

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

«На правах рукопису»

УДК 004.056.55

«До захисту допущено»

В.о. завідувача кафедри

_____ Сергій ЯКОВЛЄВ

«__» _____ 2023 р.

**Магістерська дисертація
на здобуття ступеня магістра**

за освітньо-науковою програмою

«Математичні методи криптографічного захисту інформації»

зі спеціальності: 113 Прикладна математика

на тему: **«Побудова атаки часткового вгадування на
потоків шифр Струмок»**

Виконав:

студент II курсу, групи ФІ-12мн

Ковалевський Олександр Юрійович

Керівник:

к. ф.-м. н., ст. викл. каф. ММЗІ

Фесенко Андрій В'ячеславович

Рецензент:

к. т. н., доц. каф. ІБ

Стьопочкіна Ірина Валеріївна

Засвідчую, що у цій магістерській
дисертації немає запозичень
з праць інших авторів без
відповідних посилань.

Студент _____

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»

Навчально-науковий фізико-технічний інститут
Кафедра математичних методів захисту інформації

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

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

_____ Сергій ЯКОВЛЄВ

«__» _____ 2023 р.

ЗАВДАННЯ
на магістерську дисертацію

Студент: Ковалевський Олександр Юрійович

1. Тема роботи: *«Побудова атаки часткового вгадування на потоковий шифр Струмок»*, науковий керівник дисертації: к. ф.-м. н., ст. викл. каф. ММЗІ Фесенко Андрій В'ячеславович,

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

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

3. Об'єкт дослідження: *інформаційні процеси в системах криптографічного захисту інформації*

4. Предмет дослідження: *стійкість потокового шифру Струмок до атак часткового вгадування*

5. Перелік завдань:

– Дослідити будову поточкових шифрів сімейства SNOW та наявні атаки на них.

– Дослідити будову потокового шифра Струмок і провести порівняльний аналіз із шифром SNOW-2.0.

– Дослідити методи побудови атак часткового вгадування та їхні різновиди.

– Побудувати атаку часткового вгадування на потоковий шифр Струмок та оцінити її складність.

– Надати рекомендації щодо покращення стійкості потокового шифра Струмок до атак часткового вгадування.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу:
Презентація доповіді

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

8. Дата видачі завдання: 10 вересня 2022 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання	Примітка
1	Узгодження теми роботи із науковим керівником	01-15 вересня 2022 р.	Виконано
2	Огляд опублікованих джерел за тематикою дослідження	Вересень-жовтень 2022 р.	Виконано
3	Дослідження і порівняння будови поточкових шифрів сімства SNOW	Листопад-грудень 2022 р.	Виконано
4	Дослідження будови поточкового шифра Струмок	Січень 2023 р.	Виконано
5	Дослідження атак часткового вгадування та методів їхньої побудови	Січень-лютий 2023 р.	Виконано
6	Ознайомлення із програмним застосунком Autoguess	Лютий-березень 2023 р.	Виконано
7	Пошук базисів вгадування і побудова атаки на поточковий шифр Струмок	Березень 2023 р.	Виконано
8	Аналіз і оформлення отриманих результатів	Квітень-травень 2023 р.	Виконано

Студент

_____ Олександр
КОВАЛЕВСЬКИЙ

Керівник

_____ Андрій ФЕСЕНКО

РЕФЕРАТ

Кваліфікаційна робота містить: 48 стор., 9 рисунків, 23 джерела.

Метою роботи є дослідження криптографічної стійкості потокового шифру Струмок до атаки часткового вгадування.

Об'єктом дослідження є інформаційні процеси в системах криптографічного захисту інформації.

Предметом дослідження є стійкість потокового шифру Струмок до атаки часткового вгадування.

В роботі знайдено базиси вгадування для побудови атаки часткового вгадування на потоковий шифр Струмок. Пошук базисів вгадування проведено із використанням адаптованого програмного застосунку Autoguess. Побудовано атаки часткового вгадування на потоковий шифр Струмок.

Запропоновано ідею розбиття змінних шифру на підслова для збільшення кількості функціональних зв'язків між ними та зменшення обчислювальної складності атаки.

Усього знайдено 176 базисів вгадування довжиною 7 64-бітових слів при використанні 11 тактів шифру Струмок. Складність кожної атаки становить $O(2^{448})$ операцій.

ПОТОКОВИЙ ШИФР СТРУМОК, АТАКА ЧАСТКОВОГО
ВГАДУВАННЯ

ABSTRACT

The qualification work contains: 48 pages, 9 figures, 23 sources.

The purpose of the paper is to investigate the cryptographic security of the Strumok stream cipher against guess-and-determine attacks.

The object of the research is the information processes in systems of cryptographic protection of information.

The subject of the paper is the cryptographic resistance of the Strumok stream cipher against guess-and-determine attacks.

The paper found guess bases for a guess-and-determine attacks on the Strumok stream cipher. The search for guess bases was carried out using the adapted software application Autoguess. Constructed guess-and-determine attacks on the Strumok stream cipher.

The idea of dividing cipher variables into subwords is proposed to increase the number of functional connections between them and reduce the computational complexity of the attack.

A total of 176 guess bases of length 7 64-bit words were found using 11 ticks of the Strumok stream cipher. The complexity of attack is $O(2^{448})$ operations.

STRUMOK STREAM CIPHER, GUESS-AND-DETERMINE ATTACK

ЗМІСТ

Вступ.....	9
1 Особливості будови поточкових шифрів сімейства SNOW	11
1.1 Огляд поточкового шифру SNOW	11
1.2 Огляд поточкового шифру SNOW 2.0	14
1.3 Огляд поточкового шифру <i>SNOW-3G</i>	16
1.4 Огляд поточкового шифру <i>SNOW-V</i>	20
1.5 Огляд поточкового шифру SNOW-Vi	23
Висновки до розділу 1	26
2 Поточковий шифр Струмок. Загальні поняття атаки часткового вгадування, програмний застосунок Autoguess	27
2.1 Опис поточкового шифру Струмок	27
2.2 Огляд атаки часткового вгадування	31
2.3 Програмні методи пошуку атак часткового вгадування	33
2.4 Зведення задачі пошуку базиса вгадування до задачі пошуку базиса Грьобнера	34
Висновки до розділу 2	35
3 Пошук базисів атак часткового вгадування на поточковий шифр Струмок	36
3.1 Результати пошуку базисів вгадування	37
3.2 Оптимальний алгоритм перебору	45
Висновки до розділу 3	46
Висновки	48
Перелік посилань	49
Додаток А Тексти програм	52
А.1 Python скрипт генерації файлів зв'язків	52
А.2 Файл зв'язків для 11 тактів шифру Струмок	55
А.3 Файл зв'язків для 13 тактів шифру Струмок	57
А.4 Знайдені базиси вгадування для 11 тактів шифра Струмок	59

A.5 Знайдені бази вгадування для 12 тактів шифра Струмок	65 ⁸
A.6 Знайдені бази вгадування для 13 тактів шифра Струмок	71
A.7 Знайдені бази вгадування для 14 тактів шифра Струмок	78
A.8 Знайдені бази вгадування для 15 тактів шифра Струмок	85

ВСТУП

Актуальність дослідження. У 2019 році потоковий шифр Струмок став стандартом симетричного потокового перетворення України ДСТУ:8845, який повинен забезпечувати високу стійкість та швидкість шифрування інформації. Будова шифру Струмок схожа на будову потокового шифру *SNOW-2.0*.

Шифр Струмок описано у діючому стандарті симетричного потокового перетворення України і водночас шифр є малодослідженим. Особливо малодослідженою є стійкість Струмка до атак часткового вгадування.

Метою роботи є аналіз стійкості потокового шифру *Струмок* до атак часткового вгадування.

Об'єктом дослідження є інформаційні процеси в системах криптографічного захисту інформації.

Предметом дослідження є стійкість потокового шифру *Струмок* до атаки часткового вгадування.

Для досягнення мети необхідно виконати такі завдання:

- 1) Дослідити будову поточкових шифрів сімейства *SNOW* та відомі атаки на них.
- 2) Дослідити будову потокового шифру *Струмок* і провести порівняльний аналіз із поточковим шифром *SNOW-2.0*.
- 3) Дослідити методи побудови атак часткового вгадування.
- 4) Побудувати атаку часткового вгадування до шифру *Струмок*.
- 5) Надати рекомендації щодо покращення стійкості потокового шифру Струмок до атак часткового вгадування.

Наукова новизна отриманих результатів полягає в тому, що в роботі знайдено різні базиси вгадування і побудовано атаку часткового вгадування на потоковий шифр *Струмок*.

Апробація результатів на Всеукраїнській науково-практичній

конференції студентів, аспірантів та молодих вчених.

Практичне значення отриманих результатів полягає в тому, що стійкість потокового шифру Струмок не відповідає заявленій, хоча для практичного використання є достатньою.

1 ОСОБЛИВОСТІ БУДОВИ ПОТОКОВИХ ШИФРІВ СІМЕЙСТВА SNOW

У розділі розглянуто шифри сімейства SNOW, їхню структуру і особливості будови.

1.1 Огляд потокового шифру SNOW

У 2000 році було розроблено потоковий шифр SNOW дослідниками Ekdahl P. і Johansson T. [11] Шифр виявився вразливим до деяких типів атак, зокрема до guess-and-determine attack.

