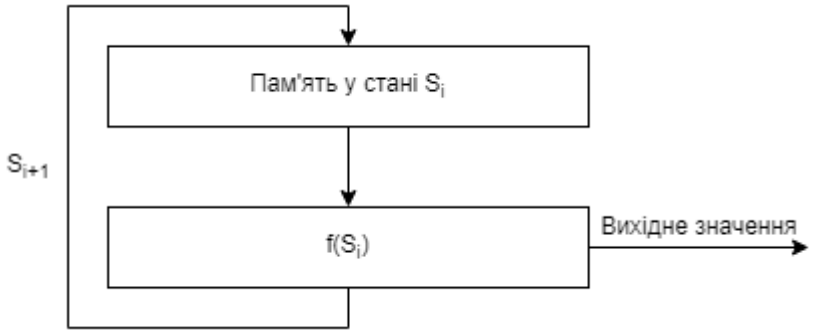

Рис.1.1. Загальна структура генератору псевдовипадкових послідовностей

Такий генератор включає у себе пам'ять, яка зберігає поточний стан регістра, та деяке перетворення над цим станом $f(S_i)$. Перед початком роботи генератора, пам'ять встановлюється у деяке початкове значення S_0=IV. На кожній ітерації, результат перетворення записується у пам'ять і стає новим станом генератора. Також на вихід подається чергове вихідне значення псевдовипадкової послідовності.

Важливим елементом є функція f(S). Вона описує перетворення, яке визначає важливі характеристики генератора, а саме період повторення, статистичні показники послідовності, складність відтворення моделі без початкового значення. В основі таких перетворень можна використати різні види функцій. Для того, щоб забезпечити простоту комбінаційних схем, зазвичай використовують булеві функції. Такі функції можуть бути швидко обчислені на апаратній основі.

Аргументом таких функцій є двійковий вектор із n бітів $S=(s_1,\ldots,s_n)$, $s_i \epsilon \{0,1\}$, i=1,..,n. Після перетворення, отримуємо вихідний вектор Y=f(S). Додатково може бути додана окрема функція для формування вихідного біту. Для реалізації перетворення зазвичай використовують нелінійні функції, або селектор, який обирає певні біти.

Одним із прикладів використання генераторів псевдовипадкових послідовностей, це отримання псевдовипадкового числа у заданому інтервалі.

Наприклад, для отримання числа віл 0 до n включно, можна згенерувати деяку двійкову послідовність довжиною log(n)+1 і відобразити її у десятковому вигляді користувачу. При цьому може статись, що отримане число більше за n. В такому випадку його требі відкинути і повторити процедуру генерації. Такі числа будуть мати рівномірний розподіл. А маючи рівномірний розподіл, можна на його основі отримати інші розподіли.

Ще одним застосуванням генератора, є використання згенерованої послідовності $K=(k_1,\ldots,k_n)$ для шифрування повідомлення $M=(m_1,\ldots,m_n)$ потоковим шифром. Таким чином, отримуємо шифротекст $C=(c_1,\ldots,c_n)$, як зображено на рисунку 1.2. Таке шифрування називають потоковим шифруванням, оскільки воно використовує псевдовипадковий потік у якості ключа.

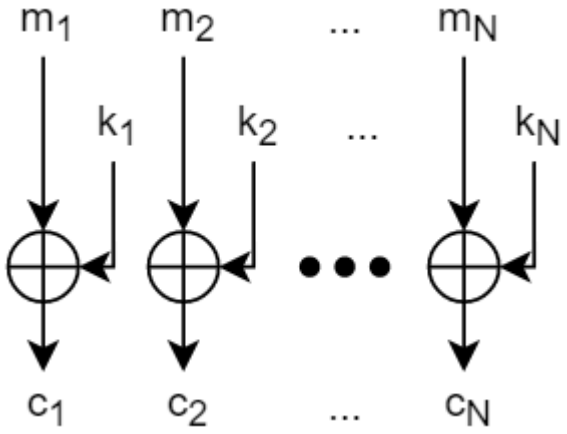

Рис.1.2. Схема потокового шифрування

Таке шифрування формально описується формулою 1.1.

$$C = M \oplus K. \tag{1.1}$$

Тоді як дешифрування відповідає формулі 1.2.

$$C \oplus K = M \oplus K \oplus K = M \oplus 0 = M. \tag{1.2}$$

Тобто для шифрування деякого повідомлення, необхідний ключ такої самої довжини. А значить, що для безпечної передачі повідомлення великої довжини, треба попередньо передати безпечно ключ такої самої довжини. Виходить, що такий шифр працює лише за попередньою домовленістю про ключ та про максимальну довжину повідомлення. У іншому ж випадку, виникає потреба безпечної передачі ключа такої самої довжини, як і повідомлення яке шифрується. Тобто початкова проблема знову зберігається.

Загальна послідовність використання потокового шифрування виглядає наступним чином:

1. Відправник та отримувач домовляються який генератор псевдовипадкових чисел використовувати.
2. Відправник обирає початковий стан генератора псевдовипадкових чисел S_1.
3. Відправник по захищеному каналу передає отримувачу початковий стан генератора S_1 один раз.
4. Відправник генерує ключ K_i за допомогою генератора псевдовипадкових чисел та стану S_i, де i – номер повідомлення. Генератор опиняється у стані S_{i+1}.
5. Відправник використовує згенерований ключ K_i для шифрування повідомлення M_i за допомогою операції XOR, та отримує зашифроване повідомлення C_i.
6. Відправник відправляє зашифроване повідомлення C_i по відкритому каналу до отримувача.
7. Отримувач, знаючи початковий стан генератора псевдовипадкових чисел S_1, генерує такий самий ключ K_i.

8. Отримувач використовує згенерований ключ K_i для дешифрування зашифрованого повідомлення C_i за допомогою операції XOR, та отримує оригінальне повідомлення M_i.

Передача початкового стану по захищеному каналу може відбуватися на фізичному носії, або з використанням більш складного шифрування. У другому випадку можна було б відправити оригінальне повідомлення з використанням того самого більш складного шифрування. Але тоді необхідно витрачати більше ресурсів (особливо при великих повідомленнях). Також у деяких випадках, потокове шифрування більше підходить для даної задачі, наприклад потокова передача даних.

При необхідності, відправник може продовжувати використовувати генератор псевдовипадкових послідовностей, продовжуючи зі стану S_{i+1} – стан, у якому опинився генератор після генерації ключа K_i.

Проте, такі генератори мають період, після якого послідовність псевдовипадкових чисел починає повторюватись. Використання однакових ключів для шифрування за допомогою операції XOR не є безпечним. Адже тоді зловмисник може частково отримати оригінальні повідомлення з шифротексту, який передається по відкритому каналу, за формулою 1.3 при умові, що шифротекст був сформований наступним чином: $C_1 = M_1 \oplus K$ та $C_2 = M_2 \oplus K$.

$$C_1 \oplus C_2 = (M_1 \oplus K) \oplus (M_2 \oplus K) = (M_1 \oplus M_2) \oplus (K \oplus K) = M_1 \oplus M_2. \quad (1.3)$$

У випадку тексту або зображень, можна виконати статистичний аналіз для відновлення оригінальних повідомлень M_1 та M_2. Також він може відновити ключ К, який використовувався для шифруванні, і якщо даний ключ знову бути використаний, то йому більше не треба бути проводити статистичний аналіз, адже він вже знає ключ. Також більша кількість повторних використань ключа К спрощує виконання статистичного аналізу для відновлення оригінальних повідомлень.

Тоді для забезпечення криптографічної стійкості, необхідно мати достатньо довгу послідовність щоб забезпечити ключ достатньої довжини.

Іншими словами, генератори псевдовипадкових послідовностей мають забезпечувати розбіжність послідовності, тобто послідовність має мати якомога більший період. Якщо стан генератора описується із n біт, то тоді максимальний теоретично можливий період становить 2^n. Таким чином, для забезпечення розбіжності, період генератор має бути близьким, або рівним 2^n.

Окрім потокового шифрування, генератори псевдовипадкових послідовностей також використовуються у ряді інших задач, які направлені на захист інформації:

- генерація випадкових паролів для доступу до певної системи, що значно ускладнює підбір паролю;
- генерація унікальних ідентифікаторів користувачів, які використовуються у деяких протоколах;
- генерація ключа для алгоритмів з симетричним шифруванням, або для алгоритмів, де частково відомий ключ.

На практиці, найважливішим застосуванням генераторів псевдовипадкових послідовностей, залишається потокове шифрування, яке потребує відтворюваність потоку на основі однакового початкового вектору. Тоді як для генерації паролів, ідентифікаторів та ключів краще використовувати дійсно випадкові генератори.

При аналізі застосувань генераторів псевдовипадкових послідовностей, було виявлено, що окрім вимог стосовно статистичних властивостей і періоду, при використанні генератору для захисту інформації, обов'язково додається критерій непередбачуваності послідовності. Для того, щоб даний критерій виконувався, генератор має використовувати незворотні перетворення.

При цьому для потокових алгоритмів шифрування, які використовуються при передачі інформації, також є важливою швидкодія генератора. Також важливо виключити можливість втручання до програми, яка генерує псевдовипадкову послідовність. Враховуючи ці дві потреби, виходить, що генератор краще реалізовувати апаратно. В такому випадку додається критерій складності комбінаційної схеми, яка реалізує генератор, а саме дана складність має бути якомога нижчою.

Тоді при побудові генераторів виникає проблема у тому, що необхідно забезпечити ряд критеріїв криптографічно стійкості послідовності (статистичні характеристики, період, непередбачуваність) і при цьому забезпечити низьку апаратну складність.

1.2 Аналіз критеріїв ефективності криптографічних генераторів псевдовипадкових послідовностей

Можна виділити наступні три критерії ефективності криптографічного генератора для потоку ключів:

- швидкодія;
- простота;
- криптографічна стійкість.

Обчислення на сучасних комп'ютерах є доволі швидкими, і навіть складні нелінійні перетворення можна швидко обчислити на центральному процесорі.

Хоча даний критерій дійсно є найважливішим для криптографічного генератора, для вбудованих систем з автономним живленням також є важливими швидкодія та простота генератора, що використовується.

Простота та швидкодія, хоча і корелюють, є різними поняттями.

Швидкодія генератора важлива для того, щоб можна було швидко зашифрувати повідомлення та передати його навіть на мікросхемах, які мають обмежені ресурси (як обрахункові, так і ресурси живлення).

Простота генератора означає, що його можна легко реалізувати на мікросхемі з використанням невеликої кількості логічних елементів. Таким чином, якщо шифрування відбувається з великою частотою (наприклад, елементи системи моніторингу у реальному часу постійно відправляють свої спостереження, які необхідно шифрувати; або центральний комп'ютер, який отримує ці спостереження і йому потрібно їх всі дешифрувати), то генератор можна реалізувати у "залізі" вбудованої системи.

Це дозволяє:

- прибрати зайве навантаження з центрального процесору;
- пришвидшити час обрахунку, оскільки тепер обчислення відбуваються на спеціалізованій мікросхемі, замість універсального процесору;
- зменшити використання живлення за рахунок меншої кількості елементів.

Криптографічна стійкість генератора все ще залишається найважливішим критерієм, оскільки основна задача криптографічного генератора – це саме забезпечення криптографічної стійкості шифротексту, що передається по відкритому каналу.

Криптографічна стійкість генератора передбачає, що згенеровану послідовність практично неможливо відрізнити від дійсно випадкової, якщо невідомий стан генератора. Тобто неможливо знайти якісь залежності, або кореляцію між бітами послідовності, і відповідно передбачити наступні біти.

Таким чином, навіть якщо зловмисник дізнається частину повідомлення M якимось чином і з нього отримає частину згенерованої послідовності K за формулою 1.4 (шифротекст C вважається завжди відомим зловмиснику, оскільки C передається по відкритому каналу), то він не зможе відновити іншу частину

послідовності і розшифрувати іншу частину повідомлення або інші повідомлення.

$$M \oplus C = M \oplus (M \oplus K) = M \oplus M \oplus K = K. \quad (1.4)$$

Отже, окрім головного критерію криптографічних генераторів (криптографічна стійкість), у роботі також ставиться мета забезпечення ефективності для вбудованих систем (швидкодія та простота).

1.3 Огляд існуючих генераторів псевдовипадкових послідовностей

Усі генератори псевдовипадкових послідовностей можна виділити у дві основні групи.

Перша група – це генератори для некриптографічних задач (наприклад, моделювання деяких випадкових подій). Для таких задач можуть використовуватися конгруентні генератори, зсувні регістри з лінійним зворотним зв'язком, функціонуючі в полях Галуа. При побудові таких генераторів не ставиться задача забезпечення непередбачуваності послідовності.

Друга група – для криптографічних задач. Такий генератор має забезпечувати непередбачуваність згенерованої послідовності, навіть якщо її попередня або наступна частини відомі. Для того, щоб забезпечити дану умову, генератор має використовувати деяку функцію f(S), яка є практично незворотною. За типом незворотних функцій перетворення, криптографічні генератори можна поділити на наступні категорії:

- використовується функція f(S) деякого потокового шифру;
- використовується функція f(S) деякого блокового шифру;
- використовується деяка одностороння функція f(S);
- використовуються блоки з стохастичним перетворенням.

Серед наведених типів генераторів, найбільш простим є конгруентний генератор. Зазвичай у таких генераторах використовується дія множення, результат якої урізається деяким максимальним значенням, зазвичай операцією

mod. Приклад перетворення, яке може використовуватись, наведено нижче у формулі 1.5.

$$S_{n+1} = (a * S_n + b)\ mod\ M, \quad (1.5)$$

де S_{n+1} – наступна послідовність яка буде згенерована, S_n – поточна послідовність, M, b та a – незмінні коефіцієнти генератору.

Початкове значення такого генератору задається деяким значенням S_0.

Коефіцієнти a та b, а також початковий стан S_0 мають бути невід'ємними, значення M має бути додатнім. Коефіцієнти a та b, та значення S_0 обираються такими, щоб вони були менші за M.

У результаті роботи такого генератора отримуємо лінійну конгруентну послідовність $x_1,x_2,x_3,\ldots$. Період ніколи не буде перевищувати значення M, оскільки всі можливі значення станів генератора знаходяться у діапазоні від нуля до M-1.

Саме значення M зазвичай обирають 2^k, це дозволяє запросто реалізувати операцію mod шляхом відсічення старших розрядів отриманого числа.

Для того, щоб отримати максимальний період M, необхідно щоб виконувались наступні умови:

- b та M мають бути взаємно простими;
- a-1 повинно бути кратним до всіх простих дільників числа M;
- у випадках, якщо Ь ділить на 4, то a-1 також має ділитися на 4.

Основною проблемою таких генераторів є погані статистичні характеристики (згенеровані підряд значення залежать один від одного, деякі біти мають нерівномірний розподіл). При цьому, якщо старший біт має приблизно рівномірний розподіл, то молодші біти не будуть задовольняти рівномірному розподілу. Також великим недоліком є те, що на основі згенерованих значень можна відновити попереднє значення, тобто перетворення є зворотним. Період такого генератора також є доволі низьким.

Тому дані генератори не є придатними для використання для захисту інформації, але вони часто застосовуються у інших задачах, оскільки такі генератори є доволі швидкими.

Інший тип генератора у якості основної операції використовує ділення, повна операція наведена у формулі 1.6.

$$S_{n+1} = \frac{S_n}{a} * M, \tag{1.6}$$

де a та M – постійні коефіцієнти, при чому M є ступенем двійки, a – просте число великого розміру.

У порівнянні з попереднім алгоритмом, у цьому алгоритмі за рахунок операції ділення трохи ускладняється генерація, і на універсальних процесорах, вона займає більше часу. Статистичні властивості та довжина періоду кращі, ніж за попередній алгоритм, але дані проблеми все ще присутні. Також залишається проблема оберненості перетворення.

Значним покращенням є алгоритм, який використовує піднесення до ступеню (формула 1.7).

$$S_{n+1} = (S_n)^k \, mod \, M. \tag{1.7}$$

При чому, чим більше значення k, тим кращі будуть статистичні властивості згенерованої послідовності. Збільшення ступеню також робить зворотне перетворення складнішим.

Є окремий випадок даного генератора, а саме, коли значення n є добутком двох простих чисел. У цьому випадку, даний генератор є широко розповсюдженим RSA генератором [1], який на сьогодні часто використовується для захисту інформації.

Ще одним простим типом генераторів псевдовипадкових послідовностей є регістр зсуву з лінійним зворотним зв'язком, або linear-feedback shift register (LFSR).

Такий генератор використовує лише один зсувний регістр та лінійні перетворення над бітами у цьому регістрі (зазвичай це просто операція XOR над певними бітами регістру). Результат лінійних перетворень подається на вхід зсувного регістру. Вихід регістру генерує псевдовипадкову послідовність.

Зворотній зв'язок також часто описують у вигляді поліномів на полях Галуа. Приклад поліному, який забезпечує максимальний період послідовності, наведено у формулі 1.8.

$$x^{16} + x^{14} + x^{13} + x^{11} + 1. \quad (1.8)$$

Причому для кожного LFSR з максимальним періодом відповідає примітивний поліном у полі Галуа. Вперше формально примітивні поліноми описав Соломон Ґоломб у 1967 році [2]. Наразі можна знайти списки примітивних поліномів від різних авторів. Наприклад Xilinx опублікували поліноми для регістрів розміром до 168 бітів у інструкції до своїх зсувних регістрів [3]. Також існують більш повні списки [4] та засоби для генерації таких поліномів [5].

Таким чином, з використанням всього одного регістра та декількох компонентів XOR, забезпечується простота та швидкодія генерації псевдовипадкової двійкової послідовності.

Але оскільки такий генератор використовує лише лінійні перетворення, то згенерована послідовність легко піддається лінійному криптографічному аналізу.

Більш того, зловмисник маючи частину згенерованої послідовності (наприклад дізнавшись іншим чином частину повідомлення, і використовуючи формулу 1.6 відновить ключ з шифротексту), може легко отримати генератор, який би згенерував таку саму послідовність. Наприклад, можна використати алгоритм Берлекемпа-Мессі [6], за допомогою якого можна отримати найпростіший LFSR для цієї послідовності.

Отже, критерій криптографічної стійкості не виконується для генераторів типу LFSR.

Для того, щоб зробити лінійний аналіз неможливим, існує модифікація LFSR, з додаванням нелінійних перетворень. Такі генератори відносять до категорії регістрів зсуву з нелінійним зворотним зв'язком, або nonlinear-feedback shift register (NLFSR).

Обчислення вхідного біту зсувного регістру все ще відбувається на основі поточного стану регістру (тобто використовуються біти цього регістру), але перетворення поточного стану тепер відбувається з використанням нелінійних перетворень. Операція XOR може все ще використовуватись, але обов'язково використовуються нелінійні перетворення як OR, NOR, AND, NAND тощо.

При розробці таких генераторів важливо переконатись, що нелінійна функція, яка використовується, не зводиться до лінійної функції, адже тоді знову виходить LFSR, який не є криптографічно стійким.

Також важливо переконатись, щоб період був достатньо довгим, адже більшість випадково обраних нелінійних функцій призводять до коротких періодів.