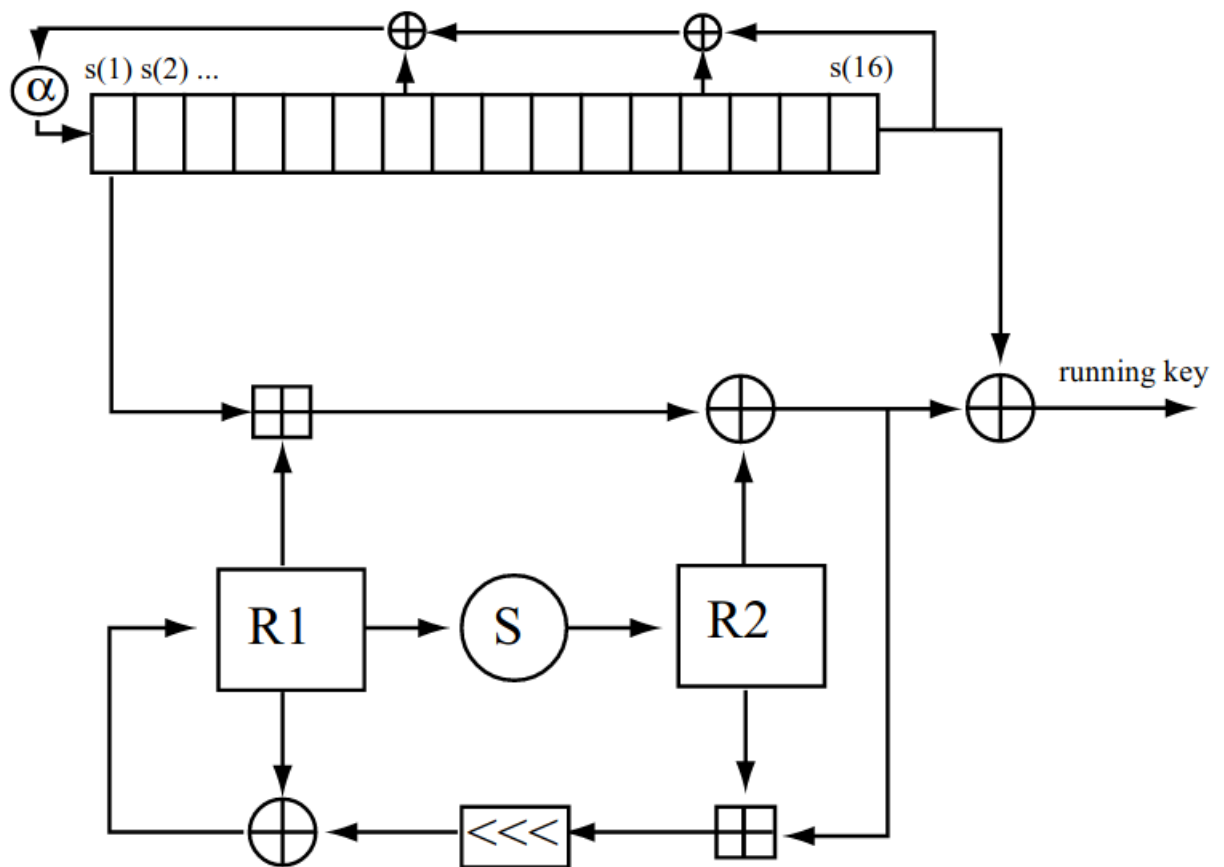


Рисунок 1.1 – Схема роботи потокового шифру *SNOW-1.0*

Шифр SNOW є слово орієнтованим і оперує словами довжиною 32 біти, довжина ключа може становити 128 або 256 бітів. Шифр складається із регістра зсуву з лінійним зворотнім зв'язком та скінченного автомата, який забезпечує нелінійне перетворення. Регістр зсуву складається з 16 комірок, в кожній з яких зберігаються слова довжиною 32 біти. Скінченний автомат складається з двох 32 бітних регістрів R_1 та R_2 і нелінійної підстановки S . FSM приймає на вхід значення з відводів регістра зсуву. $LFSR$ задано над полем $F_{2^{32}}$, зворотній зв'язок якого задано над примітивним поліномом над полем $F_{2^{32}}$

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

Поле $F_{2^{32}}$ задано над незвідним над полем F_2 поліномом

$$\pi(x) = x^{32} + x^{29} + x^{20} + x^{15} + x^{10} + x + 1, \pi(\alpha) = 0.$$

Послідовність з $LFSR$ надходить на вхід FSM , комірки $LFSR$ позначаються як $S_1, \dots, S_{16}$. Значення на виході скінченного автомату обчислюється за співвідношенням

$$FSM_{out} = (S_1 \boxplus_{32} R_1) \oplus R_2,$$

а значення 32-бітних слів шифруючої гами обчислюються за формулою

$$KEYSTREAM = FSM_{out} \oplus S_{16}.$$

Перед початком роботи шифру виконується процедура ініціалізації, яка генерує початкові значення регістрів $LFSR$ та регістрів FSM . Далі обчислюється перше слово ключового потоку, шифр виконує один такт і робить зсув регістра $LFSR$, обчислює новий внутрішній стан FSM за формулою

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

де операція $\lll$ позначає циклічний зсув вліво на 7 комірок.

Нелінійна підстановка S забезпечується розбиттям вхідного слова на 4 байти, до кожного з яких застосовуються однакові підстановки. Після підстановки біти отриманого слова переставляються певним чином і формують остаточний результат нелінійного перетворення S . Детальніше ознайомитись з будовою потокового шифру SNOW можна у роботі [11].

Вразливості потокового шифру SNOW

У 2002 році у роботах [2] описано і успішно застосовано атаку guess - and - determine на потоковий шифр SNOW із обчислювальною складністю 2^{224} операцій та просторовою складністю 2^{95} 32-бітових слів. Через вибір неправильного полінома зворотного зв'язку та лише один вхід до FSM атаку вдалося значно спростити завдяки оберненню операцій у скінченному автоматі.

Лінійне рекурентне рівняння задається співвідношенням

$$S_{t+16} = \alpha(S_{t+9} + S_{t+3} + S_t).$$

Між словами S_t і S_{t+3} маємо відстань у 3 слова, а між словами S_{t+3} і S_{t+9} — 6 слів. Після піднесення цього виразу до другого степеня отримаємо наступне співвідношення

$$S_{t+32} = \alpha^2(S_{t+18} + S_{t+6} + S_t).$$

Суму $(S_{t+i} + S_{t+i+6})$ можна розглядати у якості входу до рівнянь. Криптоаналітику достаньно вказати лише побітову сумму цих слів замість вгадування значення кожного зі слів.

Інша вразливість поліному зворотного зв'язку з'являється якщо піднести його

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

до степеня 2^{32} :

$$p^{2^{32}}(x) = x^{16 \cdot 2^{32}} + x^{13 \cdot 2^{32}} + x^{7 \cdot 2^{32}} + \alpha^{-1 \cdot 2^{32}},$$

$\alpha \in F_{2^{32}}$ тому $\alpha^{-1 \cdot 2^{32}} = \alpha^{-1}$. Якщо додати $p(x)$ і $p^{2^{32}}(x)$, то отримаємо

$$x^{16 \cdot 2^{32}} + x^{13 \cdot 2^{32}} + x^{7 \cdot 2^{32}} + x^{16} + x^{13} + x^7. \quad (1.1)$$

Якщо розділити 1.1 на x^7 , то в результаті буде отримано співвідношення, яке задовольняє рівняння

$$S_{t+16 \cdot 2^{32}-7} + S_{t+13 \cdot 2^{32}-7} + S_{t+7 \cdot 2^{32}-7} + S_{t+9} + S_{t+6} + S_t = 0.$$

Рівняння виконується для будь-якої бітової позиції. Будь-яка побітова кореляція, яка може бути знайдена у скінченному автоматі, може бути звернена до атаки розпізнавання. У роботі [13] знайдена кореляція і атака розрізнення була оцінена авторами роботи у близько 2^{100} операцій та приблизно 2^{95} 32-бітових слів.

У [11] також було застосовано комп'ютер для пошуку кореляцій. Знайдені кореляції виявилися слабшими, але шифр містив дуже багато слабких місць. Кореляції викликані нелінійною підстановкою S та циклічним зсувом бітового рядка вліво на сім позицій $\lll 7$ у скінченному автоматі.

1.2 Огляд потокового шифру SNOW 2.0

Потоковий шифр SNOW 2.0 розроблено у 2003 році [12]. SNOW 2.0 має підвищену криптографічну стійкість та швидкість роботи у порівнянні із першою версією SNOW. Не дивлячись на підвищення стійкості та швидкодії шифру концептуальних змін будови не відбулося. Регістр зсуву *LFSR* складається із 16 машинних слів, кожне слово має довжину 32 біта. Скінченний автомат *FSM* також складається із двох

32-бітних регістрів R_1 та R_2 . У автоматі FSM відбувається нелінійне перетворення T . На відміну від шифру SNOW FSM шифру SNOW 2.0 приймає на вхід два слова з регістру зсуву $LFSR$, який визначено над полем $F_{2^{32}}$.

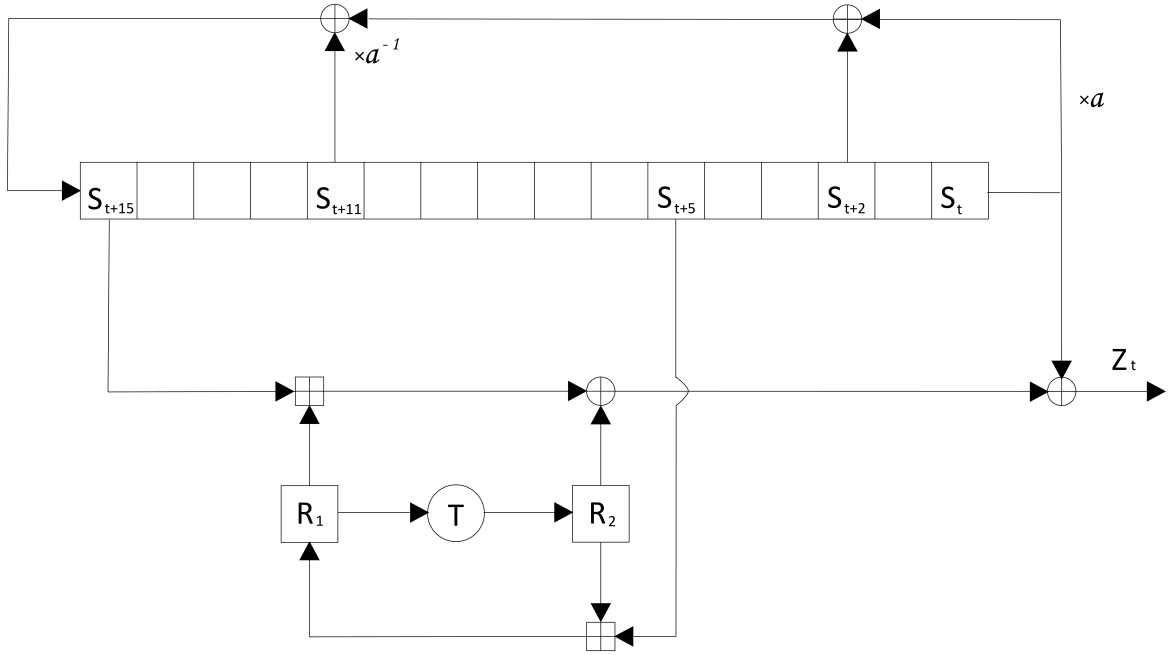


Рисунок 1.2 – Схема роботи потокового шифру SNOW 2.0 у момент часу t

Поліном зворотного зв'язку задається як

$$f(x) = \alpha x^{16} + x^{14} + \alpha^{-1} x^5 + 1,$$

де $\alpha \in F_{2^{32}}$ це корінь поліному над полем F_{2^8}

$$g(z) = z^4 + \beta^{23} z^3 + \beta^{245} z^2 + \beta^{48} z + \beta^{239},$$

де β — корінь примітивного над полем F_2 поліному

$$p(y) = y^8 + y^7 + y^5 + y^3 + 1.$$

. Вхід до FSM складається із двох машинних слів. Така зміна дозволила підвищити криптографічну стійкість SNOW 2.0. Стан $LFSR$ у довільний момент часу $t \geq 0$ позначається як $(S_{t+15}, S_{t+14}, S_{t+13}, \dots, S_{t+2}, S_{t+1}, S_t)$, де S_{t+i} це елементи поля Галуа $F_{2^{32}}$, $i \geq 0$. Значення на виході FSM обчислюються за правилом

$$FSM_{out} = (S_{t+15} \boxplus_{32} R_{1_t}) \oplus R_{2_t}, t \geq 0,$$

Після обчислення кожного слова гами стан FSM оновлюється:

$$\begin{aligned} R_{1_{t+1}} &= S_{t+5} \boxplus_{32} R_{2_t}, \\ R_{2_{t+1}} &= S(R_{1_t}), t \geq 0. \end{aligned}$$

Вихід гами шифру обчислюється аналогічно до шифру попередньої версії, а саме за співвідношенням

$$z_t = FSM_{out} \oplus S_t, t \geq 1.$$

У якості S -блока у скінченному автоматі FSM використовується раундова функція шифру *Rijndael*[14]. Ініціалізація SNOW 2.0 виконується за 32 такти, під час яких шифр не генерує ключовий потік. Із правилами заповнення початкових значень регістра зсуву $LFSR$ та скінченного автомату FSM можна ознайомитись детально у роботі [12].

1.3 Огляд потокового шифру *SNOW-3G*

SNOW-3G розроблено у 2006 році. Подібно до усіх попередніх шифрів сімейства SNOW цей шифр складається із регістра зсуву з лінійним зворотним зв'язком $LFSR$ і скінченного автомату FSM [15]. На відміну від попередників потоковий шифр *SNOW-3G* має скінченний автомат, який складається не з двох регістрів, а з трьох. Також FSM має два S -блоки. Такі незначні на перший погляд зміни дозволили підвищити

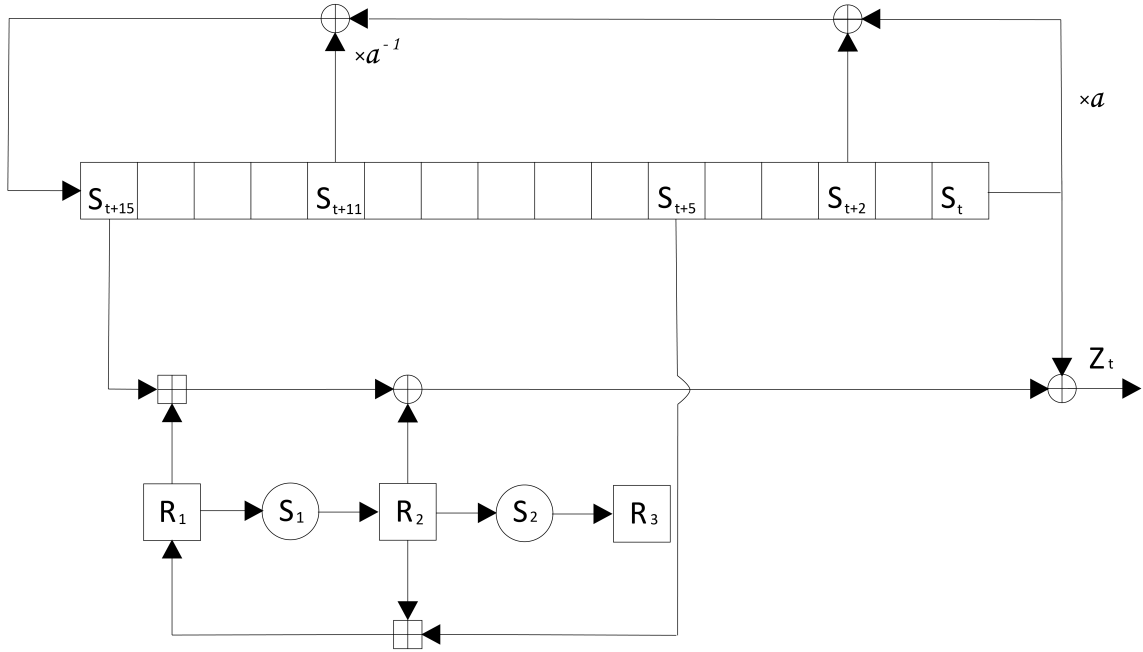


Рисунок 1.3 – Схема роботи потокового шифру *SNOW-3G* у момент часу t

стійкість до атак *guess and determine* [15]. *SNOW-3G* оперує машинними словами довжиною 32 біта, це дозволяє забезпечити високу швидкодію шифру як в програмній, так і в апаратній реалізації. Шифр приймає на вхід секретний ключ K та вектор ініціалізації IV . Довжина параметрів K та IV становить 128 бітів.

LFSR шифру як і в попередників складається з 16 комірок і задається поліномом

$$p(x) = \alpha x^{16} + x^{14} + \alpha^{-1} x^5 + 1 \in F_{2^{32}}[x],$$

де α — це корінь поліному

$$x^4 + \beta^{23} x^3 + \beta^{245} x^2 + \beta^{48} x + \beta^{239} \in F_{2^8}[x],$$

де β — це корінь поліному

$$x^8 + x^7 + x^5 + x^3 + 1 \in F_2[x].$$

У довільний момент часу t стан $LFSR$ визначається як $(S_{0_t}, \dots, S_{15_t})$, $S_{15_{t+1}}$ на кожному такті визначається за формулою

$$S_{15_{t+1}} = \alpha^{-1} S_{11_t} \oplus S_{2_t} \oplus \alpha S_{0_t}, \alpha \in F_{2^{32}}.$$

FSM шифра складається із трьох 32-бітових регістрів. Слова S_{15} та S_5 з $LFSR$ надходять до FSM , який в свою чергу генерує 32-бітове слово внаслідок обчислення за формулою

$$z_t = (R_{1_t} \boxplus_{32} S_{15_t}) \oplus R_{2_t} \oplus S_{0_t}.$$

Значення регістрів скінченного автомата оновлюються на кожному такті за формулами

$$R_{2_{t+1}} = S_1(R_{1_t}),$$

$$R_{3_{t+1}} = S_2(R_{2_t}),$$

$$R_{1_{t+1}} = R_{2_t} \boxplus_{32} (R_{3_t} \oplus S_{5_t}),$$

де перетворення S_1, S_2 — це S -блоки.

Перетворення S_1 задано як перетворення S у шифрі SNOW 2.0. S_2 — це відображення 32 бітів у 32 біти, яке виконує операції над чотирма байтами. Байти інтерпретуються як елементи поля F_{2^8} над поліномом

$y^8 + y^6 + y^5 + y^3 + 1$. S_2 описується співвідношенням

$$\begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{pmatrix} = \begin{pmatrix} y & y+1 & 1 & 1 \\ 1 & y & y+1 & 1 \\ 1 & 1 & y & y+1 \\ y+1 & 1 & 1 & y \end{pmatrix} \cdot \begin{pmatrix} S_Q(w_0) \\ S_Q(w_1) \\ S_Q(w_2) \\ S_Q(w_3) \end{pmatrix},$$

де (w_3, w_2, w_1, w_0) — це байти 32-бітного слова на вході перетворення, а (r_3, r_2, r_1, r_0) — байти на виході. 8-бітове перетворення S_Q отримано із поліномів Діксона [15]. У скороченій формі формулу можна записати як

$$S_2(W) = L_2 \cdot S_Q(W),$$

$$L_2 = \begin{pmatrix} y & y+1 & 1 & 1 \\ 1 & y & y+1 & 1 \\ 1 & 1 & y & y+1 \\ y+1 & 1 & 1 & y \end{pmatrix}.$$

На початку роботи шифр приймає на вхід секретний ключ K та вектор ініціалізації IV і розпочинає процедуру ініціалізації. Заповнення $LFSR$ значеннями відбувається за формулами

$$\begin{aligned} S_{15} &= K_3 \oplus IV_0, S_{14} = K_2, S_{13} = K_1, S_{12} = K_0 \oplus IV_1, \\ S_{11} &= \neg K_3, S_{10} = \neg K_2 \oplus IV_2, S_9 = \neg K_1 \oplus IV_3, S_8 = \neg K_0, \\ S_7 &= K_3, S_6 = K_2, S_5 = K_1, S_4 = K_0, \\ S_3 &= \neg K_3, S_2 = \neg K_2, S_1 = \neg K_1, S_0 = \neg K_0. \end{aligned}$$

Регістри FSM заповнюються нульовими значеннями. Після ініціалізації шифр робить один холостий такт без генерації слів ключового потоку.