Хоча лінійний аналіз не допоможе при аналізі NLFSR, існує ще один спосіб криптографічного аналізу псевдовипадкової послідовності, а саме – диференційний аналіз.

Зазвичай, диференційний аналіз використовують для блокових шифрів, але він також застосовується для аналізу інших криптографічних алгоритмів (потокові шифри, геш-функції).

Суть диференційного аналізу полягає в тому, що порівнюються вихідні значення при зміні вхідних значень. Наприклад, збирається статистика різниць вихідних значень при зміні першого біту вхідних значень. Якщо алгоритм не є криптографічно стійким, то такий аналіз може показати залежність вихідних значень від зміни вхідних (наприклад, при зміні певного біту вхідного значення, змінюється певний біт вихідного значення).

У випадку потокового шифрування, диференційний аналіз виконуються над станом генератора та згенерованою послідовністю. Таким чином, якщо зловмисник виконає диференційний аналіз алгоритму і знайде залежність між різницею вхідних значень (стани генератора) та різницею вихідних значень (згенеровані біти послідовності), то маючи частину згенерованої послідовності, він може наблизитись до, або навіть знайти стан генератора. А знаючи стан генератора, він може згенерувати подальші біти послідовності та розшифрувати відповідні повідомлення.

Якщо ж розглядати конкретно LFSR або NLFSR, то для диференційного аналізу можна також взяти функцію, яка використовується у генераторі. Якщо зловмисник знайде диференційну залежність між станами регістру та новим бітом, то це також наблизить його до знаходження стану регістру.

Отже, вибір такої нелінійної функції насправді є доволі складною задачею, але при правильно обраній функції, генератор NLFSR повинен бути криптографічно стійким.

Приклади відомих функцій можна побачити у наступних джерелах [7-10]. Можна помітити, що відомі нелінійні функції є доволі невеликого розміру. Такі функції призначені для регістрів невеликого розміру. Так, якщо взяти функцію для регістру із 27 біт, то загальна кількість станів регістру не буде перевищувати 2^{27} станів. Це означає, що період не буде перевищувати даного значення. Тобто з одного початкового стану можна зашифрувати максимум 2^{27} біт (або 16 мегабайт). У сучасному світі це доволі малий об'єм інформації.

Проблема синтезу нелінійних функцій для регістрів більшого розміру і досі залишається відкритою проблемою.

Отже, NLFSR є доволі простими та швидкими у плані кількості логічних елементів. Але залишається відкритою проблема синтезу нелінійних функцій достатньо великого розміру. Теоретично, такі генератори мають бути

криптографічно стійким, але із-за труднощей з отриманням таких функцій, вони ще не були достатньо проаналізовані на криптографічну стійкість.

Ще один із способів покращення криптографічної стійкості LFSR – використання декількох зсувних регістрів з лінійним зворотнім зв'язком.

Але просто використання більшої кількості регістрів недостатньо для усунення можливості лінійного аналізу. Для покращення криптографічно стійкості, до алгоритму додають деякі нелінійні дії.

Наприклад, виходи регістрів з лінійними зв'язками можуть поєднуватись між собою нелінійними зв'язками для отримання бітів згенерованої послідовності. Ще одним способом може бути непостійний зсув регістрів. Тобто для генерації одного біту послідовності, деякі регістри можуть виконати зсув більше одного разу, або не виконати зсув взагалі.

Таким чином можна отримати простий та швидкий алгоритм з використанням всього декількох регістрів.

Криптографічна стійкість таких генераторів залежить від конкретних алгоритмів, отже розглянемо деякі з них.

Найбільш популярним прикладом таких шифрів є сімейство алгоритмів A5, оскільки дані шифри використовуються у найбільш популярній системі мобільному зв'язку GSM. Алгоритми даних шифрів були засекречені, але стали відомими через витік даних та зворотної інженерії.

Найбільш вживаними є наступні алгоритми із даного сімейства:

- A5/0. Даний алгоритм насправді не використовує генератор і видає на виході оригінальну послідовність, тобто ніякого шифрування не відбувається.
- A5/1. Використовується генератор із трьох LFSR з непостійним зсувом. Даний шифр є найбільш криптографічно стійким із даного сімейства.

- A5/2. Використовується генератор із чотирьох LFSR з непостійним зсувом та нелінійним об'єднанням регістрів.
- A5/3. Даний алгоритм використовує блоковий шифр KASUMI для генерації псевдовипадкової послідовності. Це інший спосіб отримання генератора, який буде розглянуто у окремому пункті.

Інші алгоритми даного сімейства практично не використовуються, тому їх було опущено з цього списку.

Використовуються три LFSR різного розміру – 19, 22 та 23 біти. Виходи даних зсувних регістрів поєднуються за допомогою операції XOR, і після цього результат використовується як черговий біт послідовності.

Проте зсув регістрів відбувається нерегулярно, а з використанням спеціальних бітів, які підсвічені оранжевим кольором на схемі. Зсуваються тільки ті регістри, які співпадають з більшістю. Наприклад, якщо перший та другий регістри мають виділені біти у значенні нуль, а третій регістр – у значенні один, то зсув буде виконуватись лише для першого та другого регістрів. Таким чином на кожному такті буде виконуватись зсув для двох або трьох регістрів.

Алгоритм A5/1 також передбачає ініціалізацію регістрів за допомогою 64-бітового ключа та 22-бітової відомої послідовності, але нас цікавить криптографічна стійкість самого генератору.

Хоча даний алгоритм є широковживаним, насправді він не є достатньо криптографічно стійким і є багато досліджень на цю тему.

Перша атака на даний алгоритм була запропонована у 1994 році Росс Андерсоном [16]. Його ідея полягала у тому, щоб з використанням чіпів перебрати значення перших двох регістрів та половину третього регістру. Таким чином можна отримати коли які регістри зсуваються і обчислити іншу половину третього регістру.

Існує також багато інших досліджень з різними типами атак на алгоритм A5/1 [17-20].

Отже, даний алгоритм не є достатньо криптографічно стійким і для захисту даних варто використовувати або розробляти інші алгоритми.

Суть таких алгоритмів полягає у тому, щоб виконувати якісь перетворення над регістром, які відмінні від зсуву як в LFSR або NLFSR.

Популярним прикладом таких генераторів є алгоритм RC4, який також був засекречений, поки він не став відомим через витік даних.

Генератор даного алгоритму має один регістр з 256 байт та два вказівники і та j. У самому регістрі зберігаються всі можливі значення байтів від 0 до 255 у деякому порядку. Кожен з вказівників вказує на деяку позицію у цьому байті.

Основна ідея полягає у тому, що байти регістру S постійно переставляються між собою. Одна ітерація алгоритму виглядає наступним чином:

1. i = (i + 1) mod 256
2. j = (j + S[i]) mod 256
3. Переставити S[i] та S[j] між собою
4. Видати на вихід значення S[(S[i] + S[j]) mod 256]

Таким чином, за одну ітерацію отримується один байт псевдовипадкової послідовності, яку далі можна використати для шифрування повідомлення, яке передається.

Хоча даний алгоритм вважався криптографічно стійким і був широко використований (наприклад, у протоколах WEP, TLS, тощо), сучасні протоколі даний алгоритм більше не застосовують із-за знайдених проблем з криптографічною стійкістю.

Наприклад, у 2001 році була знайдена вразливість даного алгоритму, яка була використана для злому протоколу WEP. [26]

Слід зазначити, що у 2011 році була продемонстрована атака BEAST, яка зламувала протокол TLS, якщо використовувався алгоритм шифрування, відмінний від RC4. [27] Тобто у той час RC4 навпаки показав свою криптографічну стійкість.

Але було знайдено багато інших вразливостей RC4 [28-30], які спричинили видалення його з протоколу TLS [31].

Отже, даний алгоритм є відносно простим і швидким, але наразі він не є криптографічно стійким.

Існує також багато інших типів генераторів для отримання псевдовипадкових послідовностей.

Так, наприклад, є дослідження на тему генераторів на основі Гамільтонової системи (Hamiltonian system) [37,38], логістичного відображення (logistic map) [39], на основі двох систем Лоренца та Lü [40], квантового блукання (quantum walk) [41], хаотичної гри (метод створення фракталів) [42], відображення Хенона [43], тощо.

Основною проблемою даних генераторів є те, що вони використовують складні математичні перетворення, які не задовольняють встановлений критерій простоти генератора. Тобто для їх реалізації у вбудованих системах необхідно використати значно більше елементів, що також збільшує час обрахунку.

Слід зазначити, що також існує невелика кількість досліджень, які зосереджені саме на генераторах для вбудованих систем, і цілю цих досліджень є саме забезпечення ефективності генератора.

Наприклад, у 2018 році було запропоновано генетичний алгоритм для генерації ефективних генераторів для вбудованих систем. [49]. Автори зазначають, що попередні дослідженню на дану тему [50-52] не враховували складність отриманих генераторів, і тому вони не підходять для вбудованих

систем. Тоді як їх алгоритм видає на виході прості генератори з використанням невеликої кількості простих операцій.

Після отримання генераторів, автори виконали тестування, і в результаті усі генератори не пройшли як мінімум один тест. Тобто генератори, отримані даним способом, не задовольняють критерій криптографічної стійкості.

У 2020 році була опублікована стаття, у якій автори описали генератор псевдовипадкових послідовностей на основі хаотичних відображень, а саме на основі логістичного та LEL відображень [53]. Отриманий генератор проходить тести NIST[22] на криптографічну стійкість. Хоча генератори і є простими у програмній реалізації, їх апаратна реалізація є більш складною.

Також є дослідження щодо простих генераторів, які засновані на використанні дійсно випадкових чисел з використанням датчиків та сенсорів [54,55]. Але даний варіант не підходить, оскільки нам необхідно мати детерміновані генератори, послідовності з яких можуть згенерувати як відправник, так і отримувач.

Висновки до розділу 1

У першому розділі було поставлено ціль проаналізувати поточний стан генераторів псевдовипадкових послідовностей, та знайти потенційні способи покращення генерації псевдовипадкової послідовності.

У результаті проведеного аналізу генераторів, можна зробити наступні висновки:

1. На сьогоднішній день, генератори псевдовипадкових послідовностей є широко вживаними для захисту інформації у мережах та комп'ютерних системах. Основною перевагою генераторів є те, що вони дозволяють швидко зашифрувати та розшифрувати повідомлення. Оскільки з кожним днем зростають обсяги даних, які зберігаються та передаються, а також у зв'язку з поширенням систем реального часу, існує перспектива, що збільшиться потреба у використанні генераторів псевдовипадкових послідовностей для шифрування даних у комп'ютерних системах.

2. При використанні генераторів псевдовипадкових послідовностей, значно збільшуються вимоги стосовно програмних і апаратних засобів їх генерації, а також до самих послідовностей, які генеруються. Генератори псевдовипадкових послідовностей мають також забезпечувати максимальний період повторення послідовності 2^n, та певні статистичні властивості цієї послідовності. Також генератори мають забезпечувати криптографічну стійкість до аналізу для неможливості передбачення послідовності, та достатню швидкодію для систем реального часу.

3. Щоб послідовність було неможливо передбачити, алгоритм її генерації має використовувати необоротні функції. Під час аналізу було виявлено, що у якості таких функцій в існуючих генераторах псевдовипадкових послідовностей використовуються перетворення з теорії чисел, нелінійні булеві функції, стандартизовані шифроблоки чи алгоритми.

4. Найбільшу швидкість генерації та простоту комбінаційних схем при апаратній реалізації можна досягти лише з використанням регістрів зсуву. Генератори на основі відносно складних операцій як зведення у ступінь або використання складних стандартизованих алгоритмів, не можуть бути реалізовані на простих комбінаційних схемах і потребують великої кількості обчислень. Такі генератори практично використовувати лише для генерації невеликих значень, таких як ключі або паролі.

5. Генератори, які використовують зсувні регістри та лінійні перетворення не забезпечують криптографічну стійкість. Функції, які забезпечують максимальний період, є рознесеними, і це ускладнює структуру комбінаційних схем, а також спрощує злам повідомлень за допомогою декомпозиції. Властивості генеруємої послідовності можуть бути покращені, якщо для зсувних регістрів використовувати нелінійні функції зворотного зв'язку. Таким чином не можна поєднати такі функції з максимальним періодом і забезпечити непердбачуваність, і таким чином спростити комбінаційні схеми, повисити швидкодію генератора і покращити властивості послідовності, які впливають на ефективність застосування такої послідовності при шифрування певних повідомлень.

6. Для покращення властивостей псевдовипадкової послідовності, важливим напрямом досліджень є використання нелінійних функцій зворотного зв'язку із забезпеченням максимально можливого періоду 2^n. Такі функції можуть бути реалізовані на двокаскадних комбінаційних схемах, які є доволі простими, і при цьому генерація відбувається набагато швидше за рахунок того, що сигнал проходить лише через два каскади елементів. Проблема використання таких функцій полягає у тому, що синтез таких функцій є доволі складною задачею. Необхідно розробити нові методи їх синтезу для того, щоб на їх основі побудувати ефективні генератори псевдовипадкових послідовностей. Більш того

ефективність можна підвищити, якщо використовувати двокаскадну комбінаційну схему з можливістю її переналаштування, і таким чином забезпечити можливість зміни нелінійної функції зворотного зв'язку.

РОЗДІЛ 2
ДОСЛІДЖЕННЯ НЕЛІНІЙНИХ ФУНКЦІЙ ЗВОРОТНОГО ЗВ'ЯЗКУ ДЛЯ ЗСУВНОГО РЕГІСТРУ

2.1 Дослідження характеристик функцій зворотного звязку

Криптографічні генератори псевдовипадкових двійкових послідовностей були та залишаються важливою частиною захисту інформації, в частності для вбудованих систем, які передають потокову інформацію. Клод Шеннон, який відомий своїми дослідженнями у області теорії інформації та криптографії, зазначав, що такі генератори є основним засобом шифрування інформації як при передачі, так і при зберіганні [56]. Також, він показав, що можна досягти абсолютної криптографічної стійкості, якщо використовувати ключ такої самої довжини, як і повідомлення.

Генератори псевдовипадкових послідовностей насправді використовуються у широкому класі алгоритмів шифрування, у деяких випадках генератор присутній у алгоритмі неявно. Вектор ініціалізації для таких генераторів задається користувачем перед початком генерації. У контексті шифрування даний вектор частіше називають ключем.

Сучасні генератори псевдовипадкових послідовностей, які плануються використовувати у криптографічних алгоритмах, будуються на основі трьох принципів:

- Розбіжність. Оскільки стани генераторів є обмеженими, то всі генератори псевдовипадкових послідовностей мають деякий період, після якого генератор повертається у початковий стан і знову генерує ту саму послідовність. Розбіжність означає, що даний період повинен бути якомога більше і наближатись до значення 2^N-1, де N – кількість бітів, які описують стан генератора.

- Нелінійність. Сучасний генератор має обов'язково мати деякі нелінійні операції. Таким чином забезпечується неможливість проведення лінійного аналізу. Такий аналіз є простим і проводиться одним із перших при тестуванні криптографічних алгоритмів.
- Обчислювальна складність. Даний принцип полягає у тому, що даний алгоритм не спрощується до алгоритму з меншою кількістю обчислень. Таким чином, по-перше криптографічна стійкість алгоритму не зводиться до криптографічної стійкості більш простого алгоритму (які часто є менш криптографічно стійкими). А по-друге, зловмисник не зможе використовувати більш простий алгоритм для пришвидшення аналізу.

Для перетворень на двійковими послідовностями, проблема розбіжності (тобто забезпечення періоду 2^N-1) вирішена лише для лінійних перетворень. Дані перетворення використовуються у LFSR, але як вже було вказано раніше, використання лише лінійних перетворень не є криптографічно стійким способом для генерації послідовностей.

Для нелінійних перетворень, питання забезпечення розбіжності все ще залишається відкритим, і наразі знайдено лише деякі з таких перетворень [7-10].

Як вже було визначено у попередньому розділі, у роботі було визначено три критерії ефективності генератора: швидкодія, простота та криптографічна стійкість.

Сам критерій криптографічної стійкості можна розбити на декілька критеріїв:

- статистичний аналіз згенерованої послідовності має відображати рівномірний розподіл;
- період генератора має бути достатньо великим, і наближатись до 2^N-1, тобто повинен виконуватись принцип розбіжності;

- псевдовипадкова послідовність має складатись з незалежних бітів, тобто із частини послідовності не можна бути відновити іншу частину не знаючи стану поточного стану генератору; тобто маючи таку послідовність, її буде неможливо відрізнити від дійсно випадкової послідовності.

Важливо зазначити, що псевдовипадковість згенерованої послідовності оцінюється за цими критеріями у контексті задачі, яка вирішується. Тобто саме клас задач, які вирішуються, вказує який повинен бути розподіл, допустимі відхилення від даних критеріїв, тощо.

Третій критерій серед зазначених критеріїв криптографічної стійкості є найбільш важливим для криптографічних генераторів послідовностей. Він забезпечує непередбачуваність згенерованої послідовності у випадку, коли зловмиснику відома частина послідовності (яку він міг отримати різними способами, наприклад, знаючи частину повідомлення).

Також, знаючи статистичні ймовірності повідомлення та псевдовипадкової послідовності, зловмисник може провести статистичний аналіз маючи лише шифротекст та відновити оригінальне повідомлення.

По суті, два перші критерії (статистичні характеристики послідовності та розбіжність послідовності) можна розглядати, як підпункти третього критерію (незалежність бітів).

Для першого випадку, це пов'язано з тим, що якщо статистичні властивості послідовності не задовольняють поставленої задачі, то можна знайти статистичні залежності між бітами.

У другому випадку, малий період означає, що ключ почне повторюватись через малий проміжок часу. Тобто знаючи частину послідовності, яка відповідає одному періоду, можна легко відновити наступні біти, адже вони повністю співпадають з попередніми.

Є два основних способи визначення точних критеріїв для оцінки непередбачуваності послідовності.

Перший спосіб полягає у наступному. Беремо деяку частину *K* псевдовипадкової послідовності довжиною *k*. Далі інтерполюємо та будуємо модель зсувного регістра з лінійним зворотнім зв'язком (LFSR). Для цього можна використати метод Берлекемпа-Мессі [6], який знаходить найменший LFSR, який відтворює задану послідовність. У результаті отримуємо деякий регістр довжиною *l*. Дане значення відображає лінійну складність послідовності, тобто *L(K) = l*. У випадку, якщо лінійна складність дорівнює половині довжини *k* послідовності (формула 2.1), то дану послідовність *K* не можна екстраполювати (тобто передбачити).

$$L(K) = \frac{k}{2}. \tag{2.1}$$

Також існує нелінійна складність *N(K)* [32,33], яка також використовуються для того, щоб оцінити наскільки послідовність є непередбачуваною. Для цього використовуються зсувні регістри з нелінійним зворотнім зв'язком. Аналогічно до оцінки лінійної складності, також будується зсувний регістр, але у цьому випадку для зворотного зв'язку використовуються нелінійні функції. Існують різні способи отримання таких регістрів [57,58]. Тоді нелінійна складність N(K) визначається двома значеннями – математичним очікуванням та середньоквадратичним відхиленням. Для непередбачуваних послідовностей, математичне очікування m можна обрахувати за формулою 2.2.

$$m = 2\ \log_2 p. \tag{2.2}$$

Тоді як середньоквадратичне відхилення σ обраховується за формулою 2.3.

$$\sigma = \frac{\log_2 p}{2}. \quad (2.3)$$

Проте оцінки лінійної L(K) та нелінійної N(K) складності послідовності не є достатніми показниками непередбачуваності псевдовипадкової послідовності, хоча непередбачувана послідовність обов'язково задовольняє ці критерії.

Ще однією проблемою даних оцінок є те, що вони використовують генератори LFSR та NLFSR, які відтворюють послідовність з абсолютним співпадінням бітів. Але можливі ситуації, коли послідовність K_ε, яка має невелике відхилення ε від оригінальної послідовності K, може мати значно нижчу складність $L(K_\varepsilon)$ та $N(K_\varepsilon)$, ніж оригінальна послідовність K. Тобто можливо передбачити послідовність з невеликим відхиленням ε.

Тому на сьогодні при аналізі передбачуваності послідовності за допомогою оцінки лінійної та нелінійної складності використовуються більш складні алгоритми, які враховують деяке відхилення ε.

Але проблема таких методик полягає у тому, що вони вимагають багато обчислювальних ресурсів, і тому їх важко використовувати для аналізу довгих послідовностей.

Нелінійна складність з відхиленням $N(K_\varepsilon)$ дуже сильно пов'язана з критерієм лавинного ефекту (strict avalanche criterion, SAC), який має задовольняти нелінійна функція зворотного зв'язку зсувного регістра NLFSR, за допомогою якого інтерполюється послідовність з відхиленням K_ε. Суть полягає у тому, що якщо взяти диференціал даної функції $f(x_1,\ldots,x_n)$ по будь-якій змінній x_k, то у результаті вийде функція $h(x_1,\ldots,x_{k-1},x_{k+1},\ldots,x_n)$ (формула 2.4) з рівною кількістю одиниць та нулів на виході. Дана умова повинна виконуватись для всіх значень k від 1 до n. Іншими словами, при зміні будь-якого одного біту аргументу функції $f(x_1,\ldots,x_n)$, ймовірність зміни вихідного біта дорівнює 0,5.

$$h(x_1, \ldots, x_{k-1}, x_{k+1}, \ldots, x_n) = f(x_1, \ldots, x_k, \ldots, x_n) \oplus f(x_1, \ldots, \overline{x_k}, \ldots, x_n). \quad (2.4)$$

Таким чином, якщо NLFSR задовольняє критерій SAC і ми зменшимо його на один біт, то отримане відхилення ε буде дорівнювати половині послідовності, адже половина вихідних бітів змінить своє значення. Виходить, що якщо спрощувати модель, то відхилення ε становиться занадто великим.

Зворотне твердження також вірне. А саме, якщо нелінійна складність послідовності N(K) не може бути знижена з деяким відхиленням ε, тобто якщо немає такої послідовності K_ε, щоб $N(K_\varepsilon)<N(K)$ з невеликим відхиленням ε, то функція нелінійного зворотного зв'язку $f(x_1,\ldots,x_n)$ обов'язково виконує критерій строгого лавинного ефекту.

Таким чином, оскільки нелінійна складність $N(K_\varepsilon)$ та критерій SAC функції $f(x_1,\ldots,x_n)$ тісно пов'язані, то при проектуванні криптографічних генераторів, функція зворотного зв'язку, для забезпечення криптографічної стійкості, обов'язково має виконувати критерій SAC.

Отже, перший спосіб оцінки непередбачуваності послідовності – це оцінка лінійної L(K) та нелінійної $N(K_\varepsilon)$ складності згенерованої послідовності.

Другий спосіб полягає у розпізнавані шаблонів у даній псевдовипадковій послідовності. Суть полягає у тому, що є два джерела послідовностей K_1 та K_2, і одне з джерел є випадковим. Обидва джерела мають деякі розподіли P_1 та P_2 на множині M можливих невеликих частин послідовностей. І є можливість отримувати послідовності K_1 та K_2 з цих джерел.

При достатньо довгих послідовностях, можна доволі близько оцінити розподіли для першого P_1 та другого P_2 джерел. Якщо послідовності дуже довгі, і таким чином кількість невеликих частин повідомлень дуже велика, то можна використати обмеження по часу або по довжині повідомлення. Також можливо використовувати алгоритми, які замість зберігання всіх простих частин, зберігають їх у деякому узагальненому вигляді.

Тоді алгоритм, який використовується для оцінки ймовірностей P_1 та P_2 та який на виході видає оцінку яка послідовність належить до якого джерела, може з деякою ймовірністю зробити свою оцінку. При цьому, чим більша довжина послідовності, яка оцінюється, тим більше простих частин і тим більша ймовірність, що алгоритм правильно розпізнав послідовність.

Якщо алгоритм на виході видасть нуль, то це означає, що алгоритм вважає що послідовність K_2 є випадковою. Тоді як у випадку, коли алгоритм видає один, то він вважає що послідовність K_1 є випадковою.

Такий алгоритм називають розпізнавачем, оскільки він розпізнає прості шаблони у послідовностях.

Розглянемо формальне визначення таких розпізнавачів, обмежених по часу.

Нехай є два джерела послідовностей K_1 та K_2, одне з яких – є випадковим. Ймовірність точності розпізнавача R обмежена по часу роботи алгоритму, якщо всі наступні пункти виконуються:

- якщо взяти деякий набір вихідних повідомлень $(k_1,k_2,\ldots,k_n)$ джерела K, то час завершення обчислення алгоритму пропорційний значенню n^t, а на виході алгоритму отримуємо деяке двійкове значення (одиницю або нуль) $R(k_1,k_2,\ldots,k_n)$;
- ймовірність того, що $R(K_1)=1$ відображає ймовірність того, що $R(k_1,k_2,\ldots,k_n)=1$ у випадку, коли набір повідомлень $(k_1,k_2,\ldots,k_n)$ був отриманий джерелом K_1;
- ймовірність того, що $R(K_2)=1$ відображає аналогічну ймовірність для $R(k_1,k_2,\ldots,k_n)=1$ у випадку, коли набір $(k_1,k_2,\ldots,k_n)$ був отриманий з джерела K_2;
- існує деяка нескінченна послідовність $n_1,n_2,n_3,\ldots$ така, що кожне наступне значення n_i є більшим попереднього, і при цьому виконується умова, яка наведена у формулі 2.5.

$$\left|P\left(R(K_1[1..n_i])\right) - P\left(R(K_2[1..n_i])\right)\right| > \varepsilon, \quad (2.5)$$

де P позначає ймовірність, R – розпізнавач, K_1 та K_2 – перше та друге джерела відповідно, $[1…n_i]$ – частина послідовності з даних джерел, ε – деяке додатне число, $n_1<n_2<n_3<….$

Еквівалентне визначення розпізнавача також може бути сформульовано з використанням комбінаційних схем. Формальне визначення двох послідовностей, які отримані з двох джерел, які не можна розрізнити, можна описати за допомогою розпізнавача:

Якщо для двох джерел K_1 та K_2 не можна створити розпізнавач у рамках заданих обмежень стосовно часу обчислень, об'єму використованої пам'яті та довжини послідовностей, то такі джерела вважаються такими, що їх не можна розрізнити маючи лише можливість отримувати з них послідовності.

Існує більш широкий клас генераторів, який охоплює у собі генератор псевдовипадкових двійкових послідовностей, а саме генератори рядків, або розрядні генератори.

Суть розрядного генератору полягає у тому, що він на вхід отримує деякий початковий вектор IV довжиною n. Далі він генерую деяку послідовність K, довжина якої є пропорційною до n^k. Позначимо довжину згенерованої послідовності m. Тоді дана довжина буде значно більшою за довжину початкового вектора m>n. Розрядний генератор по суті є деякою детермінованою функцією, яка за певний поліноміальний час t перетворює початковий вектор IV у довшу послідовність K.

Такий генератор вважають псевдовипадковим, якщо при достатньо довгому дійсно випадковому початковому вектору IV, він представляє собою таке джерело послідовності, яке генерує послідовність K, яку неможливо відрізнити від дійсно випадкової послідовності у рамках визначених ресурсів.

Тобто неможливо створити такий розпізнавач R у рамках заданих часу та пам'яті, який би зміг з достатньою ймовірністю знайти яка з послідовностей була згенерована розрядним генератором, а яка – дійсно випадковим джерелом.

Формально дану умову можна визначити за формулою 2.6, яка по суті є зворотною до формули 2.5.

$$\left|P\left(R(K)\right) - P\left(R(K')\right)\right| \leq \varepsilon, \tag{2.6}$$

де $P\left(R(K)\right)$ – ймовірність, що розпізнавач видасть на виході значення один у випадку, коли послідовність була отримана з псевдовипадкового генератора з деяким початковим вектором IV, який був отриманий дійсно випадковим чином; $P\left(R(K')\right)$ – ймовірність, що розпізнавач також видасть одиницю у випадку, коли послідовність такої самої довжини була отримана з деякого дійсно випадкового джерела; ε – деяке додатне значення, яке показує наскільки великим має бути відхиленням, щоб розпізнавач більше не вважався таким, що може розпізнати два джерела.

Іншими словами, це означає, що генератор вважається псевдовипадковим, якщо не існує моделі, складність якої поліноміліально залежить від довжини послідовності K, над якою виконується аналіз, і яка могла б точно розпізнати з якого джерела була отримана дана послідовність K – дійсно випадкового, чи згенерована за деяким детермінованим алгоритмом.

Отже, будь-який розпізнавач по своїй суті є деяким тестом на перевірку псевдовипадковості генераторів послідовностей, і враховуючи визначення, яке наведено вище, виходить, що генератор вважається псевдовипадковим, якщо він задовольняє тести, які можна виконати за певний час з певними ресурсами, навіть для великих послідовностей K.

Розрядний генератор задовольняє всі такі тести, якщо всі біти послідовності є непердбачуваними і також задовольняють тестам на наступний біт послідовності, який полягає в наступному.

Якщо відомий алгоритм псевдовипадкового генератора і перші n бітів згенерованої послідовності K[1…n] (але початковий вектор IV залишається невідомим, і є секретним ключем), і при цьому неможливо перебачити наступний розряд K[n+1] (точно, або з певною ймовірністю, відмінною від 0,5), то це означає, що генератор пройшов тест на наступний біт.

Більш того, якщо є деякий поліноміальний розрядний генератор, і якщо він задовольняє тест на наступний розряд, то його також неможливо відрізнити від дійсно випадкових чисел за допомогою деякого розпізнавача. Зворотне також є правдивим, тобто якщо для даного генератору неможливо отримати деякий розпізнавач, який би відрізняв його від джерела дійсно випадкових послідовностей, то такий генератор буде задовольняти тест на наступний біт.

Іншими словами, тест на відмінність від дійсно випадкових чисел та тест на наступний біт є еквівалентними.

У контексті генераторів послідовностей, неможливість перебачити наступний біт послідовності також є еквівалентним до того, щоб назвати даний генератор псевдовипадковим. Тобто неможливо створити розпізнавач, який би знаходив з якого джерела була отримана дана послідовність – з генератору чи з дійсно випадкового джерела.

Всі криптографічні генератори засновані на принципі необернених перетворень.

Тобто у основі генератора знаходиться деяка функція, яка відносно швидко може бути обчислена і використана для генерації псевдовипадкової послідовності. Але зворотне перетворення функції, тобто отримання вхідного значення функції, знаючи лише вихідне значення, є практично неможливим (тобто або є взагалі неможливим, або потребує ресурсів значно більше, ніж задача цього варта).

Наразі ще не доведено чи існують функції, які є абсолютно необернеными (тобто які навіть при наявності безлічі ресурсів, вхідні значення неможливо знайти знаючи вихідні), але для практичних цілей достатньо мати функцію, зворотне перетворення якої є практично неможливим при сучасному та найближчому майбутньому розвитку обчислювальних технологій.

Отже, найбільш важливою властивістю функції, яка реалізується генератором псевдовипадкових послідовностей, є неможливість виконати зворотне перетворення для даної функції.

Тобто функції, які використовуються у криптографічних генераторах, мають засновуватись на деяких математичних функціях або перетвореннях з якоїсь області математики, і для цих перетворень наразі не повинно існувати рішення для отримання зворотного перетворення.

Одним з таких перетворень є знаходження коренів для системи рівнянь над булевими змінними. Дане перетворення не є абсолютно незворотнім, оскільки його можна виконати повним перебором. При чому для деяких систем даний перебір може бути зменшений з використанням апроксимації. Тому для забезпечення незворотності перетворення, необхідно знайти такі системи, які були б достатньо великими і які не можна було б апроксимувати. Таким чином при достатньому розмірі системи, повний перебір буде неможливим з певним обмеженням ресурсів та часу.

Для оцінки нелінійності функцій, використовується Хемінгова відстань до найближчої лінійної функції. Суть Хемінгової відстані полягає у тому, що для заданої нелінійної функції $f(x_1,\ldots,x_n)$ знаходиться найближча функція $l(x_1,\ldots,x_n)$, яка є лінійною. Для обрахунку відстані між двома функціями, обраховується кількість бітів, які відрізняються на виході. Дана кількість і є відстанню. Таким чином, якщо порівняти дану функцію з усіма лінійними функціями і обрати ту,

де відстань буде найменшою, то це і буде Хемінговою відстанню до лінійної функції (формула 2.7).

$$h(f) = \min_{l \in L} D(f(x_1, \dots, x_n), l(x_1, \dots, x_n)), \tag{2.7}$$

де f – функція, для якої виконується пошук відстані, h – відстань до найближчої лінійної функції, L – множина всіх лінійних функцій для n аргументів, D – Хемінгова відстань між парою будь-яких заданих функцій, яка обчислюється за формулою 2.8.

$$D\big(f(x_1, \dots, x_n), l(x_1, \dots, x_n)\big) = \sum_{x_i \in \{0,1\}, i=\overline{1,n}} f(x_1, \dots, x_n) \oplus l(x_1, \dots, x_n) \tag{2.8}$$

Де знаходиться різниця між значеннями двох функцій при всіх можливих значеннях аргументів x_i (операція XOR видасть нуль якщо значення співпадають, та один, якщо значення функцій різні) та рахується сума всіх різниць. Таким чином отримується кількість бітів, які відрізняються між двома функціями.

Якщо булева функція $f(x_1, \dots, x_n)$ є балансною і кількість аргументів більша трьох, то нелінійність даної функції за Хемінговою відстанню $h(f)$ не перевищує значення певного значення. При парній кількості аргументів, дане значення обраховується за формулою 2.9.

$$h(f) \leq 2^{n-1} - 2^{\frac{n}{2}-1} - 2. \tag{2.9}$$

Тоді як при непарній кількості аргументів – за формулою 2.10.

$$h(f) \leq \left\lfloor 2^{n-1} - 2^{\frac{n}{2}-1} \right\rfloor. \tag{2.10}$$

Отже, для того, щоб генератор псевдовипадкових послідовностей вважався криптографічно стійким, він повинен використовувати незворотну функцію, яка задовольняє критерій лавинного ефекту і яка має мати якомога більшу $h(f)$ (Хемінгову відстань до лінійної функції).

2.2 Виявлення загальних властивостей нелінійних функцій зворотного зв'язку для NLFSR які забезпечують максимальний період повторення послідовності

Під час дослідження існуючих типів генераторів псевдовипадкових послідовностей та прикладів їх реалізації, було виявлено, що найбільш ефективним типом генераторів у контексті вбудованих систем є NLFSR. Такі генератори є доволі простими (один регістр з невеликою кількістю елементарних нелінійних функцій) та швидкими (швидкодія дуже пов'язана з простотою генератора). Також було виявлено, що при правильно підібраній функції достатньо великого розміру (тобто такій, зворотне перетворення якої є практично неможливим при сучасних обчислювальних можливостях), вона буде забезпечувати криптографічну стійкість генератора, адже тоді буде неможливо відрізнити згенеровану послідовність від дійсно випадкової без знання внутрішнього стану генератора.

Але частка функцій, які забезпечують максимальний період, дуже мала у порівнянні з усіма можливими булевими функціями. При чому при збільшені розміру n регістру, дана частка стрімко зменшується. Це призводить до того, що пошук нелінійних функцій зворотного зв'язку, які б забезпечували критерій лавинного ефекту і забезпечували б максимальний період, є доволі складною задачею, і на сьогодні синтез таких функцій досі не є повністю вирішеною проблемою.

Найбільш очевидний спосіб пошуку – виконання повного перебору всіх можливих функцій заданого розміру. Але проблема полягає у тому, що при збільшені розміру регістра на один біт, кількість всіх можливих функцій збільшується вдвічі, перевірка кожної з таких функцій збільшується у 2^n рази (оскільки збільшується теоретично можливий максимальний період), а частка

функцій, які задовольняють умову, значно менша (хоча абсолютне значення кількості таких функцій збільшується).

На сьогодні за допомогою повного перебору було знайдено функції нелінійного зворотного зв'язку для регістрів розміром до 25 біт включно, та для 27 біт [57,58].

Регістр із 27 біт може забезпечити максимальний період у 2^{27}, що є доволі малим, оскільки це дорівнює всього 16 МБ, тому відомі на сьогодні нелінійні функції не можуть використовуватись у практичних задачах.

Також був запропонований еволюційний метод для отримання таких функцій з використанням перетворень над лінійними функціями. Але він є доволі обмеженим і видає лише малу частину можливих функцій [10]. Також є великий недолік даного способу – він все ще потребує великої кількості ресурсів та часу на обрахунок.

Існує ще один спосіб отримання обмеженого класу нелінійних функцій зворотного зв'язку, який заснований на послідовностях де Брейна [8]. Даний метод використовує кодові цикли послідовностей де Брейна (тобто послідовності, які були отримані за допомогою циклічних зсувів). Дані послідовності ітеративно об'єднуються для утворення нових більш складних послідовностей.

Серед недостатків даного способу можна виділити те, що утворені функції є обмеженими, тобто враховуються далеко не всі можливі функції з максимальним періодом. Час на обрахунок також є доволі високим, що не дає можливості ефективного синтезу даних функцій.

Для того, щоб отримати ефективний псевдовипадковий генератор на основі нелінійних функцій, було виконано аналіз властивостей відомих функцій невеликого розміру для того, щоб знайти закономірності, які б допомогли отримати функції більшого розміру.

Для кращого розуміння знайдених властивостей, на рисунку 2.1 наведено загальну схему NLFSR.