1.4 Огляд потокового шифру $SNOW-V$

$SNOW-V$ – це потоковий шифр сімейства $SNOW$ нового покоління. Вперше шифр побачив світ у 2018 році [8]. Як і всі шифри-попередники $SNOW-V$ претендує на задовільнення потреб високошвидкодкісного шифрування та високої криптографічної стійкості. Як зазначили автори розробленого шифру, $SNOW-V$ є подальшим розвиненням потокового шифру $SNOW-3G$ [8]. Архітектура нового шифру дозволяє використовувати весь потенціал сучасних процесорів для підвищення швидкості шифрування, а саме шифр підтримує інструкції прискорення обчислення функції AES та інструкції $SIMD$ (single instruction, multiple data) [8].

$SNOW-V$ успадкував основні принципи побудови від своїх попередників, а саме регістр зсуву з лінійним зворотним зв'язком $LFSR$ та скінченний автомат FSM із нелінійним перетворенням, але ці компоненти було оновлено таким чином, щоб використання векторизованих операцій забезпечувало більш швидку роботу шифру [8]. Стійкість шифру підвищено завдяки використанню збільшеного внутрішнього стану за рахунок використання 128-бітових регістрів FSM та повної функції шифрування шифру AES . Таким чином потоковий шифр $SNOW-V$ є швидшим та більш стійким за блоковий шифр AES [8].

На відміну від попередніх шифрів сімейства шифр $SNOW-V$ має два регістри зсуву: $LFSR-A$ та $LFSR-B$. Регістри складаються з 16 комірок, які заповнюються 16-бітовими словами. Комірки регістрів позначаються $a_{15}, \dots, a_0$ і $b_{15}, \dots, b_0$ відповідно. Поліноми зворотного

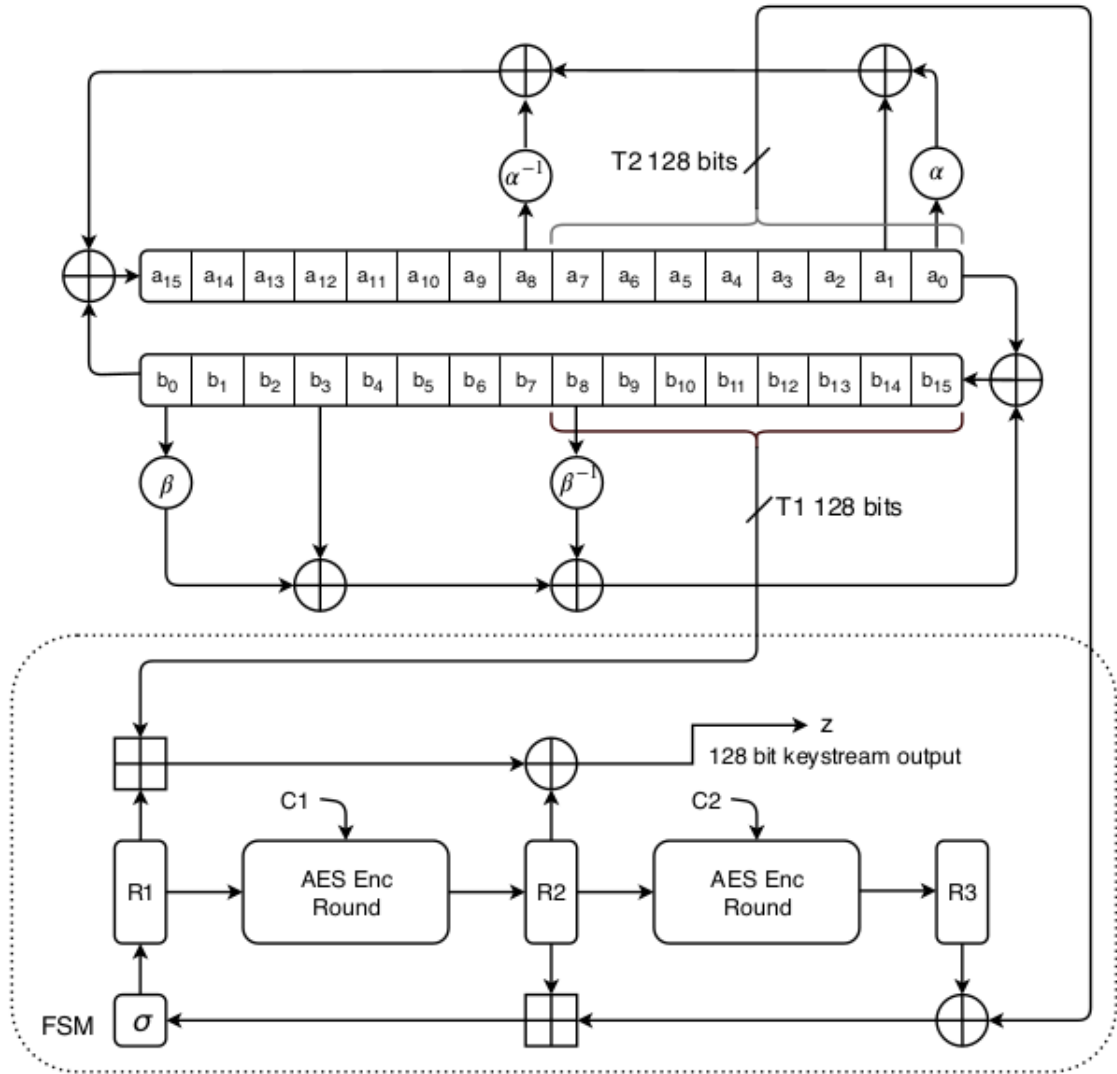


Рисунок 1.4 – Схема роботи шифру *SNOW-V*

зв'язку регістрів мають наступний вигляд:

$$g^A(x) = x^{16} + x^{15} + x^{12} + x^{11} + x^8 + x^3 + x^2 + x + 1 \in F_2[x],$$

$$g^B(x) = x^{16} + x^{15} + x^{14} + x^{11} + x^8 + x^6 + x^5 + x + 1 \in F_2[x]$$

для *LFSR-A* та *LFSR-B* відповідно.

$\alpha \in F_{2^{16}}^A$ – це корінь $g^A(x)$, а $\beta \in F_{2^{16}}^B$ – це корінь поліному $g^B(x)$, внутрішній стан регістрів зсуву у довільний момент часу $t \geq 0$ позначається

$$(a_{15}^{(t)}, \dots, a_0^{(t)}) \text{ та } (b_{15}^{(t)}, \dots, b_0^{(t)}), a_i^{(t)} \in F_{2^{16}}^A, b_i^{(t)} \in F_{2^{16}}^B$$

для *LFSR-A* і *LFSR-B* відповідно, слова $a_0^{(t)}$ і $b_0^{(t)}$ є першими на вихід у момент часу t .

LFSR-A і *LFSR-B* генерують послідовності $a^{(t)}, b^{(t)}, t \geq 0$, що описуються співвідношеннями

$$\begin{aligned} a^{(t+16)} &= b^{(t)} + \alpha a^{(t)} + a^{(t+1)} + \alpha^{-1} a^{(t+8)} \bmod g^A(\alpha), \\ b^{(t+16)} &= a^{(t)} + \beta b^{(t)} + b^{(t+3)} + \beta^{-1} b^{(t+8)} \bmod g^B(\beta). \end{aligned}$$

Слова T_1 і T_2 надходять до *FSM*, де $T_1 = (b_{15}, \dots, b_8)$, а $T_2 = (a_7, \dots, a_0)$. Регістри скінченного автомату мають довжину 128 бітів. Операція $\boxplus_{32}$ позначає паралельне застосування чотирьох операцій додавання за модулем 2^{32} для кожного підслова. Ключовий потік $z^{(t)}$ на виході *FSM* обчислюється за наступним правилом

$$z^{(t)} = (R_1^{(t)} \boxplus_{32} T_1^{(t)}) \oplus R_2^{(t)}, t \geq 0.$$

Оновлення стану регістрів R_2 та R_3 відбувається за рахунок використання функції повного раунду шифрування *AES* [8].

Під час процедури ініціалізації шифр приймає на вхід ключ $K = (k_{15}, \dots, k_0)$ та вектор ініціалізації $IV = (iv_7, \dots, iv_0)$, де кожен компонент є словом довжиною 16 бітів. На початку процедури ініціалізації відбувається заповнення регістрів зсуву початковими значеннями за правилом:

$$\begin{aligned} (a_{15}, \dots, a_0) &= (k_7, \dots, k_0, iv_7, \dots, iv_0), \\ (b_{15}, \dots, b_0) &= (k_{15}, \dots, k_8, 0, \dots, 0). \end{aligned}$$

Після заповнення регістрів початковими значеннями шифр працює впродовж 16 тактів із зацикленням ключового потоку назад до регістрів зсуву. Останні два раунди ініціалізації ключ побітово додається за модулем двійки до регістру R_1 [8].