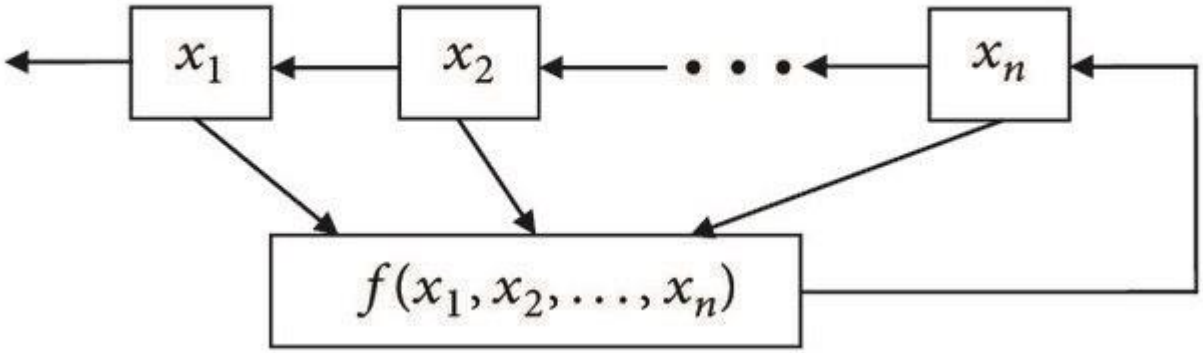

Рис.2.1. Загальна схема NLFSR

Сформулюємо першу властивість. Усі нелінійні булеві функції $f(x_1,\ldots,x_n)$ зворотного зв'язку, які забезпечують максимально можливий період, а саме 2^n, змінюють своє значення при зміні першого аргументу x_1. Формально дана властивість описується за формулою 2.11.

$$f(x_1, x_2, \ldots, x_n) = \overline{f(\overline{x_1}, x_2, \ldots, x_n)}. \quad (2.11)$$

Тобто у алгебраїчній формі функція може бути представлена з використанням операції НІ над функцією та першим аргументом.

Інтуїтивно дану властивість можна зрозуміти наступним чином – якщо перший аргумент не впливає на вихід функції, то рисунок 2.1 змінюється таким чином, як зображено на рисунку 2.2.

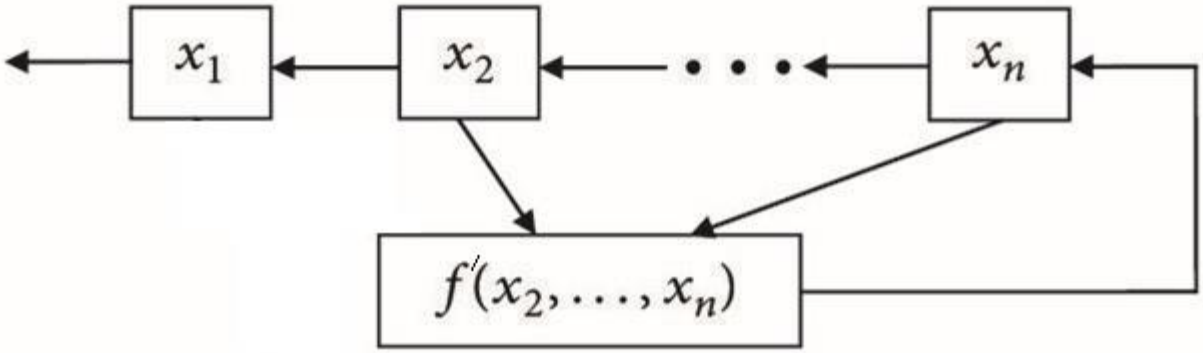

Рис.2.2. Генератор NLFSR, у якому перший аргумент не впливає на значення функції

Де f ' (x_2,…,x_n) по суті є тією самою функцією f(x_1,…,x_n), оскільки перший аргумент не впливає на її значення. З даної схеми можна виділити NLFSR, розмір якого є на один розряд меншим. При чому перший розряд x_1 тепер виступає лише буфером, який затримує вихідну послідовність на один такт, але при цьому не змінюючи її значення.

Тобто, у випадку, коли перший розряд не впливає на роботу генератора (окрім затримки), то даний генератор NLFSR розміром n можна звести до NLFSR розміром n-1.

Таким чином, тепер теоретичний максимальний період послідовності обмежений меншим NLFSR і не буде перевищувати значення 2^{n-1}. Тобто у такому випадку, період 2^n є неможливим, бо даний генератор спрощується до більш простого генератора.

Тепер, коли інтуїтивно зрозуміло, чому перший розряд повинен обов'язково впливати на значення функції, приведемо формальний доказ вищенаведеної властивості.

Для того, щоб забезпечити максимальний період 2^n згенерованої послідовності, генератор має пройти всі можливі стани, кількість яких також становить 2^n. Тобто для кожного стану існує лише один можливий попередній стан та лише один можливий наступний стан. Друге твердження щодо наступного стану означає, що функція, яка використовується, повинна бути однозначно визначена для всіх можливих станів регістру.

Також слід відзначити, що якщо за 2^n циклів генератора, він не повторив жодного стану (тобто всі стани, в яких побував генератор за 2^n цикли, були унікальними), то це означає що він побував у всіх можливих станах.

Тепер візьмемо деякий стан S_k генератора. На основі тверджень, вказаних вище, виходить, що для формування даного стану S_k із попереднього стану S_{k-1} відбувається або зсувом з заповненням з нулем, тобто ділення на два (формула 2.12),

$$S_{k\text{п}} = \frac{S_{k-1}}{2}. \tag{2.12}$$

Або зсувом з заповненням одиницею, тобто ділення на два з додавання старшого біту (формула 2.13).

$$S_{k\text{н}} = \frac{S_{k-1}}{2} + 2^{n-1}. \tag{2.13}$$

Для того, щоб значення S_k було парним, тобто щоб заповнення при зсуві використовувало нуль, значення функції від попереднього стану $f(S_{k-1})$ має бути рівним нулю. Тоді як для непарних S_k, відповідне значення $f(S_{k-1})$ дорівнює одиниці.

Оскільки кожен стан має унікальний попередній стан, то для забезпечення унікальності станів на протязі максимального періоду, необхідно, щоб функції для станів $\frac{S_{k-1}}{2}$ та $\frac{S_{k-1}}{2} + 2^{n-1}$ відрізнялись між собою, тобто щоб задовольнялась умова у формулі 2.14.

$$f\left(\frac{S_{k-1}}{2}\right) \neq f\left(\frac{S_{k-1}}{2} + 2^{n-1}\right). \tag{2.14}$$

Оскільки $\frac{S_{k-1}}{2}$ є меншим за 2^{n-1}, то додавання 2^{n-1}лише змінює старший біт x_1 на протилежний. Тобто формулу 2.13 можна звести до формули 2.15.

$$f(x_1, x_2, \dots, x_n) \neq f(\overline{x_1}, x_2, \dots, x_n). \tag{2.15}$$

Оскільки булеві функції мають лише два можливих значення, то знак "не дорівнює" можна замінити на знак "дорівнює" з оберненням значення виразу на одній із сторін рівняння. Тобто, формулу 2.16 можна звести далі до формули 2.15.

$$f(x_1, x_2, \dots, x_n) = \overline{f(\overline{x_1}, x_2, \dots, x_n)}. \tag{2.16}$$

У результаті отримана формула 2.15 є ідентичною до формули 2.10, яка вказана у першій властивості.

На схемі, обов'язковість впливу старшого біту можна показати так, як зображено на рисунку 2.3.

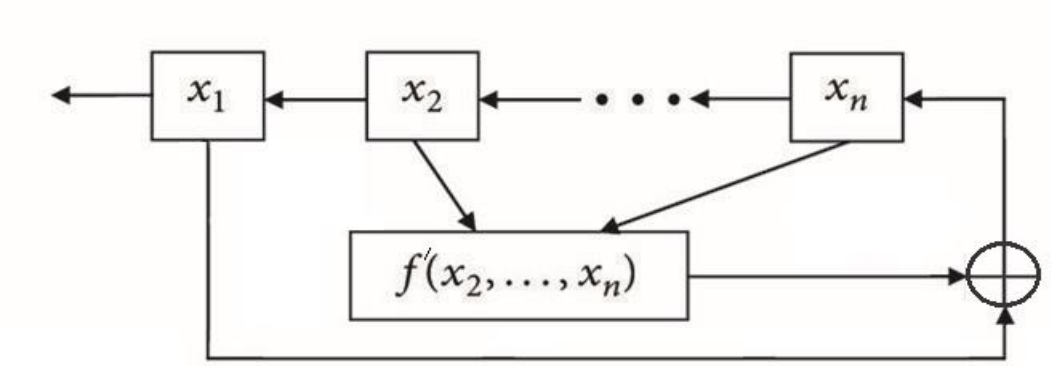

Рис.2.3. NLFSR з обов'язковим впливом першого біту

Таким чином, можна обмежити перебір до функції $f'(x_2, \ldots, x_n)$ меншого розміру n-1, і цим самим пришвидшити перебір вдвічі (оскільки половину функцій було виключено з перебору).

Це означає, що є лише один спосіб повторного отримання стану S_k, а саме, для цього необхідно, щоб спочатку повторно був отриманий стан S_{k-1}.

Візьмемо для прикладу всі функції для регістрів розміром n=4, які забезпечують максимальний період $2^n=2^4=16$. Знайти їх доволі легко повним перебором, оскільки перебрати необхідно всього 2^{16} функцій. Всього у результаті було знайдено 16 функцій.

Таблиця істинності для цих функцій наведена у таблиці 2.1. Реалізація перебору наведена у четвертому розділі.

Таблиця 2.1

Таблиця істинності для функцій від n=4 аргументів

S_k	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	1	1	0	1	1	1	1	1	1

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

З цієї таблиці можна побачити, що верхня половина таблиці, тобто $f_i(0, x_2, \dots, x_n)$, та нижня половина таблиці, тобто $f_i(1, x_2, \dots, x_n)$, мають повністю протилежні значення, а саме $f_i(0, x_2, \dots, x_n) = \overline{f_i(1, x_2, \dots, x_n)}$.

Виходить, що описана властивість дійсно виконується для нелінійних функцій зворотного зв'язку з максимальним періодом 2^n, що було доведено формально та наведено приклад на практиці для n=4.

Дану властивість також можна гарно побачити, якщо записати дані функції у формі алгебри Жегалкіна. У формулі 2.17 наведено декілька з цих функцій, проте описані властивості виконуються для усіх знайдених функцій.

$$f_1 = x_0x_1x_2 \vee x_1x_2 \vee x_1x_3 \vee x_2x_3 \vee x_0 \vee x_2 \vee x_3,$$
$$f_2 = x_0x_1x_2 \vee x_1x_3 \vee x_0 \vee x_2 \vee x_3,$$
$$f_3 = x_0x_1x_2 \vee x_1x_2 \vee x_0 \vee x_1 \vee x_3, \quad (2.17)$$

$$f_4 = x_0x_1x_2 \vee x_0 \vee x_3,$$

$$f_5 = x_0x_1x_2 \vee x_1x_2 \vee x_1x_3 \vee x_0 \vee x_3.$$

Можна помітити, що старший розряд x_0 входить лише до частини $x_0x_1x_2$ та як окрема частина, при чому вони включені до всіх нелінійних функцій, які задовольняють період. Всі інші терми у даних виразах не включають у себе старший розряд.

Також з даної таблиці можна побачити ще одну властивість – у випадку коли стан регістру дорівнює нулю $S_k=0$, то функція обов'язково має видавати на вихід одиницю. Це пов'язано з тим, що якщо б функція видавала нуль, то наступний стан регістру знову був би у нулі S_{k+1}, але тоді генератор не буде забезпечувати максимальний період.

Така сама ситуація і у випадку, коли всі біти регістру дорівнюють одиниці, тобто $S_k=2^n-1$. Тоді для того, щоб наступний стан S_{k+1} був відмінним від поточного стану S_k, необхідно щоб функція видавала нуль. Інакше регістр "застряне" у стані з усіма одиницями.

Також у таблиці 2.1 можна побачити схожу ситуацію для значень аргументів, які знаходяться посередині, тобто $2^{n-1}-1$ та 2^{n-1}.

У даному випадку причиною є те, що для забезпечення максимального періоду, генератор має побувати у всіх можливих станах, тобто має забезпечуватись можливість потрапляння у стани, де всі значення регістру є однаковими (всі нулі або всі одиниці).

Так, щоб потрапити у стан S_k рівним нулю, тобто S_k = (0,0,0,0), то попередній стан S_{k-1} повинен закінчуватись на (0,0,0). Тобто S_{k-1} = (0,0,0,0) або S_{k-1} = (1,0,0,0). Але щойно перед цим було показано, що для забезпечення максимального періоду, після стану (0,0,0,0) буде завжди стан (0,0,0,1). Таким чином, перший варіант є неможливим, і тому S_{k-1} має бути завжди рівним

(1,0,0,0). Тобто, якщо S_{k-1} дорівнює 2^{n-1}, то для потрапляння у стан зі всіма нулями, нелінійна функція має завжди видавати нуль.

Така сама ситуація і для $S_{k-1}=2^{n-1}-1$, або $S_{k-1}=(0,1,1,1)$. Аналогічно необхідно забезпечити можливість потрапляння у стан $S_k=(1,1,1,1)$, і для цього єдиний можливий попередній стан є саме $S_{k-1}=(0,1,1,1)$. Тому всі функції з максимальним періодом мають видавати на вихід одиницю у даному випадку.

Враховуючи знайдені властивості, які описані вище, можна стверджувати наступне. Для того, щоб знайти всі нелінійні функції, які надають генератору максимальний період, можна значно пришвидшити перебір таких функцій шляхом виключення тих, які не задовольняють даним властивостям і відповідно не надають максимального періоду.

Знайдемо наскільки швидше буде виконуватись перебір з врахуванням наведених властивостей.

Отже, для n аргументів кожна функція має визначити значення для всіх можливих 2^n наборів аргументів. Тоді кількість всіх можливих функцій становить 2^{2^n}. Відповідно, необхідно виконати перебір 2^{2^n} функцій і для кожної з них провести аналіз на забезпечення періоду 2^n.

Тепер врахуємо знайдені властивості функцій, які нам потрібні:

- з перебору необхідно виключити всі функції, які для першого та останнього наборів (тобто набори з всіма нулями та всіма одиницями) не повертають одиницю та нуль відповідно;
- також необхідно виключити всі функції, які для наборів посередині (тобто набори, де всі значення є однаковими, окрім старшого біту) не забезпечують перехід у перший або останній набір, тобто не забезпечується можливість перейти у стан з всіма однаковими бітами;
- оскільки друга половина усіх знайдених функцій є оберненим значенням першої половини, тобто вони прямо залежать один від одного, то достатньо

перебрати лише одну половину, а другу обрахувати на основі першої, таким чином з перебору виключається половина наборів аргументів.

Після всіх зазначених виключень, кількість наборів аргументів, які слід перебрати, значно зменшується. А саме, з усіх 2^n наборів достатньо перебрати лише набори від 1 до 2^{n-1}-2. Тоді загальна кількість функцій, які будуть розглядатись, буде дорівнювати $2^{2^{n-1}-2}$.

Хоча дане значення і досі є доволі великим, це краще, ніж повний перебір всіх 2^{2^n} функцій.

У формулі 2.18 наведено у скільки разів пришвидшився перебір після врахування знайдених властивостей.

$$\frac{2^{2^n}}{2^{2^{n-1}-2}} = 2^{2+2^{n-1}}. \tag{2.18}$$

Виділимо набори, які залишилися у окрему таблицю (ТУДУ).

Таблиця 2.2

Частина таблиці істинності функцій для n=4 для аргументів від 1 до 6

S_k	f_1	f_{14}	f_2	f_7	f_3	f_{12}	f_4	f_{10}	f_5	f_6	f_8	f_{16}	f_9	f_{11}	f_{13}	f_{15}
1	0	1	0	0	0	1	0	1	0	0	0	1	1	1	1	1
2	0	1	0	1	1	1	1	0	1	1	1	1	0	1	1	1
3	0	1	1	1	0	0	0	1	0	1	1	1	0	0	0	1
4	1	0	1	1	0	0	1	0	1	0	1	1	0	0	1	0
5	1	0	1	0	1	1	0	1	1	1	1	1	1	0	1	1
6	1	0	0	0	1	0	1	0	0	0	1	0	1	1	1	1

У даній таблиці функції були переставлені у іншому порядку. Це зроблено для наглядності при поясненні однієї з наступних властивостей, тому поки що проігноруємо порядок функцій у таблиці.

Наступна властивість, яка була помічена у всіх знайдених функціях – це той факт, що кількість одиниць та кількість нулів як для виділеної частини функції для наборів від 1 до 2^{n-2} (таблиця 2.2), завжди дорівнюють непарним значенням. Наприклад, функції від f_1 до f_7 мають по три нулі і три одиниці, а функція f_8 має п'ять одиниць і один нуль.

При аналізі функцій більшого розміру, було підтверджено дану властивість стосовно непарності.

При цьому також помітно ще одну властивість – для парних значень n, обрана частина таблиці частіше має більше одиниць, тоді як для непарних n – більше нулів. Але це не є правилом, тому дану властивість не можна використати для пришвидшення перебору.

Ще однією властивістю шуканих функцій є те, що для кожної нелінійної функції з повним періодом, існує симетрична функція, яка також має повний період. Саме для цього, функції були згруповані у пари у таблиці 2.2. Точніше симетричними є не повністю вся таблиця істинності, а саме виділена частина, яка була винесена у дану таблицю.

Наприклад, функцію f_1, яка забезпечує повний період, а точніше її частину, можна записати у вигляді 000111. Тоді існує деяка функція, яка аналогічно може бути записана у вигляді 111000, і вона також буде надавати максимальний період. І дійсно, такою функцією є f_{14}. Дана теорія була перевірена для функцій різного розміру, і вона дійсно завжди виконується.

Таким чином, можна відкинути ще половину функцій, які залишились, і розглянути лише іншу половину. Результат перевірки функції буде також відображати результат для відповідної відкинутої симетричної функції, саме тому її і можна пропустити.

Повернемося до таблиці 2.1, а саме до рядків $S_k=5$ та $S_k=10$. При детальному розгляді, можна помітити, що якщо $f_i(5)=0$, то та сама функція також має нуль при $S_k=10$, тобто $f_i(10)=0$.

Схожа ситуація також при $f_i(10)=1$. А саме, якщо для деякої функції виконується умова $f_i(10)=1$, то дана функція також має одиницю коли аргумент дорівнює $S_k=5$, тобто $f_i(5)=1$.

Розглянемо причину даних властивостей. Якщо записати стани регістра у даних випадках у двійковому вигляді, то отримаємо $5_{10}=0101_2$ та $10_{10}=1010_2$. Припустимо, що регістр знаходиться у стані $S_k=0101_2$, а значення функції від даного стану дорівнює нулю $f(0101_2)=0$. Тоді при зсуві регістра на вхід подається нуль, і наступний стан виходить рівним $S_{k+1}=1010_2=10_{10}$. Тепер припустимо, що є деяка функція з повним періодом, яка при даному стані видає одиницю, тобто $f(1010_2)=1$. У такому випадку наступний стан дорівнює $S_{k+2}=0101_2$. Але тоді виходить, що стан регістру "застряє" у цих двох станах, і він не може забезпечити повний період. Тому при $f(0101_2)=0$, дана функція не може мати значення $f(1010_2)\neq1$, і відповідно воно має дорівнювати нулю.

Аналогічна ситуація для $S_k = 1010_2$ та $f(1010_2) = 1$. Провівши такий самий ланцюг з можливими наступними станами, у результаті отримаємо, що при $f(1010_2) = 1$, для забезпечення періоду 2^n, значення $f(0101_2) \neq 0$, і тому воно має дорівнювати одиниці.

Така сама умова була перевірена і при інших значеннях n, і вона також виконується для них.

Але оскільки ми розглядаємо лише верхню половину таблиці істинності, то дану властивість слід відобразити на цю половину, враховуючи те, що дані половини є повністю протилежними.

Для набору аргументів $10_{10}=1010_2$ відповідає набір з протилежним першим аргументом, тобто $0010_2=2_{10}$. При цьому, значення функцій на цих наборах є

протилежними, тобто $f(2) = \overline{f(10)}$. Таким чином можна описати зв'язок між наборами 2 та 5 через набір 10 за формулами 2.19.

$$f_i(5) = 0 \Rightarrow f_i(10) = 0 \Rightarrow f_i(2) = \overline{f_i(10)} = 1,$$
$$f_i(2) = 0 \Rightarrow f_i(10) = \overline{f_i(2)} = 1 \Rightarrow f_i(5) = 1. \quad (2.19)$$

Тепер, встановивши зв'язок між наборами 2_{10} та 5_{10}, повернемося знову до таблиці 2.2.

Знайдені зв'язки, які наведені у формулі 2.19, можна виразити іншими словами. Якщо функція забезпечує максимальний період, то вона обов'язково виконує наступну умову:

- якщо хоча б на одному серед наборів S_k=(010101…) або S_k= (001010…) функція f(S_k) приймає значення нуль, то вона приймає значення один на іншому наборі з цих двох.

Тобто функція не може одночасно приймати значення нуль на обох з цих наборів, інакше вона не дасть максимальний період.

Ще одна властивість пов'язана з даною – якщо функція f з повним періодом має різні значення на наборах S_k=(010101…) та S_k= (001010…), то існує функція з відмінністю лише на цих наборах (якщо розглядати лише обрану верхню частину, оскільки є також ще два набори, які прямо залежать від даних), яка також має повний період.

Прикладом таких функцій є функції f_1 та f_4, які майже повністю співпадають, окрім наборів 2, 5, та відповідним їм наборів 13 та 10.

Обидві з цих властивостей також були перевірені для інших значень n і вони також підтвердилися.

Знову повернемося до повних функцій та для кращої наглядності розглянемо функції для n=3. Функцій такого розміру, які задовольняють умову повного періоду, є всього дві, і вони наведені у таблиці 2.3.

Таблиця 2.3

Таблиця істинності функцій повного періоду для n=3

S_k	f_1	f_2
0	1	1
1	0	1
2	1	0
3	1	1
4	0	0
5	1	0
6	0	1
7	0	0

Тепер представимо дані функції у вигляді рядка з нулів та одиниць, починаючи з першого набору до останнього сьомого набору. До речі, саме таким чином функції зберігаються у пам'яті у програмному забезпеченні, яке було розроблене для даної роботи.

Отже, функція f_1 тоді має вигляд $f_1=10110100_2$, а функція f2 має вигляд $f_2=11010010_2$. Тепер зведемо дані значення до десяткового:

- $f_1=10110100_2=180_{10}$;
- $f_2=11010010_2=210_{10}$.

Можна помітити, що дані значення мають найбільший спільний дільник (НСД), який дорівнює 30.

Спробуємо також знайти найбільший спільний дільник для функцій з збільшим значенням n. Для n=4:

- $f_1= 1000111101110000_2=36720_{10}$
- $f_2= 1001110101100010_2=40290_{10}$;
- $f_3= 1010011101011000_2=42840_{10}$;
- $f_4= 1010101101010100_2=43860_{10}$;
- $f_5= 1010110101010010_2=44370_{10}$;

- f_6= 1011010101001010_2=46410_{10};
- f_7= 1011100101000110_2=47430_{10};
- f_8= 1011111101000000_2=48960_{10};
- f_9= 1100011100111000_2=51000_{10};
- f_{10}= 1101010100101010_2=54570_{10};
- f_{11}= 1110001100011100_2=58140_{10};
- f_{12}= 1110010100011010_2=58650_{10};
- f_{13}= 1110111100010000_2=61200_{10};
- f_{14}= 1111000100001110_2=61710_{10};
- f_{15}= 1111011100001000_2=63240_{10};
- f_{16}= 1111110100000010_2=64770_{10}.

Обраховуючи НСД, отримуємо значення 510.

При подальшому розгляді найбільшого спільного дільника для більших значень n, можна помітити, що до кожного наступного найбільшого спільного дільника домножується деяке просте число.

Позначимо найбільший спільник дільник для n символом M_n. Тоді при подальшому розгляді НСД, було помічено, що для всіх n виконується умова, наведена у формулі 2.20.

$$M_n = 2^{2^{n-1}+1} - 2. \quad (2.20)$$

Тобто виходить, що усі функції з повним періодом, при перетворені їх таблиці істинності у число, мають обов'язково ділитись на значення M_n.

Оскільки обрахунок значення M_n є дуже простим (формула 2.24), а функції у пам'яті вже зберігаються у вигляді чисел, то перевірка функції на те, чи задовольняє вона дану властивість, є дуже простою.

Таким чином можна набагато пришвидшити перебір, попередньо перевіривши чи задовольняє поточна функція даній властивості. Дана перевірка є набагато швидшою, аніж перевірка на повний період.

Виходить, що можна пропустити $\frac{M_n-1}{M_n}$ частку функцій і перевіряти лише $\frac{1}{M_n}$ частку функцій. При чому, значення M_n швидко зростає при збільшенні значення n, тобто перебір значно зменшується у $M_n = 2^{2^{n-1}+1} - 2$ рази.

Приклади значень M_n наведено нижче:

- $M_3 = 2^{2^{3-1}+1} - 2 = 30$
- $M_4 = 2^{2^{4-1}+1} - 2 = 510$
- $M_5 = 2^{2^{5-1}+1} - 2 = 131070$
- $M_6 = 2^{2^{6-1}+1} - 2 = 8589934590$
- $M_7 = 2^{2^{7-1}+1} - 2 = 36893488147419103230$
- $M_8 = 2^{2^{8-1}+1} - 2 = 680564733841876926926749214863536422910 \approx$
  $\approx 6.8 * 10^{36}$

Ще одним важливим моментом при розробці алгоритму для синтезу нелінійних функцій зворотного зв'язку для NLFSR є дослідження комбінаторних властивостей наборів аргументів, тобто станів регістру, у яких він перебуває під час генерації псевдовипадкової послідовності.

Метою дослідження комбінаторних властивостей полягає в тому, щоб знайти залежність від кількості одиниць або нулів.

Припустимо, що є деяка множина D_n для регістру розміром n, яка складається з дільників $d_{n,i}$ числа n (формула 2.21).

$$D_n = \{d_{n,1}, d_{n,2}, d_{n,3}, \dots, d_{n,k}\}, \quad (2.21)$$

де k – кількість дільників, n – розмір регістру, $d_{n,i}$ – дільник числа n, D_n – множина всіх дільників числа n.

Тоді існують такі стани регістрів, при яких періоди отриманих NLFSR мають деяку довжину m, значення якої є одним із дільників n. Також будуть існувати періоди довжиною n (оскільки число n також є дільником n), та періоди

довжиною один (у випадках, коли стан регістру "застряє" у всіх однакових значеннях (тобто всі нулі або всі одиниці).

Покажемо цю властивість.

Нехай деяке число d є одним із дільників числа n. У такому випадку регістр можна умовно поділити на n/d частин, і у кожну частину записати однакові значення S довжиною d.

Після виконання d тактів роботи алгоритму, тобто після зсуву регістру на d значень, існує така функція, яка б заповнила регістр при зсуві значенням S. Тобто регістр, через d тактів повернувся до початкового стану.

Тоді виходить, що після d бітів згенерованої послідовності, регістр, знаходячись знову в тому самому стані, буде генерувати таку саму послідовність. Тобто виходить, що період такого генератора NLFSR буде дорівнювати значенню d, яке є дільником числа n.

При чому кількість таких станів визначається кількістю всіх можливих значень частини регістру довжиною d, а саме за формулою 2.22.

$$b = 2^d. \tag{2.22}$$

Виходить, що для того, щоб зменшити перебір функцій, виключивши ті, які мають значно менший період, а саме кратний до одного з дільників числа n, достатньо прибрати їх.

Кількість таких функцій є сумою кількостей функцій для кожного можливого дільника n (формула 2.23).

$$H = \sum_{d_i \in D_n} B(d_i), \tag{2.23}$$

де D_n – множина всіх дільників числа n, d_i – елемент даної множини, B – загальна кількість функцій з періодом d_i, але без тих значень, які є дільниками числа d_i, оскільки вони вже враховані (тому що якщо деяке число є дільником

числа d_i, то воно також є дільником числа n, і відповідно було вже окрема враховано у множині дільників D_n.

Для того, щоб обрахувати значення $B(d_i)$, тобто щоб виключити повторення за рахунок менших дільників числа d_i, використовуються розрахунки, наведені у формулі 2.24.

$$\begin{cases} B(d_i) = \dfrac{\left(2^n - 2 - \sum_{j=\overline{1,i-1}} B(d_j)\right)}{n}, \text{при } d_i \geq 3 \\ B(d_i) = 1, \text{при } d_i = 2. \end{cases} \tag{2.24}$$

Повернемося до формули 2.23 та винесемо перший дільник, тоді отримаємо формулу 2.25.

$$H = B(2) + \sum_{d_i \in D_n / \{2\}} B(d_i). \tag{2.25}$$

Тоді, якщо об'єднати формули 2.24 та 2.25, то для обрахунку загальної кількості H функцій, період яких дорівнює одному з дільників числа n, тобто значення періоду яких належить до множини D_n, отримуємо формулу 2.26.

$$H = 1 + \sum_{d_i \in D_n / \{2\}} \frac{\left(2^n - 2 - \sum_{j=\overline{1,i-1}} B(d_j)\right)}{n}. \tag{2.26}$$

Таким чином, шляхом виключення функцій, які зациклюються на станах, які складаються з повторюваних частин, можна ще більше зменшити кількість функцій, які перебираються.

Розрахунок даної кількості для деяких значень n наведено нижче, де видно, що із збільшенням n, також очевидно збільшується значення H:

- при n=3, H=4
- при n=4, H=6
- при n=5, H=8
- при n=6, H=14
- при n=10, H=108

Хоча значення H і зростає із збільшенням n, воно все ще є набагато меншим за загальну кількість можливих функцій.

Оскільки ймовірність, що випадкова функція буде мати дану властивість, є доволі низькою, то перевірку даної властивості слід виконувати лише після перевірки попередніх, які є більш ймовірними.

Проте виконання даної перевірки все ще має сенс, оскільки вона простіша за повний прохід всіх станів регістру, а її результат, навіть якщо він не виключив функцію, все ще може бути використаний при повному проході, якщо зберегти відвідані стани (тобто можна пришвидшити перевірку, але при цьому збільшується об'єм необхідної пам'яті та ускладняється написання коду, який повинен врахувати збережені стани).

Ще одна важлива річ, яку слід врахувати при аналізі нелінійних функцій з повним циклом, це розподілення кількості одиниць та нулів у регістрі під час проходження через різні стани.

Для того, щоб обрахувати частку функцій з періодом довжиною n, і які мають рівно k одиниць у згенерованій послідовності довжиною n (відповідно кількість нулів у даній послідовності буде дорівнювати n-k), можна обрахувати за допомогою комбінаторних формул, а саме за допомогою формули для комбінацій. Для цього використовується формула 2.27.

$$G(n,k) = \frac{C_n^k}{n}, \tag{2.27}$$

де C_n^k розраховується за формулою 2.28.

$$C_n^k = \frac{n!}{k!\,(n-k)!}. \tag{2.28}$$

Наприклад, візьмемо період довжиною n=8, а кількість одиниць у даному періоді k=4. Тоді отримуємо наступні значення, які забезпечують повний період n=8:

- 11110000

- 11101000
- 11100100
- 11100010
- 11011000
- 11010100
- 11010010
- 11001010

Також отримуємо одну послідовність, період якої становить 4:

- 11001100

Та одну послідовність, період якої дорівнює 2:

- 10101010

Таким чином, загальна кількість значень, які задовольняють властивість стосовно періоду, який є дільником розміру регістра, дорівнює 70 (формула 2.29).

$$g = 8 * 8 + 4 * 1 + 2 + 1 = 70. \quad (2.29)$$

Перевіривши дане значення, отримаємо, що кількість дійсно дорівнює 70 (формула 2.30).

$$C_8^4 = \frac{8!}{4!\,(8-4)!} = 70. \quad (2.30)$$

Висновки до розділу 2

У другому розділі була поставлена мета дослідити нелінійні функції зворотного зв'язку, які б забезпечували максимальний період генератора на основі зсувного регістру.

У результаті проведеного дослідження, можна зробити наступні висновки:

1. Визначено критерії ефективних генераторів псевдовипадкових чисел, а саме їх простота, швидкодія та криптографічна стійкість. Криптостікість генератора передбачає рівномірний розподіл згенерованої послідовності, необерненість функції перетворення, непередбачуваність послідовності навіть якщо відомі наступні або попередні її частини. Генератор також має забезпечувати період якомога ближче до значення 2^n.

2. Виконано аналіз нелінійних функцій зворотного зв'язку, при яких генератор видає послідовність з періодом 2^n. Під час аналізу було встановлено ряд властивостей, які притаманні для всіх функції, які забезпечують повний період.

3. Було знайдено залежності у таблиці істинності таких функцій між певними наборами аргументів. А саме, було встановлено, що при наборах від 0 до 2^{n-1}-1 та при наборах від 2^{n-1} до 2^n-1 функції приймають повністю протилежні значення. Також було виявлено, що кожен з цих діапазонів наборів аргументів є симетричним відносно того самого діапазону. Додатково також було встановлено більш специфічні залежності, такі як між наборами (1,0,1,…) та (0,1,0,…).

4. Було виявлено, що для 4 наборів, усі функції, які забезпечують поставленим умовам, завжди мають певне значення. А саме – для першого набору, а саме (0,…,0,0), функція має завжди приймати значення 1, а на останньому наборі (1,…,1,1) – значення 0. Подібні властивості також притаманні для наборів (0,1,…1) та (1,0,…,0), на яких функція має приймати значення 0 та 1 відповідно.

5. Встановлено, що представлення таблиці істинності у вигляді числа завжди є кратним до певного значення M_n, яке можна розрахувати знаючи значення n.

РОЗДІЛ 3 МЕТОД ПОБУДОВИ ГЕНЕРАТОРА ПСЕВДОВИПАДКОВИХ ПОСЛІДОВНОСТЕЙ НА ОСНОВІ НЕЛІНІЙНИХ ФУНКЦІЙ ЗВОРОТНОГО ЗВ'ЯЗКУ

3.1 Розробка методу формування нелінійних функцій зворотного зв'язку для забезпечення повного періоду зсувного регістру

Запропонований метод для отримання булевих нелінійних функцій зворотного зв'язку заснований на знайдених властивостях функцій, які забезпечують повний період.

Для того, щоб забезпечити повний період, для кожного можливого стану S_n повинен існувати лише один можливий попередній стан S_{n-1}.

Тобто для усіх можливих значень станів S від 0 до 2^n-1, можна визначити унікальну пару станів (s_1,s_2), де значення s_1 розраховується за формулою 3.1.

$$s_1 = (2 * S) mod\ 2^n. \quad (3.1)$$

А значення s_2 за формулою 3.2.

$$s_2 = (2 * S) mod\ 2^n + 1. \quad (3.2)$$

Стани регістрів s_1 та s_2 також можуть набувати певне значення з множини всіх можливих станів від 0 до 2^n-1, і для кожного з таких станів існує своя пара можливих наступних значень, позначимо їх (s_1',s_1") та (s_2',s_2").

Значення станів s_1 та s_2 відрізняються всього лише одним старшим бітом. Враховуючи першу знайдену властивість, необхідно забезпечити, щоб виконувались наступні твердження стосовно цих станів: $f(s_1) \neq f(s_2)$, $f(s_1') \neq f(s_1'')$ та $f(s_2') \neq f(s_2'')$.

Стан регістру можна описати із множини його частин $D=\{d_1,\ldots,d_k\}$, де значення k дорівнює кількості таких частин. Кожній такій частині відповідає пара станів (s_1,s_2),

Запропонований метод полягає у наступному:

1. Визначити множину D із k=2 частин. Для кожної частини визначається пара (s_1,s_2). Позначимо їх (s_1',s_1") та (s_2',s_2"). При чому обов'язково мають виконуватись умови s_1'=(1,0,..,0,0), f(s_1')=0, s_1"=(0,0,..,0,1),f(s_1")=1, s_2'=(0,1,..,1,1), f(s_2')=1, s_2"=(1,1,..,1,0), f(s_2")=0.

2. Обрати одну із двох частин стану, тобто один елемент із множини D. Позначимо номер елементу hє{1,2}. Тоді для обраного елементу d_h, приймається як поточне значення. Після цього, для нього визначається відповідне йому значення стану s_h.

3. Виконати пошук такої частини d_g, щоб одним із відповідним йому значенням стану було значення s_h. Якщо знайдений фрагмент є таким, що його інший відповідний стан s_g=s_h, то перейти до пункту 5.

4. Якщо значення станів s_g та s_h не співпадають, то шукається інша частина d_{h+1} така, що її початковий та кінцевий стани співпадають і мають всього лише один стан у множині станів. Після знаходження такого елементу множини D, необхідно збільшити лічильник, який зберігає кількість знайдених елементів для поточного h.

5. Визначається пара станів s_1 та s_2, у які може перейти регістр після стану S, який зараз розглядається. Тобто s_1=((S*2) mod 2^n) та s_2=((S*2) mod 2^n)+1, де операція множення на 2 відповідає зсуву регістра вліво, при чому для s_1 заповнення відбувається нулем, а для s_2 – одиницею.

6. Якщо перше значення дорівнює s_1', визначеному у пункті 1, або друге значення дорівнює s_2', то перейти до пункту 9.

7. Якщо s_1 дорівнює s_1", або s_2 дорівнює s_2', то виконати перехід до пункту 10.

8. У іншому випадку, перехід можна виконувати до любого з пунктів 9 або 10.

9. Встановлюємо значення функцій для знайдених станів у відповідні значення $f(s_1)=1$, $f(s_2)=0$. При чому, якщо знайдене s_1 не відповідає жодному з початкових кодів, то необхідно встановити $s_1=s_1'$. Така сама дія виконується для другого стану, тобто $s_2=s_2'$ у випадку, коли s_2 не відповідає початковим станам. При цьому, фрагмент, який розглядався, замінюється на кінцевий фрагмент. Якщо початкове значення S не співпадає з жодним кінцевим значенням s_1 або s_2, то встановити s_1=S. Якщо S співпадає з початком фрагменту, то даний фрагмент видаляється.

10. Встановлюємо значення функції на протилежні до попереднього пункту, тобто $f(s_1)=0$ та $f(s_2)=1$. Якщо s_1 не дорівнює одному із початкових станів, то встановлюємо $s_1=s_1''$. Аналогічно для s_2, встановлюється значення $s_2=s_2''$. Якщо s_2 не співпадає з початковими станами, то даний фрагмент видаляється, а s_2 встановлюється у значення $s_2=s_2''$. Якщо співпадає, то видаляємо відповідний фрагмент.