1.5 Огляд потокового шифру SNOW-Vi

Будова SNOW-Vi у частині генерації потоку ключів і процедура ініціалізації точно така ж, як і в SNOW-V. Відмінність полягає лише у функції оновлення LFSR і положенні відводів, яке тепер переміщено у вищу половину LFSR-A — ці зміни значно підвищують швидкість програмних реалізацій та підвищують захист шифру. Новий LFSR зображено на рисунку нижче, а нові відводи зворотного зв'язку такі:

$$\begin{aligned}a^{t+16} &= b^t + \alpha a^t + a^{t+7} \bmod g^A(\alpha), \\ b^{t+16} &= a^t + \beta b^t + b^{t+8} \bmod g^B(\beta),\end{aligned}$$

де α і β відповідно корені поліномів, які визначаються наступним чином:

$$\begin{aligned}g^A(x) &= x^{16} + x^{14} + x^{11} + x^9 + x^6 + x^5 + x^3 + x^2 + 1 \in F_2[x], \\ g^B(x) &= x^{16} + x^{15} + x^{14} + x^{11} + x^{10} + x^7 + x^2 + x + 1 \in F_2[x].\end{aligned}$$

128-бітовий дизайн. Для сучасних шифрів, які мають працювати швидко в програмному середовищі, аспекти реалізації дуже важливі для досягнення високої пропускну здатності. SNOW 3G — це 32-бітовий шифр, який породжує 32-бітове слово ключового потоку за такт.

Щоб пришвидшити SNOW 3G, можна було б створити чотири такі 32-бітові ключові слова паралельно із використанням 128-бітового регістра — великі регістри підтримуються багатьма сучасними процесорами. Це означає, що блок FSM також повинен містити чотири паралельні 32-бітові слова, щоб він міг породжувати 128-бітові слова на виході. LFSR може зберегти свій розмір, але його слід переробити, щоб оновлювати щонайменше 128 бітів за раз. Це фактично призводить до

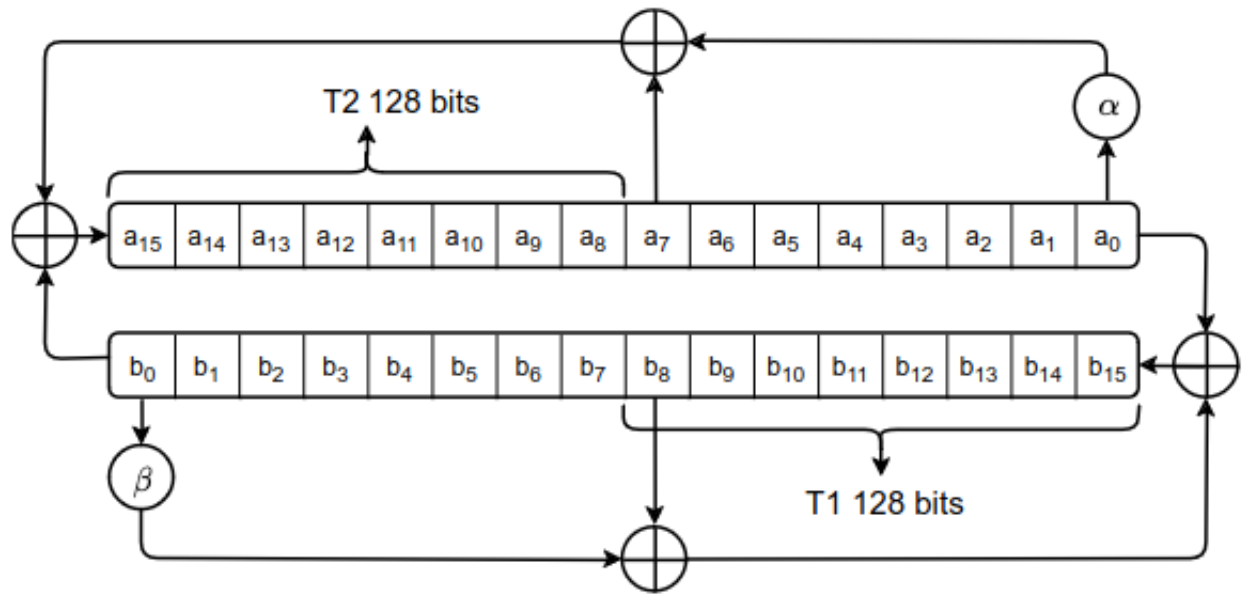


Рисунок 1.5 – Схема роботи шифру *SNOW-VI*

128-бітових дизайнів оригінального SNOW-V і варіанту SNOW-Vi, в якому FSM містить три 128-бітових слова, LFSR можна розділити на чотири 128-бітових слова, а згенероване слово ключового потоку має довжину 128 бітів.

Довжина циклу. У сімействі поточкових шифрів SNOW LFSR є лінійним блоком довжиною 512 бітів, який служить джерелом псевдовипадковості. Однією з важливих властивостей безпеки є те, що LFSR не повинен мати коротких циклів і, в ідеалі, має максимальну довжину циклу $2^{512} - 1$, за винятком нульового стану. Запропонований LFSR має таку максимальну довжину циклу, яку можна перевірити тими ж методами, що й у [9, 8]. Характеристичний поліном є примітивним і має 209 членів.

LFSR. Добре відомо, що для запобігання тривіальним лінійним атакам із кількома короткими потоками ключів кількість відводів t , які будуть використовуватися для функції оновлення LFSR, має бути більше трьох, а краще навіть більше (близько 5 – 6) залежно від того, як добре

можна наблизити FSM. Однак усі значення відводів потрібно обробити у функції оновлення LFSR, що означає, що складність коду та часу лінійно зростає з кількістю відводів. У кільцевій конструкції LFSR ми маємо два суб-LFSR із залученими t_a та t_b відводами відповідно. Загальна кількість відводів дорівнює $t_a + t_b$, але існує мультиплікативний ефект для кількості відводів для еквівалентного LFSR, який можна використовувати в лінійній атаці. Еквівалентний LFSR може мати до $(1 + t_a) \cdot (1 + t_b)$ кранів. Таким чином, ми можемо досягти основних цілей безпеки, маючи меншу кількість відводів, і, отже, обчислювальна складність для синхронізації LFSR менша. Тобто, якщо класичний LFSR має t відводів, циклічний LFSR може мати лише $2 \cdot (\sqrt{t} - 1)$ відводи для досягнення того самого рівня безпеки, таким чином зменшуючи складність реалізації та часову складність для функції оновлення. Наприклад, у SNOW-V загальна кількість відводів становить $3 + 3$, але еквівалентний LFSR має 11 відводів, тому замість виконання операцій на 11 відводах у класичному LFSR потрібно працювати лише з шістьма кранами в кільцевій конструкції LFSR.

Атака часткового вгадування

У атаках часткового вгадування вгадується частина стану шифра та з рівнянь оновлення стану визначаються інші частини. Мета – вгадати якомога менше бітів, а потім визначити якомога більше бітів за допомогою заданих рівнянь. Рівняння $z^t = (R1_t \boxplus_{32} T1^t) \oplus R2^t$ містить три невідомі, розмір кожного з яких становить 128 бітів. Треба вгадати значення двох із них (256 бітів), щоб визначити решту. Дивлячись на рівняння для наступного слова гами, потрібно вгадати ще 128 бітів. Це свідчить про те, що атака часткового вгадування на потоковий шифр SNOW-Vi все ще має велику складність. Пряма атака часткового вгадування наведена в [9], яка вимагає вгадування 512 бітів у трьох послідовних словах ключового потоку для відновлення повних 896 бітів стану. В роботі [10] описано атаку часткового вгадування на байтову версію шифру SNOW-V зі складністю 2^{406} за допомогою семи ключових

слів. Під час атаки регістри в LFSR і FSM розбиваються на байти, а операції оновлення відповідно перетворюються на байтові.

Остаточний початковий набір вгадувань, використаний у [10], має 24-байтові змінні, і всі ці змінні походять із регістрів FSM або старших половин регістрів LFSR, тоді як змінні, які вибрано для оновлення, не використовуються. У процесі вгадування вгадуються ще 12 байтів із регістрів FSM і ще 13 байтів з LFSR. Оскільки в SNOW-V є три відводи для LFSR-A і LFSR-B, а в SNOW-Vi – два, ми робимо найгірше припущення, що коли для вгадування в SNOW-V потрібно 13 байтів у LFSR, потрібно лише близько 8 байтів в SNOW-Vi. У цьому випадку потрібно вгадати $24 + 12 + 8 = 44$ байти, тобто 352 біти. Крім цих байтів, потрібно вгадати деякі додаткові біти переносу. Таким чином, складність атаки типу часткового вгадування на потоковий шифр SNOW-Vi перевищує 2^{352} . Ми можемо зробити ще гірше припущення, що вгадані змінні в LFSR можуть бути вільно отримані, в результаті чого вгадуються $24 + 12 = 36$ байтів, усі з регістрів FSM, тобто 288 бітів, для яких складність все ще перевищує 2^{256} . Таким чином, ми робимо висновок, що атака часткового вгадування не буде швидшою, ніж вичерпний пошук ключа проти SNOW-Vi.

Висновки до розділу 1

У розділі описано потокові шифри сімейства SNOW, розглянуто особливості будови кожного із представників, прослідковано еволюцію будови шифрів сімейства. Будова поточкових шифрів із використанням *LFSR* та *FSM* із нелінійним перетворенням отримала назву SNOW-подібної. Внаслідок високих показників швидкості шифрування інформації у сукупності із високим рівнем захисту ідея будови шифрів SNOW успішно використовується багатьма розробниками поточкових шифрів завдяки тому, що така будова добре вивчена та перевірена часом.

2 ПОТОКОВИЙ ШИФР СТРУМОК. ЗАГАЛЬНІ ПОНЯТТЯ АТАКИ ЧАСТКОВОГО ВГАДУВАННЯ, ПРОГРАМНИЙ ЗАСТОСУНОК AUTOGUESS

У розділі описано шифр Струмок, детально описано його будову та процедури, описано загальні принципи побудови атак часткового вгадування, розглянуто програмний застосунок для пошуку атак часткового вгадування під назвою Autoguess, застосовано Autoguess для пошуку та аналізу атаки часткового вгадування на шифр Струмок.