11. Якщо після попередніх пунктів, кількість фрагментів, які залишилися, більше нуля, то збільшити значення h на одиницю (h:=h+1) та повернутись до пункту 2.

3.2 Оцінка ефективності запропонованого методу побудови генераторів псевдовипадкових послідовностей

Отже тепер, маючи знайдені властивості, можна значно пришвидшити перебір за рахунок виключення функцій.

У першу чергу слід зазначити яким чином функції представляються, оскільки для деяких властивостей це є важливим фактором для перевірки відповідності до даної властивості.

Один із способів – це зберігання функції у вигляді числа, яке у бінарному вигляді відповідає таблиці істинності для відповідної функції, яка зберігається та представляється у вигляді числа.

Даний спосіб є доволі зручним. Можна легко знайти необхідне значення функції від аргументу, застосувавши відповідну двійкову маску. Також даний спосіб дозволяє легко перевірити подільність даного значення на деяке число, що також є однією із властивістю, які були знайдені.

Але даний спосіб має наступну проблему – більшість мов програмування використовують розмір числа 32 чи 64 біти. Тобто таким чином можна представити лише функції максимум для n=6 аргументів.

Тому для дослідження функцій більшого розміру, які б забезпечували більший період, необхідно інший спосіб представлення функцій. Але слід зазначити, що тоді перевірка на кратність буде більш затратна.

Тому для даної задачі було обрано інший спосіб – таблиця істинності зберігається у вигляді рядка символів.

Оскільки рядок символів – це по суті масив з елементів однакової довжини, то доступ до окремого символу відбувається доволі швидко за його індексом (положенням у масиві).

Але тоді перевірка на ділення стає важкою задачею, і тому властивість стосовно НСД можливо краще пропустити.

Отже, перша властивість, яка була знайдена, полягає у тому, що старший біт регістру завжди впливає на значення функції, і тому перша та друга половина функції є повністю протилежними.

Дану властивість можна використати вже на етапі представлення функції у пам'яті, адже тоді замість зберігання повною функції, достатньо зберігати лише її половину. Для знаходження значень другої половини, достатньо взяти відповідне значення першої половини та інвертувати його.

Такий спосіб представлення функції (тобто зберігання лише її половини), також автоматично виключає всі функції, які не задовольняють першу властивість.

Таким чином, з перебору буде повністю виключено значну кількість функцій, а саме, замість 2^{2^n} функцій, після застосування першої властивості, отримаємо кількість функцій за формулою 3.3.

$$2^{\frac{2^n}{2}} = 2^{2^{n-1}}. \quad (3.3)$$

Також з перебору можна повністю виключити функції, перша половина яких починаються та/або закінчується на нуль.

Дане твердження випливає з другої знайденої властивості, яка передбачає, що для уникнення циклів одиничної довжини при значеннях станів 00..00, 11..11, 10..00 та 01.11, функція має приймати значення нуль при S=0 та $S=2^{n-1}-1$, і значення один при відповідних наборах $S=2^{n-1}$ та $S=2^n-1$.

Оскільки набори $S=2^{n-1}$ та $S=2^n-1$ є протилежними до S=0 та $S=2^{n-1}-1$, то вони вже були виключені з перебору за рахунок першої властивості. Тому для даної властивості важливі лише двоє з цих чотирьох наборів.

Таким чином кількість функцій, які перебираються, зменшується, і тепер обраховується за формулою 3.4.

$$\frac{2^{2^{n-1}}}{2^2} = 2^{2^{n-1}-2}. \quad (3.4)$$

Наступна властивість полягає у тому, що кількість одиниць та нулів у таблиці істинності функції є непарними. Оскільки половина усіх функцій буде мати парну кількість, то з перебору виключається ще половина функцій.

Тоді нова кількість функцій, які залишись, наведена у формулі 3.5.

$$\frac{2^{2^{n-1}-2}}{2} = 2^{2^{n-1}-3}. \quad (3.5)$$

Наступна знайдена властивість нелінійних функцій з повним періодом полягає у тому, що для кожної такої функції, існує симетрична їй функція, яка також забезпечує повний період.

Зворотне твердження також вірне. Тобто якщо функція не забезпечує повний період, то симетрична їй функція також не буде забезпечувати повний період. Це дійсно так, оскільки у іншому випадку (тобто коли друга функція забезпечує повний період), перша функція є також симетричною до другої, і тоді перша повинна також належати до множини підходящих функцій, що суперечило б початковому твердженню.

Таким чином, при розгляді будь-якої функції та визначенні чи задовольняє вона умові повного періоду, ми також будемо знати, що симетрична функція так само задовольняє чи не задовольняє нашим потребам.

Тобто нам необхідно перебрати лише ту частину функцій, які не є симетричними до вже перебраних функцій. Тобто всі функції можна розділити на дві рівні частини, при чому симетричні функції завжди будуть знаходитись у різних частинах.

Таким чином прибирається необхідність перевірки ще половини функцій, і нова кількість наведена у формулі 3.6.

$$\frac{2^{2^{n-1}-3}}{2} = 2^{2^{n-1}-4}. \tag{3.6}$$

Наступна знайдена властивість, а саме залежність між значеннями функції на наборах S=(010101…) та S= (001010…), полягає у тому, що функція не може приймати значення нуль на обох з цих наборів.

У таблиці 2.4 наведено комбінації можливих значень функції на цих наборах та чи задовольняє вона властивості.

Таблиця 2.4

Можливі комбінації значень f(010101…) та f(001010…)

f(010101…)	f(001010…)	Забезпечення повного періоду

0	0	не забезпечує
0	1	потенційно може забезпечувати
1	0	потенційно може забезпечувати
1	1	потенційно може забезпечувати

Тобто виходить, що четверту частину функцій також можна пропустити за рахунок функцій, які мають обидва нулі на цих наборах аргументів (тобто станів регістру генератора).

Виходить, що кількість функцій стає ще меншою, значення якої наведено у формулі 3.7.

$$2^{2^{n-1}-4} * \frac{3}{4} = 3 * \frac{2^{2^{n-1}-4}}{2^2} = 3 * 2^{2^{n-1}-6}. \quad (3.7)$$

Наступна властивість вказує на те, що якщо деяка функція з максимальним періодом має різні значення на вищевказаних наборах, то існує відповідна їй функція, яка відрізняється значеннями на цих наборах при незмінних значеннях на інших, така, що також має максимальний період.

Відповідно і вірне зворотне твердження про те, що якщо перша функція не має максимальний період, то і друга також його не має. Таким чином, з залишених трьох комбінацій у таблиці 2.4, можна виключити другу або третю комбінацію, адже вони залежать одна від одної.

Виходить, що можна виключити ще третину від функцій, які потенційно можуть задовольняти повному періоду (формула 3.8).

$$3 * 2^{2^{n-1}-6} * \frac{2}{3} = 2^{2^{n-1}-5}. \quad (3.8)$$

Наступним кроком запропонованого методу є перевірка функції на кратність до числа M_n, яке було описане у попередньому підрозділі.

Даний крок дозволяє виключити доволі велику кількість функцій та залишити всього лише частку 1/ M_n від потенційно підходящих функцій, де значення Mn розраховується за формулою 3.9.

$$M_n = 2^{2^{n-1}+1} - 2. \quad (3.9)$$

З врахуванням даної частки, нова кількість потенційних функцій, які забезпечать критерій розбіжності згенерованої послідовності (тобто забезпечення максимального періоду), наведена у формулі 3.10.

$$2^{2^{n-1}-5} * \frac{1}{M_n} = \frac{2^{2^{n-1}-5}}{2^{2^{n-1}+1} - 2}. \quad (3.10)$$

Даний крок запропонованого методу є доволі важливим, оскільки він виключає доволі велику кількість функцій з розгляду. Але проблема даного кроку полягає у тому, що виконання перевірки на дану властивість є доволі складною задачею, яка потребує окремого вирішення.

Тому при програмній реалізації даного методу, дана властивість не була врахована, і відповідно даний крок був пропущений з реалізації даного методу у програмному забезпеченні.

Ще одна властивість, яка була знайдена – це середня кількість одиниць та нулів у верхній та нижній половинах таблиці істинності функцій, які задовольняють критерій розбіжності.

А саме, було помічено, що для функцій від парної кількості аргументів (тобто коли довжина регістру n дорівнює парному значенню), верхня половина таблиці істинності у середньому має більше одиниць, а нижня половина, яка є повністю протилежною – має більше нулів.

Тоді як для функцій від непарної кількості аргументів, ситуація повністю протилежна – верхня половина таблиці має більше нулів, а нижня половина – більше одиниць.

Хоча дана властивість була помічена і дійсно виконується, використати її для пришвидшення перебору не зовсім можливо, оскільки вона є лиже статистичною. Тобто, хоча частіше вона виконується, ніж не виконується, все ще

існують функції, які навпаки мають протилежну ситуацію з розподілом значень у верхній та нижній половинах.

Тобто навіть знаючи дану властивість, все одно прийдеться перебирати всі функції, кількість яких була наведена у формулі 3.8 (або формулі 3.10 якщо врахувати кратність).

Хоча загальний час перебору не можна зменшити за допомогою цієї властивості, все ще є можливим зміна порядку перебору функцій таким чином, щоб спочатку перебирались ті функції, які відповідають даному розподілу.

Таким чином, ми збільшуємо ймовірність потрапляння нелінійної функції з повним періодом до тих функцій, які розглядаються раніше, і таким чином перша половина часу роботи алгоритму, який відповідає описаному методу, знайде більше підходящих нелінійних функцій, ніж друга половина роботи алгоритму. Ця властивість не пришвидшить перебір, а лише дасть можливість отримати частковий результат швидше.

Висновки до розділу 3

У третьому розділі була поставлена мета розробити метод побудови генератора псевдовипадкових чисел на основі зсувних регістрів розміром від 100 розрядів, використовуючи знайдені властивості функцій, які забезпечують максимальний період генератора.

У результаті проведеної роботи, було виконано наступні задачі:

1. На основі знайдених властивостей нелінійних функцій, при яких зсувний регістр проходить через максимальну кількість станів, було теоретично обґрунтовано та досліджено метод формування нелінійних функцій зворотного зв'язку, який відрізняється тим, що він враховує знайдені властивості функцій, які забезпечують період 2^n, і за рахунок цього досягається прискорення формування функцій при великій кількості n змінних.

2. Проведено оцінку ефективності запропонованого методу для побудови криптографічних генераторів. Встановлено, що розроблений метод значно пришвидшує процес формування нелінійних функцій, які забезпечують повний період, у порівнянні з повним перебором множини всіх можливих булевих функцій від n змінних.

РОЗДІЛ 4
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ МЕТОДУ ПОБУДОВИ ГЕНЕРАТОРА ПСЕВДОВИПАДКОВИХ ПОСЛІДОВНОСТЕЙ

4.1 Вимоги до програмного забезпечення

Оскільки для даного дослідження, програмне забезпечення потрібно у першу чергу для проведення аналізу та перебору з метою пошуку функцій, які далі будуть використовуватись, і саме програмне забезпечення не передбачає використанням даним забезпеченням іншими користувачами, то створювати графічний інтерфейс не має сенсу.

Основною задачею є допомога при дослідженні функцій зворотного зв'язку, і для достатньо реалізувати програмне забезпечення за допомогою інтерфейсу через командний рядок. Це включає у собі наступні підзадачі:

1. Повний перебір функцій відносно невеликого розміру для того, щоб знайти всі функції з повним періодом.
2. Пошук властивостей знайдених функцій, це включає у собі статистичні значення таблиці істинності функцій, її симетричність, парність, кратність деякому значенню, тощо.
3. Перебір функцій на основі методу, який був запропонований під час дослідження. Важливо забезпечити можливість перебору для великих значень довжини регістра n.
4. Реалізація генератору з використанням знайдених функцій, а також реалізація потокового шифрування з використання отриманих генераторів псевдовипадкових чисел.
5. Криптографічний аналіз згенерованої послідовності та шифротекстів на основі цих послідовностей.

Для забезпечення роботи програмного забезпечення без неочікуваних перебоїв, визначено мінімальні характеристики персонального комп'ютера, на якому буде працювати програма:

- одно-ядерний процесор з частотою 1 ГГц;
- 4 ГБ оперативної пам'яті;
- операційна система Windows 7 64-розрядна або більше, або операційна система на ядрі Linux 5.10 або більше;
- встановлений інтерпретатор мови програмування Python версії 3.9 або більше.

Для реалізації програмного забезпечення було використано мову програмування Python версії 3.9.

Дана мова є відносно простою для використання, і часто використовується дослідниками для створення невеликих скриптів під час математичних досліджень.

Оскільки дана робота передбачає саме створення скриптів для пошуку та аналізу нелінійних функцій зворотного зв'язку, то дана мова програмування добре підходить під поставленні задачі до програмного забезпечення для цієї роботи.

4.2 Організація структури даних

Враховуючи дослідження, яке було виконано у другому розділі, а саме на основі виявлених властивостей нелінійних функцій, які надають повний період, необхідно забезпечити представлення даних функцій у пам'яті таким чином, щоб було можливо:

- швидко дізнатись значення функції для заданого значення стану регістру;
- відносно легко виконати перевірку властивостей, які було досліджено у другому розділі;

- легко змінювати функцію на іншу таким чином, щоб можна було розглянути усі функції.

Для забезпечення останньої властивості, найкращим варіантом представлення функції є її таблиця істинності. Таким чином можна зберігати значення функції для усіх можливих аргументів від нуля до 2^n-1. Використовуючи такий спосіб представлення функції, можна легко отримати будь-яку функцію для заданого n підставивши відповідні значення у таблицю істинності. Це також дозволяє просто перебрати всі функції, підставляючи всі можливі комбінації нулів та одиниць у таблицю істинності.

Проблемою такого представлення може бути кількість пам'яті, яка необхідна для зберігання такої функції. Якщо кожне значення таблиці істинності зберігати у вигляді одного біту, то тоді загальний об'єм пам'яті для функції від n змінних буде дорівнювати 2^n біт.

Для отримання більш точних значень, візьмемо n=27. Наразі відомі нелінійні функції з повним періодом лише для аргументів до 27 біт, саме тому було обрано дане значення. Дана оцінка є приблизною, оскільки досліджуватись будуть функції різного розміру. Отже, для представлення функції від 27 змінних запропонованим способом, необхідно використати $2^n = 2^{27}$ біт $\approx 16{,}8$ МБ.

Даний об'єм є відносно малим у порівнянні з сучасними можливостями персональних комп'ютерів. Після знаходження функції, її можна записати у мінімізованій формі, яка відображається у відносно просту комбінаційну схему. Якщо ж необхідна програмна реалізація, то мінімізовано форма також може бути легко обчислена на універсальному процесорі.

Оскільки комп'ютери частіше використовують байти, ніж біти, то для знаходження значення функції при певному аргументі, необхідно бути спочатку обрахувати позицію байту, далі позицію біту у даному байті, створити відповідну маску і за допомогою неї отримати потрібне значення. Дана операція є доволі

простою, але оскільки вона виконується постійно, то це може призвести до довшої роботи алгоритму.

Оскільки було визначено, що функція займає не так багато пам'яті, то для пришвидшення обчислень, значення функції було обрано зберігати у байтах.

Ще одним типом даних, які необхідно представити у програмному забезпечені, це власне регістр, який зберігає поточний стан генератору, я з якого береться аргумент функції.

Операції, які виконуються над регістром:

- запис початкового значення стану регістру заданим ключем від користувача;
- зчитування стану регістру, та використання його числового значення для пошуку значення функції у відповідній позиції таблиці істинності цієї функції;
- зсув регістру з заповненням нулем чи одиницею при зсуві;

Для забезпечення швидкого зсуву, та швидкого отримання числового значення регістру, найкращим способом є зберігання бітів один за одним, оскільки для регістру нам не треба буде знаходити значення окремих бітів його стану.

У мові Python розмір типу int залежить від розрядності операційної системи. Сучасне ядро Linux більше не має 32-розрядних релізів, і наразі підтримується лише 64-розрядні системи на даному ядрі. Операційна система Windows і досі надає можливість встановити 32-розрядну операційну систему.

Таким чином, розмір регістру обмежується відповідно n=32 або n=64. Розміру регістру у 32 біти буде не достатньо для дослідження функцій більшого розміру, тому попередньо було вказано у мінімальних характеристиках персонального комп'ютера, що операційна система має бути 64-розрядна.

Хоча 64 бітів достатньо для виконання дослідження, враховуючи що наразі відомо функції лише до 27 бітів, але якщо запропонований метод буде реалізований для значень n>64, то необхідно буде використовувати інший спосіб представлення стану регістру.

Зручним для цього може бути вбудований у Python тип bytearray, або відповідний тип у інших мовах програмування. Якщо відповідного типу не має, то можна створити масив байтів або чисел, саме так реалізували тип bytearray у Python його розробники.

Ще одним важливим типом даних у дослідженні є історія відвіданих станів регістру.

Це необхідно для того, щоб перевірити, чи дійсно побудований генератор відповідає умові розбіжності (тобто чи дійсно він відвідує всі стани і забезпечує період 2^n).

Визначимо, які операції необхідно буде виконувати над історією станів генератора:

- очистити історію перед початком перевірки чергової функції зворотного зв'язку;
- для заданого поточного стану генератора S, перевірити, чи знаходився генератор у стані S за останні 2^n ітерацій;
- записати до історії поточний стан генератора для того, щоб відмітити його як відвіданий стан.

Крім операцій, до історії станів також ставиться вимога стосовно можливості зберігання останніх 2^n станів.

Оскільки всього є 2^n станів, то для кодування одного стану необхідно використати як мінімум n бітів. Таким чином, щоб зберігати історію із 2^n станів, необхідно використати як мінімум $n*2^n$ бітів.

Ще однією проблемою є постійне зростання кількості станів, і відповідно ускладняється пошук поточного стану з кожним новим відвіданим станом генератора.

Важливо зазначити, що алгоритм перевірки генератора обмежений 2^n ітераціями. Тобто нам не треба зберігати порядок станів, у якому вони були відвідані, адже нам не треба буде видаляти стани після 2^n ітерацій алгоритму перевірки.

Це є важливою особливістю, оскільки це дозволяє зберігати стани у тому порядку, як нам зручно.

У програмному забезпечені для зберігання історії станів було використано структуру, подібну до структури представлення таблиці істинності функцій зворотного зв'язку.

А саме – кожен байт (можливо використовувати біти, але тоді додається час на обрахування його положення та відповідної маски) позначає нулем, що стан, який відповідає позиції цього байту (або біту) ще не був відвіданий. Тоді як одиницею позначаються стани, які були відвідані.

По-перше, це зменшує витрати пам'яті, і замість $n*2^n$ бітів тепер достатньо використати лише 2^n бітів. Хоча у програмному забезпечені використовувались байти, і тому загальна кількість пам'яті, необхідної для зберігання історії станів, становить $8*2^n$ бітів.

Дане значення співпадає з розміром таблиці істинності функції при однакових значеннях n.

По-друге, таке представлення історії дозволяє швидко знайти чи був відвіданий певний стан, використовуючи цей стан як індекс байту у історії станів регістру.