2.1 Опис потокового шифру Струмок

За своєю будовою шифр є майже повною копією потокового шифру SNOW-2.0, але відрізняється від нього тим, що оперує словами довжиною 64 біта. Струмок використовує вектор ініціалізації IV довжиною 256 бітів, секретний ключ K може мати довжину 256 або 512 бітів.

Оскільки Струмок є SNOW-подібним шифром, то він складається із регістру зсуву з лінійним зворотним зв'язком та скінченного автомату з двома 64-бітовими регістрами, де відбувається нелінійне перетворення. Шифр генерує слова довжиною 64 біта. Поліном зворотного зв'язку $f(x)$ є примітивним поліномом над полем $GF(2^{64})$

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

де α – це корінь полінома $g(z)$, примітивного над полем $GF(2^8)$

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

β – це корінь полінома $p(y)$, який є примітивним над полем $GF(2)$, за яким

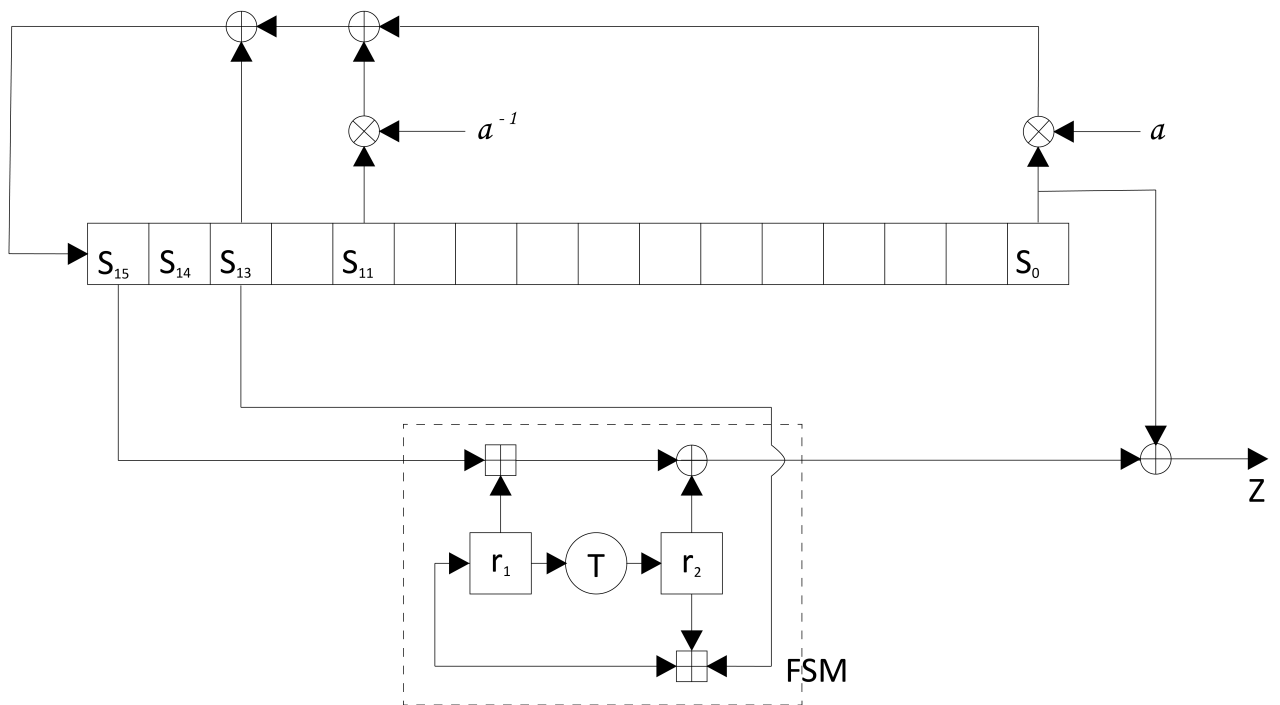


Рисунок 2.1 – Схема потокового шифру Струмок

будується поле $GF(2^8)$

$$p(y) = y^8 + y^4 + y^3 + y^2 + 1.$$

Шифр Струмок можна представити у вигляді процедур як і всі інші шифри. Нижче детально розглянуто процедури та описано їхню роботу. Згідно з [1] потоковий шифр Струмок структурно поділяється на такі процедури:

- **Init** – процедура ініціалізації шифру
- **Next** – процедура оновлення внутрішнього стану
- **Strm** – процедура генерації ключового потоку
- **FSM** – процедура оновлення стану скінченного автомата

Процедура **Init** приймає на вхід ключ K та вектор ініціалізації IV . Результатом виконання процедури є початковий стан шифру $S_0(s^{(0)}, r^{(0)})$. Ключ в залежності від його довжини представляється у вигляді чотирьох або восьми слів довжиною 64 біта кожне. Вектор ініціалізації

представляється у вигляді чотирьох слів. Спочатку процедура заповнює *LFSR* початковими значеннями за правилами

$$\begin{aligned}
s_{15}^{(-33)} &= \neg K_0, s_{14}^{(-33)} = K_1, \\
s_{13}^{(-33)} &= \neg K_2, s_{12}^{(-33)} = K_3, \\
s_{11}^{(-33)} &= K_0, s_{10}^{(-33)} = \neg K_1, \\
s_9^{(-33)} &= K_2, s_8^{(-33)} = K_3, \\
s_7^{(-33)} &= \neg K_0, s_6^{(-33)} = \neg K_1, \\
s_5^{(-33)} &= K_2 \oplus IV_3, s_4^{(-33)} = K_3, \\
s_3^{(-33)} &= K_0 \oplus IV_2, s_2^{(-33)} = K_1 \oplus IV_1, \\
s_1^{(-33)} &= K_2, s_0^{(-33)} = K_3 \oplus IV_0
\end{aligned}$$

для ключа довжиною 256 бітів або за правилами

$$\begin{aligned}
s_{15}^{(-33)} &= K_0, s_{14}^{(-33)} = \neg K_1, \\
s_{13}^{(-33)} &= K_2, s_{12}^{(-33)} = K_3, \\
s_{11}^{(-33)} &= \neg K_7, s_{10}^{(-33)} = K_5, \\
s_9^{(-33)} &= \neg K_6, s_8^{(-33)} = K_4 \oplus IV_3, \\
s_7^{(-33)} &= \neg K_0, s_6^{(-33)} = K_1, \\
s_5^{(-33)} &= K_2 \oplus IV_2, s_4^{(-33)} = K_3, \\
s_3^{(-33)} &= K_4 \oplus IV_1, s_2^{(-33)} = K_5, \\
s_1^{(-33)} &= K_6, s_0^{(-33)} = K_7 \oplus IV_0
\end{aligned}$$

у випадку використання ключа довжиною 512 бітів. Після заповнення комірок шифр виконує 32 такти без генерації гами

$$S_{-1} = Next^{32}(S_{-33}, Init),$$

процедура **Next** працює у режимі ініціалізації. Далі обчислюється початкове значення змінної стану $S_0 = (s^{(0)}, r^{(0)})$, $S_0 = Next(S_{-1})$. Процедура ініціалізації завершує роботу та повертає початкове значення

змінної внутрішнього стану шифру. Процедура ініціалізації графічно представлена на рис.2.2.