Таким чином, було визначено структуру даних для регістру стану генератора, булевої функції від кількості змінних n, та історії відвіданих раніше станів.

4.3 Опис модулю перебору множини функцій зворотного зв'язку

Першою задачею під час виконання даного дослідження було поставлено знаходження функцій відносно невеликого розміру, які б видавали повний період послідовності 2^n.

Для того, щоб знайти їх без попередніх знань стосовно того, які вони мають властивості, окрім забезпечення повного періоду, необхідно виконати повний перебір функцій заданої довжини n.

Для цього було реалізовано модуль програмного забезпечення findAllFunc.py.

Даний модуль містить ряд процедур, які виконують певну частину задачі перебору.

Запуск модуля відбувається процедурою findAll(n, limit=None), де параметри задаються користувачем і мають наступний зміст:

- n – розмір регістру, для якого виконується пошук функцій зворотного зв'язку;
- limit – обмеження по кількості перебираємих функцій; дане значення дозволяє виконувати пошук лише над підмножиною всіх можливих функцій заданого розміру: якщо користувач не передає даний параметр, то за замовчуванням виконується пошук у всій множині функцій.

Сама процедура findAll на кожній ітерації виконує наступні дії:

1. Сформувати чергову таблицю істинності f функції для заданого значення n=4.
2. Викликати процедуру checkFunc(f).

3. Якщо checkFunc(f) повернула значення True, то перетворити значення таблиці істинності f до типу рядка символів, та вивести даний рядок на екран.
4. Зменшити значення limit на одиницю.
5. Якщо limit перейшов значення нуля, то завершити виконання даної процедури.

Можна побачити, що розглянута процедура лише формує чергову функцію та забезпечує взаємодію з користувачем. Перевірка функції на повний період виконується у процедурі checkFunc(f), де f – це представлення таблиці істинності у вигляді послідовності байтів.

Алгоритм процедури checkFunc(f):

1. Значення регістру встановлюється рівним 0.
2. Створюється історія відвіданих станів, яка заповнюється значеннями 0 (тобто жодного відвіданого стану).
3. Далі починається цикл, який повторюється maxValues = 2^n разів.
4. У кожній ітерацій цього циклу, знаходиться значення функції від поточного стану регістру.
5. Регістр зсувається на 1 біт, і якщо функція приймає значення 1, то до кінця регістру додається дане значення.
6. Якщо новий стан вже був відвіданий, то дана функція не надає максимальний період, і повертається значення False.
7. Якщо для деякої булевої функції не було знайдено повторного стану за 2^n ітерацій, до дана функція пройшла повний період, тобто задовольняє поставленій умові, і тому процедура повертає значення True.

Приклад запуску процедури findAll(4) зображений на рисунку 4.1.

```
D:\Programs\Python39\python.exe D:/GoogleDrive/Projects/ThesisMag/findAllFunc.py
1000111101110000
1001110101100010
1010011101011000
1010101101010100
1010110101010010
1011010101001010
1011100101000110
1011111101000000
1100011100111000
1101010100101010
1110001100011100
1110010100011010
1110111100010000
1111000100001110
1111011100001000
1111110100000010
```

Рис.4.1. Результат роботи модуля findAllFunc.py при n=4

На рисунку 4.1 можна помітити, що перший та останній стовпці завжди мають значення 1 та 0 відповідно, що відповідає одній із властивостей, описаних у другому розділі.

4.4 Опис процедур для пошуку властивостей нелінійних функцій зворотного зв'язку з максимальним періодом

Для того, щоб дослідити знайдені функції та знайти спільні властивості, у першу чергу було виконано поверхневий огляд даних функцій, під час якого було виявлено потенційні властивості, які необхідно додаткового перевірити.

Для цього було створено модуль checkProperties.py, у якому реалізовано перевірку на певні властивості.

Головною процедурою у даному модулі є checkProperties(n). Ця процедура використовує процедуру findAll(n) з попереднього модуля для отримання списку функцій, а далі для кожної з функцій виконує перевірку з використанням наступних процедур:

- countOnes(f)
- checkHalfs(f)

- checkFirstLast(f)
- countOnesFirstHalf(f)
- check101(f)
- gcd(fs)

Де f – одна знайдена функція, fs – масив з множини усіх знайдених функцій для заданого n. Даний модуль також містить ряд інших перевірок, але оскільки вони не виявили властивостей, то їх було видалено із даного модуля.

Приклад роботи процедури checkProperties(n) для n=4 наведено нижче на рисунку 4.2.

```
D:\Programs\Python39\python.exe D:/GoogleDrive/Projects/ThesisMag/checkProperties.py
1000111101110000 8, True, True, 5, True, True
1001110101100010 8, True, True, 5, True, True
1010011101011000 8, True, True, 5, True, True
1010101101010100 8, True, True, 5, True, True
1010110101010010 8, True, True, 5, True, True
1011010101001010 8, True, True, 5, True, True
1011100101000110 8, True, True, 5, True, True
1011111101000000 8, True, True, 7, True, True
1100011100111000 8, True, True, 5, True, True
1101010100101010 8, True, True, 5, True, True
1110001100011100 8, True, True, 5, True, True
1110010100011010 8, True, True, 5, True, True
1110111100010000 8, True, True, 7, True, True
1111000100001110 8, True, True, 5, True, True
1111011100001000 8, True, True, 7, True, True
1111110100000010 8, True, True, 7, True, True
gcd = 510
```

Рис.4.2. Результат роботи процедури checkProperties(n) при n=4

Розглянемо як були реалізовані процедури, які використовуються у даному модулі.

Процедура countOnes(f) реалізується доволі просто. Для цього у таблиці істинності функції рахується кількість одиниць. У результаті було помічено, що дане значення завжди дорівнює половині розміру таблиці істинності. Після більш детального дослідження було виявлено, що перша та друга половини функції доволі схожі, і тому була реалізована наступна процедура.

checkHalfs(f) порівнює дві половини функції за наступним алгоритмом:

1. Для кожної позиції i=0..2^{n-1}-1, тобто для кожного значення першої половини, виконується наступна перевірка.
2. Якщо значення f[i] дорівнює значенню f[i+2^{n-1}], то дані значення не є протилежними і повертається значення False.
3. Якщо ітерація значень завершилася, то повертається значення True, оскільки всі значення є протилежними.

Наступна процедура, а саме checkFirstLast(f), перевіряє перший та останній біти на те, щоб вони дорівнювали одиниці та нулю відповідно. Також у даній процедурі було реалізовано перевірку двох значень посередині, а саме f[2^{n-1}-1] має дорівнювати 1, а f[2^{n-1}] = 0. Таким чином, дана процедура виконує перевірку відразу чотирьох властивостей.

Процедура countOnesFirstHalf(f) рахує кількість одиниць у першій половині функції. Дана процедура включає у собі перевірку відразу двох властивостей:

- отримана кількість є непарною;
- отримана кількість для непарних значень n зазвичай становить менше половини, а для парних значень n – більше.

Процедура check101(f) перевіряє, чи має досліджувана функція обидва нулі на позиціях (010…) та (101…). Властивість, яка перевіряється, полягає у тому, що функція від цих значень не може одночасно приймати нулі. Відповідно, якщо вона має два нулі, то властивість не виконується і повертається значення False. У іншому випадку (коли значення різні, або обидва дорівнюють одиниці), повертається значення True.

Процедура gcd(fs) шукає найбільший спільний дільник для заданого масиву функцій. Алгоритм обчислення НСД:

1. Із масиву fs вилучити перше значення і зберегти його у змінну divider.

2. Із масиву вилучити перше значення, та записати його у змінну currentf.
3. Знайти найбільший спільний дільник для значень divider та currentf, та зберегти його у змінну divider.
4. Якщо у масиві fs залишилася хоча б одне значення, перейти до п.2.
5. Повернути значення divider.

На рисунку 4.3 наведено частковий вивід модуля checkProperties.py при розмірі n=5.

```
11111110110110110000000100100100 16, True, True, 13, True, True
11111110100100100000000010110110 16, True, True, 11, True, True
11111110100111100000000010110000 16, True, True, 13, True, True
11111110101000100000000010101110 16, True, True, 11, True, True
11111110101011100000000010101000 16, True, True, 13, True, True
11111110101101100000000010100100 16, True, True, 13, True, True
11111111110000010000000000111110 16, True, True, 11, True, True
11111111110001110000000000111000 16, True, True, 13, True, True
11111111110011010000000000110010 16, True, True, 13, True, True
11111111110100110000000000101100 16, True, True, 13, True, True
11111111110101010000000000101010 16, True, True, 13, True, True
11111111110110010000000000100110 16, True, True, 13, True, True
gcd = 131070
```

Рис.4.3. Частковий результат роботи модуля checkProperties.py при n=5

4.5 Опис модулю для побудови генератора псевдовипадкових послідовностей

Побудова генераторів псевдовипадкових послідовностей за запропонованим методом складається у 2 етапи, які рознесені у окремі модулі програмного забезпечення.

Перший модуль findAllFuncOptimized.py реалізовує пошук функцій з врахуванням знайдених властивостей.

У першу чергу враховуються властивості стосовно того, що друга половина повністю відображає першу половину з оберненими значеннями. Також

враховується, що перша половина має починатися на одиницю і закінчуватись на нуль. Враховуючи ці дані властивості, розглядаються лише всі можливі комбінації для значень від 1 до 2^{n-1}-2, а інші значення вираховуються з даних.

Під час перебору, функції спочатку тестуються на властивості за допомогою тих самих процедур, які були описані у пункті 3.4. Якщо функція не проходить хоча б одну з цих перевірок, то вона відразу пропускається.

Також було використано властивість, що для кожної функції з повним періодом, є симетрична функція, яка також має повний період. Таким чином, при знаходженні однієї функції, виконується симетричне обернення у середині кожної з половин функції, і таким чином отримується ще одна функція з повним періодом.

Частковий результат роботи даного модуля наведено на рисунку 4.4.

```
D:\Programs\Python39\python.exe D:/GoogleDrive/Projects/ThesisMag/findAllFuncOptimized.py
10000000000000001010110011111111011111111111111110101001100000000
11111111001101010000000000000001000000001100101011111111111111110

10000000000000001010110111111011011111111111111101010010000000100
11011111101101010000000000000001001000000100101011111111111111110

10000000000000001010111010111111011111111111111101010001010000000
11111101011101010000000000000001000000101000101011111111111111110

10000000000000001010111011101111011111111111111101010001000100000
11110111011101010000000000000001000010001000101011111111111111110

10000000000000001010111011111101011111111111111101010001000000010
10111111011101010000000000000001010000010001010111111111111111110
```

Рис.4.4. Частковий вивід модуля findAllFuncOptimized.py

Другий етап побудови генератора полягає у використанні даних функцій для генерації псевдовипадкової послідовності. Цей етап було реалізовано у модулі generator.py.

Головною частиною даного модуля, є клас Generator, який має наступні методи:

- __init__(n, f, s)

- setFunc(f)
- setState(s)
- getNextBit()

Клас Generator також має внутрішні поля n, f та s, у яких зберігаються його розмір, функція зворотного зв'язку та його поточний стан. Ці поля мають бути обов'язково передані у якості параметрів при створенні екземпляру класу Generator. Під час створення екземпляру, викликається метод __init__, який валідує введені параметри і встановлює їх у внутрішні поля класу.

Для того, щоб можна було змінити стан або функції без повторного створення екземпляру, також були створені методи setFunc та setState, які дозволяють змінити функцію або стан, якщо це необхідно.

Найбільш важливим методом даного класу є getNextBit, який на основі внутрішніх змінних, генерує псевдовипадковий біт та змінює внутрішній стан регістру. Алгоритм даного методу:

1. Знайти значення функції у поточному стані f[s] та зберегти його до змінної lastBit= f[s].
2. Зсунути регістр вліво. Дана операція реалізована шляхом множення на 2, тобто s*=2.
3. Відкинути старший біт за допомогою маски 2^n-1.
4. Додати lastBit до стану s=s+lastBit. Оскільки після множення на два молодший біт завжди дорівнює нулю, а lastBit може мати значення лише 0 або 1, то додавання є еквівалентом запису значення lastBit у молодший біт.
5. За допомогою маски 2^n отримати значення старшого біту, і повернути його значення.

З використанням даного модулю, було створено екземпляр класу для однієї із знайдених функцій та було запущено цикл із 500 ітераціями. Результат наведено на рисунку 4.5.

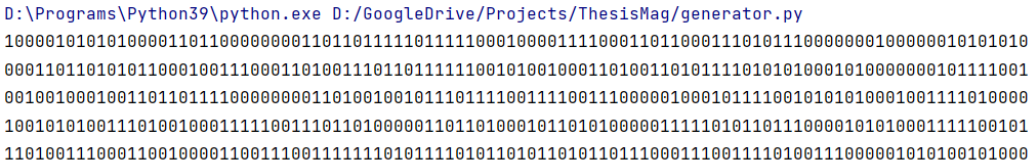

Рис.4.5. Результат роботи модулю generator.py

Висновки до розділу 4

У розділі 4 було поставлене завдання розробки програмного забезпечення, яке має допомагати виконати дослідження булевих функцій зворотного зв'язку, а також розробити програмне забезпечення для побудови генератору за запропонованим методом.

У результаті роботи можна зробити наступні висновки:

1. Досліджено можливі способи представлення булевих функцій, регістру стану та історії станів у пам'яті. Для булевих функцій було обрано зберігання таблиці істинності і вигляді послідовності бітів. Регістр стану було представлено у вигляді цілого числа int. Історія станів зберігається у вигляді послідовності бітів, де індекс біту відповідає значенню стана.

2. Створено програмне забезпечення для пошуку властивостей булевих функцій з повним періодом. Для цього було розроблено модуль повного перебору та модуль перевірки різних властивостей заданих функцій. У результаті було знайдено ряд властивостей, які можна використати для створення методу пошуку функцій зворотного зв'язку.

3. На основі знайдених властивостей, було реалізовано запропонований метод побудови генераторів псевдовипадкових послідовностей. Реалізація даного методу описана у двох модулях. Перший модуль виконує синтез нелінійних функцій з повним періодом. Другий модуль реалізує генератор псевдовипадкових послідовностей на основі синтезованих функцій.

ВИСНОВКИ

У дисертації було поставлено мету розробити метод синтезу функцій для криптографічного генератору на основі зсувного регістру з зворотнім зв'язком, який дозволяє ефективно синтезувати функції від кількості змінних більше 100.

У результаті виконаної роботи можна зробити наступні висновки:

1. На сьогодні, криптографічні генератори дуже широко використовуються у комп'ютерних системах. Перевагою потокових шифрів є те, що розшифрування та шифрування інформації відбуваються дуже швидко. Оскільки кількість інформації постійно збільшується, а системи реального часу стають все більш вживаними, то відповідно все більше використовуються криптографічні генератори.

2. Вимоги, які ставляться до апаратних та програмних реалізацій генераторів псевдовипадкових послідовностей, значно збільшуються при використанні їх для захисту інформації. Має забезпечуватись максимальний період повторення, а також мають дотримуватись вимоги стосовно статистичних характеристик псевдовипадкової послідовності. Послідовність, яка генерується, має бути непередбачуваною. Для систем реального часу додатково ставиться умова стосовно швидкодії генератора.

3. Для забезпечення непередбачуваності послідовності, у генераторі мають бути присутні необоротні функції. Дослідження існуючих генераторів показали, що розповсюдженими прикладами таких функцій є нелінійні булеві функції, математичні перетворення над числами, алгоритми чи шифроблоки, які є стандартизованими.

4. Найкращі показники стосовно швидкодії генератора належать до генераторів, які використовують зсувні регістри. Інші типи генераторів, такі як зведення у ступінь, або з використанням інших складних перетворень, неможливо реалізувати на апаратурі з невеликою кількістю елементів. Також такі

генератори потребують великої кількості обчислень. Для практичних задач такі генератори підходять лише для генерації невеликих послідовностей (ключі, паролі, тощо) або у системах, де щвидкодія та складність комбінаційних схем не є важливими.

5. При використання зсувних регістрів з лінійним зворотнім зв'язком, генератори не виконують критерій криптографічної стійкості. При цьому функції, при яких генератор має період 2^n, є рознесеними. Із-за цього побудова комбінаційних схем ускладняється, а повідомлення легко зламуються декомпозицією.

6. Генератори з регістром зсуву можна значно покращити, якщо для зворотного зв'язку використовувати нелінійну необернену булеву функцію, яка б забезпечувала максимальний період. На сьогодні існує проблема синтезу таких функцій, оскільки їх частка у множині всіх функцій розміру n різко зменшується при збільшенні n, а для забезпечення більшого періоду, необхідно використовувати більше значення n. Тому виникає потреба у розробці нових методів формування таких функцій.

7. У якості критеріїв ефективного генератору було встановлено швидкодію, простоту та криптостійкість. Останній критерій включає у собі статистичні властивості послідовності, використання необернених перетворень, неможливість передбачити послідовність. Крім того, має забезпечуватись максимально можливий період.

8. Аналіз булевих функцій, які при їх використанні надають максимальний період, показав, що всі такі функції мають спільні властивості, які можна легко перевірити для будь-якої заданої булевої функції.

9. Частина таких властивостей полягає у властивостях таблиці істинності даної функції. Це включає у себе протилежність значень на різних половинах

таблиці істинності, симетричність значень усередині кожної половини. Також було виявлено залежності між деякими парами позицій у таблиці істинності.

10. Показано, що існує чотири набори аргументів, для який встановлені відповідні значення, які однакові для всіх функцій, які виконують умову повного періоду генератора. А саме, $f(0)=1$, $f(2^n-1)=0$, $f(2^{n-1}-1)=1$ та $f(2^{n-1})=0$. Невиконання цих властивостей призводить до періоду з всього одного стану, а саме з стану 0 або з стану 2^{n-1}.

11. Встановлено, що числове представлення таблиці істинності завжди ділиться на певне число M_n. Було виведено формулу, за якою можна розрахувати дане значення, та показано, що дана властивість дійсно виконується при більших значеннях n.

12. З врахуванням вищезазначених властивостей, було розроблено метод для формування булевих функцій, які б забезпечували максимальний період. Запропонований метод має значну відмінність від існуючих – він враховує властивості нелінійних булевих функцій, які надають можливість генерації послідовності з максимальним періодом.

13. Виконано аналіз стосовно ефективності запропонованого методу та часових витрат на його реалізацію. У результаті було показано, що цей метод дозволяє синтезувати функції для великих значень n з значно меншими часовими витратами у порівнянні з прямим пошуком, який розглядає всі можливі функції для заданого значення n.

14. Проаналізовано різні способи для відображення у пам'яті комп'ютера значень функцій, історії станів та регістру генератора. Виявлено ефективні способи для їх зберігання, які дозволяють швидко виконувати відповідні операції над ними. Важливим моментом є запропонована оптимізація для зберігання історії станів з урахуванням того, що порядок відвідування не є важливим у цьому випадку.

15. Створено програмний модуль, який використовувався при пошуку загальних властивостей функцій, які задовольняють поставлену умову повного періоду. Даний модуль реалізує прямий пошук функцій без врахування будь-яких властивостей та ряд процедур, які перевіряють відповідність до певних властивостей.

16. Створено програмний модуль для побудови генераторів на основі зсувного регістру. Цей модуль включає у себе формування булевих функцій зворотного зв'язку за запропонованим методом, та побудову генераторів на основі сформованих функцій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Rivest R.L. Cryptographic communications system and method / R.L. Rivest, A. Shamir, L.M. Adleman // U.S. Patent Office. – 1977. – 20 p.
2. Golomb, S.W. Shift register sequences // Aegean Park Press. – 1967. – 224 p.
3. Alfke P. Efficient Shift Registers, LFSR Counters, and Long PseudoRandom Sequence Generators // Xilinx. – 1996. – 6 p.
4. Koopman P. Maximal Length LFSR Feedback Terms [Електронний ресурс] // Carnegie Mellon University. – 2014. – Режим доступу до ресурсу: https://users.ece.cmu.edu/~koopman/lfsr/.
5. Hayati A. Maximal Length polynomial generator [Електронний ресурс] // GitHub. – 2021. - Режим доступу до ресурсу: https://github.com/hayguen/mlpolygen.
6. Massey J. L. Shift-register synthesis and BCH decoding // IEEE Transactions on Information Theory, IT-15 (1). – 1969. – P.122-127.
7. Rachwalik T. Generation of Nonlinear Feedback Shift Registers with special-purpose hardware / Rachwalik T.; Szmidt J.; Wicik R.; Zabłocki J. // Military Communication Institute. – 2012. – 6 p.
8. Günther C.G. Alternating Step Generator Controlled by de Bruijn Sequence // Advances in Cryptology — EUROCRYPT '87. – 2000. – 290 p.
9. Dubrova, E. On analysis and synthesis of (n, k)-non-linear feedback shift registers / E. Dubrova, M. Teslenko, H. Tenhunen // Proceedings of the Conference on Design, Automation and Test in Europe - DATE ’08. – 2008. – 6 p.
10. Poorghanad A. Generating High Quality Pseudo Random Number Using Evolutionary Methods / A. Poorghanad, A. Sadr, A. Kashanipour // IEEE Congress on Computational Intelligence and Security. – 2008. – P.331-335.

11. Конахович Г.Ф., Климчук В.П., Паук С.М., Потапов В.Г. Защита информации в телекоммуникационных системах. К.: ”МК-Пресс”.- 2005.- 288 с.
12. Соколов А.В. Защита информации в распределенных корпоративных сетях и системах/ А.В. Соколов, В.Ф. Шаньгин.- М.:”ДМК Пресс”.- 2002.- 636 с.
13. Широчин В.П., Генератор ПВЧ з реконфігуративною структурою на базі каскаду Голлмана / В.П. Широчин,. І.В.Васильцов, Б.З. Карпінський // Вісник Національного технічного університету України “КПІ” Інформатика, управління та обчислювальна техніка. К., ТОО „ВЕК+”.- № 41.- 2004.- С. 15-20.
14. Coppersmith P. The shrinking generator / P.Coppersmith, C. Krawchuk, E. Mansour // Proceeding of conference CRYPTO-93, LNCS-773.: Springer-Verlag.- 1993.- P.22-39.
15. Blum M. How to generate cryptographically strong sequences of pseudorandom bits/ M. Blum, S. Micali // SIAM Journal on Computing.- 1984.- Vol.13.-P. 850-864.
16. Anderson R.J. A5 (Was: HACKING DIGITAL PHONES) [Електронний ресурс] // Google Groups. – 1994. - Режим доступу до ресурсу: https://web.archive.org/web/20071214085910/http://groups.google.com/group/uk.telecom/msg/ba76615fef32ba32.
17. Biryukov A. Real Time Cryptanalysis of A5/1 on a PC / Biryukov A. Shamir A., Wagner D. // Fast Software Encryption-FSE. – 2000. – P.1-18.
18. Golic J. Cryptanalysis of Alleged A5 Stream Cipher // Eurocrypt. - 1997. - P.239–255.
19. Eli B., Dunkelman O. Cryptanalysis of the A5/1 GSM Stream Cipher // Indocrypt. - 2000. - P.43–51.

20. Ekdahl P. Another attack on A5/1 / Ekdahl P., Johansson T. // IEEE Transactions on Information Theory. 49 (1). - 2003. - P.284–289.
21. Марковский А.П., Гаваагийн Улзисайхан, Бардис Е.Г., Осадчий В.В. Способы обеспечения ацикличности и необратимости генераторов псевдо-случайных последовательностей/ А.П.Марковский, Гаваагийн Улзисайхан, Е.Г.Бардис, В.В. Осадчий // Вісник Національного технічного університету України ”КПІ”. Інформатика, управління та обчислювальна техніка. К.:"ВЕК++".- 1999, - № 32. - С.67-74.
22. NIST Special Publicaion 800-22: A Statistical Test Suite for Random and Pseudorandom Number. 2000. -348 p.
23. Чугунков И.В. Система оценки качества генераторов псевдослучайных кодов/ И.В. Чугунков // Научная секция МИФИ-2000.- Т.11,12.- С.45-46.
24. Марковский А.П.. Об одном подходе к определению сложности случайных и псевдослучайных двоичных последовательностей / А.П. Марковский, Мустафа Акрам Ареф Найеф, А.В. Бойко // Вісник національного технічного університету України ”КПІ”. Інформатика, управління та обчислювальна техніка. – 2002.- №37.- С.120-129.
25. Kurosava K. A relationship between linear complexity and *k*-error linear complexity./ К.Kurosava K., F. Sato, T.Sakata, W.Kishimoto // IEEE Trans. Information Theory, 2000.- V.46,-№3.-P.694-698.
26. Fluhrer S. Weaknesses in the Key Scheduling Algorithm of RC4 / Fluhrer S., Mantin I., Shamir A. // Selected Areas of Cryptography: SAC. - 2001, - P.1-24.
27. Duong T. Rizzo/Duong chosen plaintext attack (BEAST) on SSL/TLS 1.0 [Електронний ресурс] / Duong T., Rizzo J. // Bugzilla. – 2011. - Режим доступу до ресурсу: https://bugzilla.mozilla.org/show_bug.cgi?id=665814.
28. Sklyarov D. Hidden Keys to Software Break-Ins and Unauthorized Entry // A-List Publishing. - 2004. - P.92–93.

29. Isobe T. Security of RC4 Stream Cipher / Isobe T., Ohigash T. // Hiroshima University. - 2013. - 6 p.
30. AlFardan N. On the Security of RC4 in TLS / AlFardan N., Bernstein D., Paterson K., Poettering B., Schuldt J. // Royal Holloway University of London. - 2013. - 6 p.
31. Popov A. Prohibiting RC4 Cipher Suites [Електронний ресурс] // Internet Engineering Task Force (IETF). – 2015. - Режим доступу до ресурсу: https://datatracker.ietf.org/doc/html/rfc7465.
32. Марковский А.П. Оценка сложности двоичных последовательностей с использованием нелинейных моделей / А.П. Марковский, Зохре Карим Заде Сейфоллах, О.О. Яцишина // Труды 8-й международной научно-технической конференции ”Современные информационные и электронные технологии” (21-25 травня 2007 р., м. Одеса). Одеса: ОНТУ.-2007. - С.194.
33. Gutierrez J. On the Linear and Nonlinear Complexity Profile on Nonlinear Pseudorandom Number Generators / J. Gutierrez, I.A. Shparlinski, A. Winterhof // IEEE Trans. Information Theory.-2003.- Vol. 49.- № 1. - P.60-64.
34. Самофалов К.Г., Марковський О.П., Абабне О.А. Оцінка якості генераторів двійкових послідовностей з використанням нелінійної відтворюючої моделі / К.Г. Самофалов, А.П. Марковський, О.А. Абабне // Проблеми інформатизації та управління. Збірник наукових праць: Випуск 1(19).- К.,НАУ.- 2007.- С.142-147.
35. Zhang M. Maximum Correlation Analysis of Nonlinear Combining Function in Stream Cipher / M. Zhang // Journal of Cryptology,- 2000- Vol.13.- P. 301-313.
36. Зохре Карим Заде Сейфоллах. К проблеме оценки качества случайных и псевдослучайных двоичных последовательностей / Зохре Карим Заде Сейфоллах// Матеріали VII Міжнародної науково-технічної конференції

”Системний аналіз та інформаційні технології” (13-16 вересня 2006 р., м.Київ). К.: НТУУ ”КПІ”.-2006.-С.171-174.

37. Shijian C. Pseudo-random number generator based on a generalized conservative Sprott-A system / Shijian C., Zhijun K., Zenghui W. // Springer: Nonlinear Dynamics. - 2021. – 19 p.
38. Dong E. A new class of Hamiltonian conservative chaotic systems with multistability and design of pseudo-random number generator / Dong E., Yuan M., Du S., Chen, Z. // Applied Mathematical Modelling. - 2019. – 12 p.
39. Luyao W. Pseudo-Random Number Generator Based on Logistic Chaotic System / Luyao W., Hai C. // Entropy Vol.21. – 2019. – 960 p.
40. Rezk A.A. Reconfigurable Chaotic Pseudo Random Number Generator based on FPGA / Rezk A.A., Madian A.H., Radwan A.G., Soliman A.M. // International Journal of Electronics and Communications. – 2019. – P.174-180.
41. EL-Latif A.A. Controlled alternate quantum walk-based pseudo-random number generator and its application to quantum color image encryption / EL-Latif A.A., Abd-El-Atty B., Venegas-Andraca S.E. // Physica A: Statistical Mechanics and its Applications Vol. 547. – 2020. – 17 p.
42. Ayubi P. Deterministic chaos game: A new fractal based pseudo-random number generator and its cryptographic application / Ayubi P., Setayeshi S., Rahmani A.M. // Journal of Information Security and Applications. – 2020. – 23 p.
43. Datcu O. Chaos Based Cryptographic Pseudo-Random Number Generator Template with Dynamic State Change / Datcu O., Macovei C., Hobincu R. // International Conference on Telecommunications and Signal Processing. – 2020. – 17 p.

44. Seberry J., Zhang X., Zheng Y. Nonlinearity and propagation characteristics of balanced Boolean functions / J. Seberry, X. Zhang, E. Zheng //Information and Computation Academic Press. 1995.-Vol. 119,- № 1 - P.1-13.
45. Марковский А.П. Метод побудови ефективного нелінійного генератора псевдовипадкових двійкових послідовностей / А.П. Марковский, Л.Б. Ермоленко // Вісник Національного технічного університету України ”КПІ”. Інформатика, управління та обчислювальна техніка. 2003, - № 39. - С.15-23.
46. Марковский А.П. Построение генараторов псевдослучайных последовательностей на сдвиговом регистре с нелинейной обратной связью / А.П. Марковский, Хан Йонг Ли, Л.Б. Ермоленко // Вісник Національного технічного університету України ”КПІ”. Інформатика, управління та обчислювальна техніка. 2004, - № 41. - С.75-84.
47. Самофалов К.Г. Метод синтеза нелинейной функции обратной связи для сдвигового регистра с максимальным периодом повторения / К.Г. Самофалов, А.П. Марковский, Зохре Карим Заде Сейфоллах // Проблеми інформатизації та управління. Збірник наукових праць: К.,НАУ.- 2006.- Випуск 2(17).- С.105-111.
48. Марковський О.П.. Ефективний метод побудови нелінійних генераторів для телекомунікаційних систем / О.П. Марковський, Зохре Карім Заде Сейфолах, В.Е.Гурін // Электроника и связь. Тематический выпуск ”Проблемы электроники” ч.3. – ПЦ ”Аверс” - 2007.- С.87-89.
49. Kösemen C. Genetic programming based pseudorandom number generator for wireless identification and sensing platform / Kösemen C., Dalkılıç G., Aydın Ö. // Turkish Journal of Electrical Engineering and Computer Sciences. – 2018. – P.2500-2511.

50. Picek S. Evolving cryptographic pseudorandom numbergenerators / Picek S., Sisejkovic D., Rozic V., Yang B., Jakobovic D., Mentens N. // Lect Notes Comp Sci. - 2016. - P.613-622.
51. Chlumecky M. Application of random number generators in genetic algorithms to improve rainfall-runoff modelling / Chlumecky M., Buchtele J., Richta K. // Hydro. – 2017. – P.350-355.
52. Knezevic K. Combinatorial optimization in cryptography // 40th International Convention on Information andCommunication Technology, Electronics and Microelectronics. - 2017. - P.1324-1330.
53. Hanouti I.E. A Lightweight Pseudo-Random Number Generator Based on a Robust Chaotic Map / Hanouti I.E., Fadili H.E., Souhail W., Masood F. // Fourth International Conference On Intelligent Computing in Data Sciences. – 2020. 6 p.
54. AL-khatib M. Acoustic lightweight pseudo random number generator based on cryptographically secure LFSR / AL-khatib M., Lone A. // Computer Network and Information Security. – 2018. P.35-45.
55. Ullah I. Entropy as a Service: A Lightweight Random Number Generator for Decentralized IoT Applications / Ullah, I., Meratnia, N., & Havinga, P. J. M. // IEEE International Conference on Pervasive Computing and Communications Workshops. – 2020. – 6 p.
56. Shannon C. Communication Theory of Secrecy Systems // Bell System Technical Journal 28. – 1949. – P.656–715.
57. Chabloz J.M. An Algorithm for Constructing a Fastest Galois NLFSR Generating a Given Sequence / Chabloz J.M., Shohreh S.M. & Dubrova E. // International Conference on Sequences and Their Applications. – 2010. – P.41-54.

58. Dubrova E. Finding Matching Initial States for Equivalent NLFSRs in the Fibonacci and the Galois Configurations // IEEE Transactions on Information Theory. – 2010. – P.2961-2966.

59. Широчин В.П. Вопросы проектирования средств защиты информации в компьютерных системах и сетях / В.П. Широчин, В.Е. Мухин, А.В. Кулик.- К.ВЕК++.- 2000.- 111 с.

60. Харин Ю.С. Математические и компьютерные основы криптологии./ Ю.С. Харин, В.И. Бердник, Г.В. Матвеев, С.В. Агиевич. - Мн.:Новое знание.- 2003.- 382 с.

61. Солодовников В.И. Бент-функции из конечной абелевой группы в конечную абелеву группу / В.И. Солодовников // Дискретная математика.- 2002,Т.14.-Вып.1.- С.99-113.

62. Kurosawa K. Design of SAC/PC(l) of Order k Boolean Functions and Three Other Cryptographic Criteria / K. Kurosawa, T. Satoh //Proc. International Conf. Advanced in Cryptology – Eurocrypto'97, LNCS 1233 – 1997- P.433-449.

63. Maintra S. Further construction of resilient Boolean functions with very high nonlinearity / S. Maintra, E.Pasalic // IEEE Trans. on Information Theory.-2002.- Vol.48, No. 7, , pp. 1825-1834.

64. Zhang X. On Relationship among Avalanche, Nonlinearity and Correlation Immunity / X. Zhang, Y. Zheng // Proceedings of Asiacrypt-2000, LNCS 1976, Springer,-2000.- P.135-142 .

65. Самофалов К.Г. Комбинаторный подход к получению булевых функций, обладающих строгим лавинным эффектом / К.Г.Самофалов, А.П. Марковский // Электронное моделирование.- 2004,- Том. 26, - № 3, - с.27-40.

66. Марковський О.П. Метод одержання булевих балансних функцій, які задовольняють критерію чіткого лавинного ефекту / О.П. Марковський, І. Ель-Хамі, Л.Р.Рябуха // Наукові вісті НТУУ "КПІ",- 2001- № 2,- К.,"ЕКМО".- С.31-40.

67. Орлова М.Н. Метод синтеза управляемых генераторов балансных SAC-функций / М.Н.Орлова, Аль-Хавальди Али, Зохре Карим Заде. // Вісник Національного технічного університету України ”КПІ” Інформатика, управління та обчислювальна техніка. К.: „ВЕК+”.- 2004. – № 41.- С.155-164.

68. Марковский А.П. Получение систем ортогональних SAC-функций для систем защиты информации / А.П. Марковский, А.Н. Абу Усбах, Аль-Омар Салех // Вісник Національного технічного університету України ”КПІ”. Інформатика, управління та обчислювальна техніка.- 2001.- № 36.- К.,"ВЕК++",- С.94-108.

69. Марковский А.П. Метод синтеза ортогональных систем булевых SAC–функцій / А.П.Марковский, Зохре Карим Заде Сейфоллах, О.С. Троян // Вісник Національного технічного університету України ”КПІ”. Інформатика, управління та обчислювальна техніка. К.: „ВЕК+”.- 2005 – № 43.- С.21-31.

70. Liu J. A New Upper Bound on the Order of Affine Sub-families of NFSRs / J. Liu, Q. Zheng, D. Lin // Journal of Systems Science and Complexity. 2020.- vol.33.- № 2.- P.196-214.

71. Zhao X-X. Further results on the equivalence between Galois NFSRs and Fibonacci NFSRs / X-X. Zhao, W-Q. Qi, J-M. Zhang // Design, Codes and Cryptography.-2020.-Vol.88.- № 1.- P.153-171.

72. Zhang J-M. A new Method for Finding Affine Sub-Families of NFSR-Sequences / J-M.Zhang, T.Tian, W-F.Qi., Zheng Q-X. // IEEE Transactions on Information Theory. – 2019.-vol. 65.- № 2.- P.1249-1257.
73. Shay Gueron, Vlad Krasnov Software Implementation of Modular Exponentiation, Using Advanced Vector Instructions Architectures // Proceeding of 4th International Workshop, WAIFI 2012, Bochum, Germany, July 16-19, 2012.- P.119-135
74. Жуков И. Ю. Принципы построения криптостойких генераторов псевдослучайных кодов / И. Ю. Жуков, М. А. Иванов, С. А. Осмоловский // Проблемы информационной безопасности. Компьютерные системы. 2001. № 1. С. 55-65.
75. Иванов М. А. Криптографические методы защиты информации в компьютерных системах и сетях: учеб. пособие / М. А. Иванов, И. В. Чугунков; под ред. М. А. Иванова. -М.: НИЯУ МИФИ, 2012. 400 с.
76. Власенко А.В. Генерация псевдослучайных последоватеьностей на основе линейного конгруэнтного метода и полиномиального счетчика / А.В. Власенко, П.И. Дзьобан // Intellectual Technologies on Transport. -2017.- № 4.-С.47-53.
77. Szmidt J. Generation of Nonlinear Feedback Shift Registers with Special-Purpose Hardware //
78. J. Szmidt // Proceeding of 4th International Workshop, WAIFI 2012, Bochum, Germany, July 16-19, 2012.- P.119-135.
79. Zheng W. A generalization of modied de Bruijn sequences / W. Zheng, Y. L. Cao, Y. C. Zhou and T. Y. Xu // Proceeding of The 2nd International Conference on Information Science and Engineering, Hangzhou, China.-2010.- P. 1705-1708.

80. Li C. Construction of de Bruijn Sequences From LFSRs With Reducible Characteristic Polynomials / C. Li, X. Zeng, C. Li, T. Helleseth and M. Li //IEEE Transactions on Information Theory. – 2016.-vol. 62.- № 1.- P.610-624.