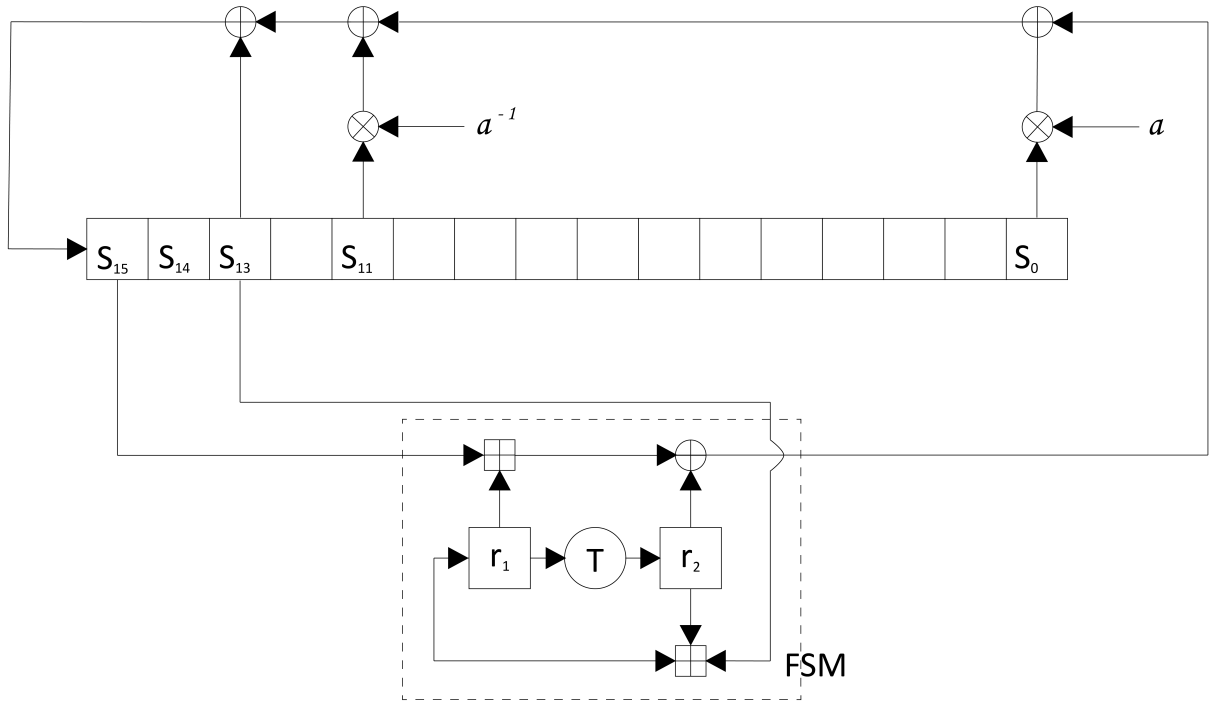


Рисунок 2.2 – Ініціалізація шифру Струм

Процедура **Next** приймає на вхід змінну стану $S_i(s^{(i)}, r^{(i)})$, обчислює та повертає наступне значення змінної стану $S_{i+1}(s^{(i+1)}, r^{(i+1)})$. Процедура може працювати в двох режимах: звичайному режимі та режимі ініціалізації. В процесі виконання процедури перш за все відбувається обчислення та оновлення значення регістру скінченного автомату $r_2^{(i+1)} = T(r_1^{(i)})$, після оновлення другого регістра оновлюється значення першого регістра $r_1^{(i+1)} = r_2^{(i)} \boxplus_{64} s_{13}^{(i)}$ і значення 15 комірок *LFSR* за формулою $s_j^{(i+1)} = s_{j+1}^{(i)}, j \in [0, 14]$. Значення останньої комірки *LFSR* оновлюється за формулою $s_{15}^{(i+1)} = (s_0^{(i)} \otimes \alpha) \oplus (s_{11}^{(i)} \otimes \alpha^{-1}) \oplus s_{13}^{(i)}$, якщо процедура викликана у звичайному режимі, або за правилом $s_{15}^{(i+1)} = FSM(s_{15}^{(i)}, r_1^{(i)}, r_2^{(i)}) \oplus (s_0^{(i)} \otimes \alpha) \oplus (s_{11}^{(i)} \otimes \alpha^{-1}) \oplus s_{13}^{(i)}$, якщо процедура виконується у режимі ініціалізації. Процедура на виході повертає значення стану S_i .

Нелінійна підстановка T . Підстановки T виконує перестановку згідно із стандартом *ДСТУ 7624:2014*. На вхід надходить слово довжиною 64 біта, слово розбивається на вісім байтів, кожен із яких проходить через підстановку, що описана у стандарті *ДСТУ 7624:2014*. Результат перетворення домножується на матрицю байтів T . З метою пришвидшення роботи шифру таблицю підстановок та матрицю T рекомендовано передобчислити.

Процедура **Strm** отримує на вхід змінну стану $S_i(s^{(i)}, r^{(i)})$, обчислює та повертає ключовий потік $Z_i = FSM(s_{15}^{(i)}, r_1^{(i)}, r_2^{(i)}) \oplus s_0^{(i)}$.

Процедура **FSM** отримує на вхід три значення: x, y, z і повертає $q = (x \boxplus_{64} y) \oplus z$. Детальніше з будовою шифру можна ознайомитися у роботах [1].

2.2 Огляд атаки часткового вгадування

У атаці типу guess-and-determine значення одних змінних вгадуються, а значення інших визначається згідно з деякими попередньо відомими функціональними зв'язками між регістрами та комірками регістрів шифру. Якщо весь внутрішній стан шифру у деякий момент часу може бути відновлений через вгадування кількості n бітів, де n є меншим за рівень безпеки, тоді атака буде швидшою, ніж перебір усіх можливих значень.

Множина змінних, вгадування значень яких призвело до визначення усього внутрішнього стану, називається базисом вгадування (англ. guess basis). Знання всього стану потокового шифру в певний момент часу дозволяє відновити весь ключовий потік, який відповідає конкретному секретному ключу K та публічному вектору ініціалізації IV .

Якщо процедура ініціалізації шифру не використовує необоротні перетворення, то можна навіть відновити значення секретного ключа або початкове заповнення регістрів шифру. Зазвичай складність атаки часткового вгадування обчислюється як 2^n , якщо просто виконати цикл

для перебору всіх можливих значень обраних для вгадування змінних. Однак вгадуючи змінні в певному порядку можна або вгадати меншу кількість змінних, або скоротити деякі шляхи вгадування (шляхи розповсюдження знань), у яких уже відомі (або вгадані, або визначені) змінні не задовольняють деяким обмеженням рівняння в середині. В такому випадку ми можемо негайно зупинити процес вгадування, не йдучи далі, і повернутися до вгадування іншого значення, таким чином можна зменшити обчислювальну складність за рахунок збільшення просторової складності атаки [3].

Алгоритм атаки часткового вгадування частіше за все відрізняється для кожного шифру [4]. Методи, які добре працюють для одних шифрів, можуть працювати гірше для інших шифрів. Але не дивлячись на це є деякі рекомендації щодо побудови атак, а саме запропоновано евристичну GnD атаку [4] та динамічну атаку [3] із розбиттям великих процедур на менші. Але головна ідея атаки завжди залишається незмінною. Різні методи дозволяють лише зменшити складність атаки у певних випадках. У роботі [6] запропоновано узагальнений підхід пошуку базисів вгадування і розроблено програмний застосунок для пошуку і аналізу можливих атак під назвою Autoguess. Проблема пошуку базисів вгадування можна представити у вигляді задачі для моделей обчислень SAT, SMT, MILP, CP та інших [6].

Атка часткового вгадування є одним із найпоширеніших методів криптоаналізу для відновлення невідомих змінних у внутрішньому стані шифру. У таких атаках підмножина невідомих змінних вгадується таким чином, що значення решти невідомих можна визначити, використовуючи інформацію із вгаданих змінних і наведених алгебраїчних співвідношень.

Ця ідея застосовується в різних сферах криптоаналізу, таких як пошук внутрішнього стану поточкових шифрів, коли доступна достатня кількість вихідних даних, або відновлення внутрішнього стану та секретного ключа блокового шифру з дуже малої кількості відомих відкритих текстів [6].

Оскільки складність атаки часткового вгадування безпосередньо залежить від кількості вгадуваних змінних, то дуже важливо знайти найменший можливий базис вгадування, тобто підмножину вгадуваних змінних, із якої можна встановити значення всіх інших змінних шифру.

2.3 Програмні методи пошуку атак часткового вгадування

Програму Autoguess розроблено для автоматизованого пошуку базисів атак часткового вгадування, а також для пошуку та аналізу складності атак типу *key-bridge* на блокові шифри. Даний програмний застосунок спочатку зводить задачу пошуку базисів вгадування до задач у моделях обчислень SAT, SMT, пошуку базиса Грьобнера та інших, в наслідок вирішення цих задач криптоаналітик отримує базис вгадування. Autoguess використовує високоефективні наукові математичні бібліотеки такі як Sagemath, SAT, SMT, PuLP, MiniZinc та інші. Завдяки використанню таких ефективних бібліотек пошук наявних атак часткового вгадування із заданими параметрами займає досить мало часу.

Autoguess розроблявся на платформі Debian, тому встановити його можна лише на цю платформу, але завдяки використанню віртуалізації, яку надає Docker, Autoguess можна встановити та використовувати на будь-якій сучасній операційній системі, у середовищі якої встановлено Docker. На сторінці програми описано процес встановлення і використання Autoguess, наведено багато прикладів застосування фреймворка для пошуку базисів вгадування на реальних шифрах. Дане програмне забезпечення надає дуже багато налаштувань для криптоаналітика, завдяки яким можна моделювати різні ситуації. Результати пошуку базиса вгадування можуть бути представлені у графічному вигляді, що дуже сильно спрощує розуміння атаки та шляху розповсюдження знань. Завдяки своїй простоті у встановленні і використанні, швидкодії, великій кількості налаштувань та моделей обчислень, графічному відображенню отриманих результатів прийняте

рішення про використання фреймворка Autoguess для пошуку атак часткового вгадування на потоковий шифр Струмок.

2.4 Зведення задачі пошуку базиса вгадування до задачі пошуку базиса Грьобнера

У роботі далі використано алгоритм пошуку базиса Грьобнера для розв'язання задачі пошуку базиса вгадування. Тому доцільно буде описати як працює цей метод.

На відміну від підходів на основі СР для розв'язання проблеми часткового вгадування, у яких глибина поширення знань повинна бути визначена на початку роботи алгоритма пошуку, Даннер і Кройцер у запропонували новий алгебраїчний підхід для моделювання задач часткового вгадування, в якій не тільки немає необхідності вказувати глибину пошуку, але й гарантовано знайти базис вгадування мінімальної довжини.

У цьому розділі описано метод, представлений у роботі [19] для зведення задачі часткового вгадування до задачі обчислення скороченого базиса Грьобнера ідеалу булевого полінома. У роботі [6] розширено модель із роботи [19], щоб врахувати цільові змінні та відомі змінні.

Згідно з [19], система зв'язків транслюється до біноміального ідеалу так, що його скорочений базис Грьобнера включає принаймні один базис вгадування мінімальної довжини. Таким чином, цей підхід має два важливих кроки, включаючи переклад зв'язків у систему булевих поліномів і наступне обчислення скороченого базису Грьобнера.

Дано систему зв'язків (X, R) , де $X = \{x_0, \dots, x_{n-1}\}$, можемо вважати, що всі відношення є відношеннями імплікації. Для кожного відношення імплікації, такого як $x_{i_0}, \dots, x_{i_{m-2}} \longrightarrow x_{i_{m-1}}$, ми замінюємо кожну змінну x_{i_k} на булеву змінну X_{i_k} , яка показує, чи відома вона. Це дає логічну формулу $(\neg X_{i_0} \vee \dots \vee \neg X_{i_{m-2}} \vee X_{i_{m-1}})$. У математичній логіці

така логічна формула, що містить щонайбільше один додатний літерал, називається виразом (реченням, пунктом) Горна та сполучення кількох виразів Горна називається формулою Горна, яка є особливим видом CNF. Відповідно систему зв'язків можна перевести у формулу Горна.

Далі формула Горна перекладається на алгебраїчну мову. Нехай $\mathbb{C}$ буде похідною CNF з n двійковими змінними, і нехай $Sat(\mathbb{C})$ є підмножиною $\{0, 1\}^n$, включаючи всі розв'язки $\mathbb{C}$. $\mathbb{C}$ можна представити через набір булевих поліномів $\mathbb{F}$ таких, що $Sat(\mathbb{C}) = \mathbb{Z}(\mathbb{F})$, де $\mathbb{Z}(\mathbb{F})$ позначає множину розв'язків $\mathbb{F}$. Для цього кожен вираз Горна $(\neg X_{i_0} \vee \dots \vee \neg X_{i_{m-2}} \vee X_{i_{m-1}})$ перекладається на біном $x_{i_0} \cdots x_{i_{m-2}} \cdot (x_{i_{m-1}} + 1)$ у кільці булевих поліномів.

Висновки до розділу 2

У розділі розглянуто структуру та процедуру ініціалізації потокового шифру Струмок. Подібність будови Струмка до будови потокового шифру SNOW 2.0. Описано атаку часткового вгадування та кроки її побудови. Розглянуто основні підходи до пошуку базисів вгадування, описано програмний застосунок для пошуку базисів вгадування для побудови за ними атак часткового вгадування. Обґрунтовано застосування програмної реалізації Autoguess для пошуку атак на потоковий шифр Струмок. Розглянуто метод побудови атаки часткового вгадування у випадку, коли глибина графу невідома. Описано метод зведення задачі часткового вгадування до задачі пошуку базиса Грьобнера.

3 ПОШУК БАЗИСІВ АТАК ЧАСТКОВОГО ВГАДУВАННЯ НА ПОТОКОВИЙ ШИФР СТРУМОК

У розділі розглянуто особливості роботи із програмним застосунком Autoguess. Описано процес пошуку атак часткового вгадування на потоковий шифр Струмок, оцінено складність знайдених атак.

Створення файла функціональних зв'язків Autoguess приймає на вхід файл, у якому описано функціональні зв'язки між регістрами та комірками регістрів шифру. Для отримання вхідного файлу треба знати зв'язки між змінними стану у момент часу t . Розглянемо співвідношення для потокового шифру Струмок рис.2.1:

- $S_{t+16}, S_{t+13}, S_{t+11}, S_t$ приймають участь у оновленні LFSR, а саме генерації значення комірки S_{t+16} ,
- $S_{t+15}, R1_t, R2_t, S_t$ приймають участь у породженні гами,
- $R1_{t+1}, R2_t, S_{t+13}$ приймають участь у оновленні значення регістра $R2$,
- $R2_{t+1}, R1_t$ приймають участь у оновленні значення регістра $R1$.

Із останнього співвідношення можна виразити $R1_t$ та дещо спростити систему [6]. В результаті отримано систему зв'язків:

$$\begin{aligned} &S_{t+16}, S_{t+13}, S_{t+11}, S_t, \\ &S_{t+15}, R2_{t+1}, R2_t, S_t, \\ &R2_{t+2}, R2_t, S_{t+13}. \end{aligned}$$

Створивши Python скрипт, у якому в циклі перебираються значення індекса t , можна отримати текстовий файл з усіма необхідними зв'язками між змінними внутрішнього стану Струмка. Із методами переходу від системи рівнянь до системи зв'язків можна детальніше ознайомитись у роботі [6].

3.1 Результати пошуку базисів вгадування

У результаті пошуку базиса атаки на основі згенерованого файла за допомогою моделі обчислень SAT знайдено базис довжиною 7 слів. Запуск пошуку із використанням різних обчислюючих моделей та з різними параметрами дозволяє отримувати різні базиси, але жодного разу не вдалося знайти базис, потужність якого була б меншою, ніж 7 слів. Також експериментальним шляхом встановлено, що мінімальна глибина графа поширення знань становить 9 кроків. Якщо використовувати меншу кількість кроків, то модель не буде мати рішень. [6].

Програмний застосунок Autoguess є загальним і універсальним інструментом, тож його було дещо змінено і адаптовано для пошуку і аналізу наявності атак часткового вгадування на потоковий шифр Струмок. Зокрема пришвидшено і оптимізовано пошук базисів вгадування із використанням алгоритма пошуку базисів Грьобнера. Алгоритм змінено таким чином, щоб він повертав усі мінімальні базиси. Тому наступні пошуки продовжено із застосуванням алгоритма пошуку базисів Грьобнера.

У результаті пошуку атак часткового вгадування із використанням адаптованого алгоритма пошуку базисів Грьобнера визначено існування базисів вгадування, довжина яких складає 7 слів довжиною 64 біта. Менших базисів вгадування не вдалося знайти. Таким чином атака часткового вгадування на потоковий шифр Струмок із перебором усіх можливих значень базиса має обчислювальну складність $O(2^{7 \times 64}) = O(2^{448})$ операцій, що є меншою, ніж складність перебору усіх значень ключа для варіанта Струмка із ключем довжиною 512 бітів.

Експериментальним шляхом встановлено, що 11 тактів шифру Струмок є мінімальною кількістю тактів для знаходження базиса вгадування і повного відновлення внутрішнього стану. При використанні 11 тактів Струмка пошук із використанням алгоритма пошуку базисів

Грьобнера знаходить 176 базисів вгадування, які складаються із 7 слів, хоча деякі із них можуть бути ізоморфними. Зміст файла зі зв'язками та перелік знайдених базисів приведено у додатку.

Підтверджено існування атаки часткового вгадування зі складністю $O(2^{448})$, описаної у роботі [7], і знайдено раніше невідомі базиси вгадування. Різні базиси реалізують різні графи розповсюдження знань. Найбільш перспективними для подальшого вивчення стійкості шифру Струмок до атак часткового вгадування можуть бути базиси, графи яких мають найменшу глибину.

Приклади побудови атак

Розглянемо конкретні приклади побудови атак часткового вгадування на потоковий шифр Струмок. У цьому прикладі розглянуто варіант атаки на шифр із використанням 13 тактів. Файл із функціональними зв'язками, як вже зазначалося вище, створюється за шаблоном:

$$\begin{aligned} S_{t+16}, S_{t+13}, S_{t+11}, S_t, \\ S_{t+15}, R2_{t+1}, R2_t, S_t, \\ R2_{t+2}, R2_t, S_{t+13}. \end{aligned}$$

Для цього треба перебрати у циклі значення індекса t від 0 до 13. Отриману систему вивести у текстовий файл. У другому рядку на першому місці пропущено слово Z_t тому, що це слово гами, яке ми знаємо. Autoguess вимагає вказувати відомі слова після ключового

слова *known*. Тому система зв'язків матиме наступний вигляд:

$$S_{16}, S_{13}, S_{11}, S_0$$

$$Z_0, S_{15}, R_1, R_0, S_0$$

$$R_2, S_{13}, R_0$$

$$S_{17}, S_{14}, S_{12}, S_1$$

$$Z_1, S_{16}, R_2, R_1, S_1$$

$$R_3, S_{14}, R_1$$

$$S_{18}, S_{15}, S_{13}, S_2$$

$$Z_2, S_{17}, R_3, R_2, S_2$$

$$R_4, S_{15}, R_2$$

$$S_{19}, S_{16}, S_{14}, S_3$$

$$Z_3, S_{18}, R_4, R_3, S_3$$

$$R_5, S_{16}, R_3$$

$$S_{20}, S_{17}, S_{15}, S_4$$

$$Z_4, S_{19}, R_5, R_4, S_4$$

$$R_6, S_{17}, R_4$$

$$S_{21}, S_{18}, S_{16}, S_5$$

$$Z_5, S_{20}, R_6, R_5, S_5$$

$$R_7, S_{18}, R_5$$

$$S_{22}, S_{19}, S_{17}, S_6$$

$$Z_6, S_{21}, R_7, R_6, S_6$$

$$R_8, S_{19}, R_6$$

$$S_{23}, S_{20}, S_{18}, S_7$$

$$Z_7, S_{22}, R_8, R_7, S_7$$

$$R_9, S_{20}, R_7$$

$$S_{24}, S_{21}, S_{19}, S_8$$

$$Z_8, S_{23}, R_9, R_8, S_8$$

$$\begin{aligned}
&R_{10}, S_{21}, R_8 \\
&S_{25}, S_{22}, S_{20}, S_9 \\
&Z_9, S_{24}, R_{10}, R_9, S_9 \\
&R_{11}, S_{22}, R_9 \\
&S_{26}, S_{23}, S_{21}, S_{10} \\
&Z_{10}, S_{25}, R_{11}, R_{10}, S_{10} \\
&R_{12}, S_{23}, R_{10} \\
&S_{27}, S_{24}, S_{22}, S_{11} \\
&Z_{11}, S_{26}, R_{12}, R_{11}, S_{11} \\
&R_{13}, S_{24}, R_{11} \\
&S_{28}, S_{25}, S_{23}, S_{12} \\
&Z_{12}, S_{27}, R_{13}, R_{12}, S_{12} \\
&R_{14}, S_{25}, R_{12}
\end{aligned}$$

Повний файл зв'язків наведено у додатку. Оберемо один із знайдених базисів вгадування:

$$S_6, R_7, R_6, S_{19}, S_5, R_5, R_4.$$

За цим базисом можна відновити стан шифру Струмок у певний момент часу за 9 кроків.

Для атаки на шифр криптоаналітику треба мати результати роботи шифру якнайменше для 13 тактів. Послідовність обчислень значень слів стану шифру позначено на графі розповсюдження знань. Криптоаналітик має перебирати усі можливі заповнення базиса від 0 до $2^{448} - 1$ включно. Для кожного зафіксованого значення базиса потрібно відновити внутрішній стан Струмка користуючись графом розповсюдження знань. Після відновлення внутрішнього стану потрібно згенерувати гаму. Якщо згенероване слово не співпадає зі словом відомої криптоаналітику гами,

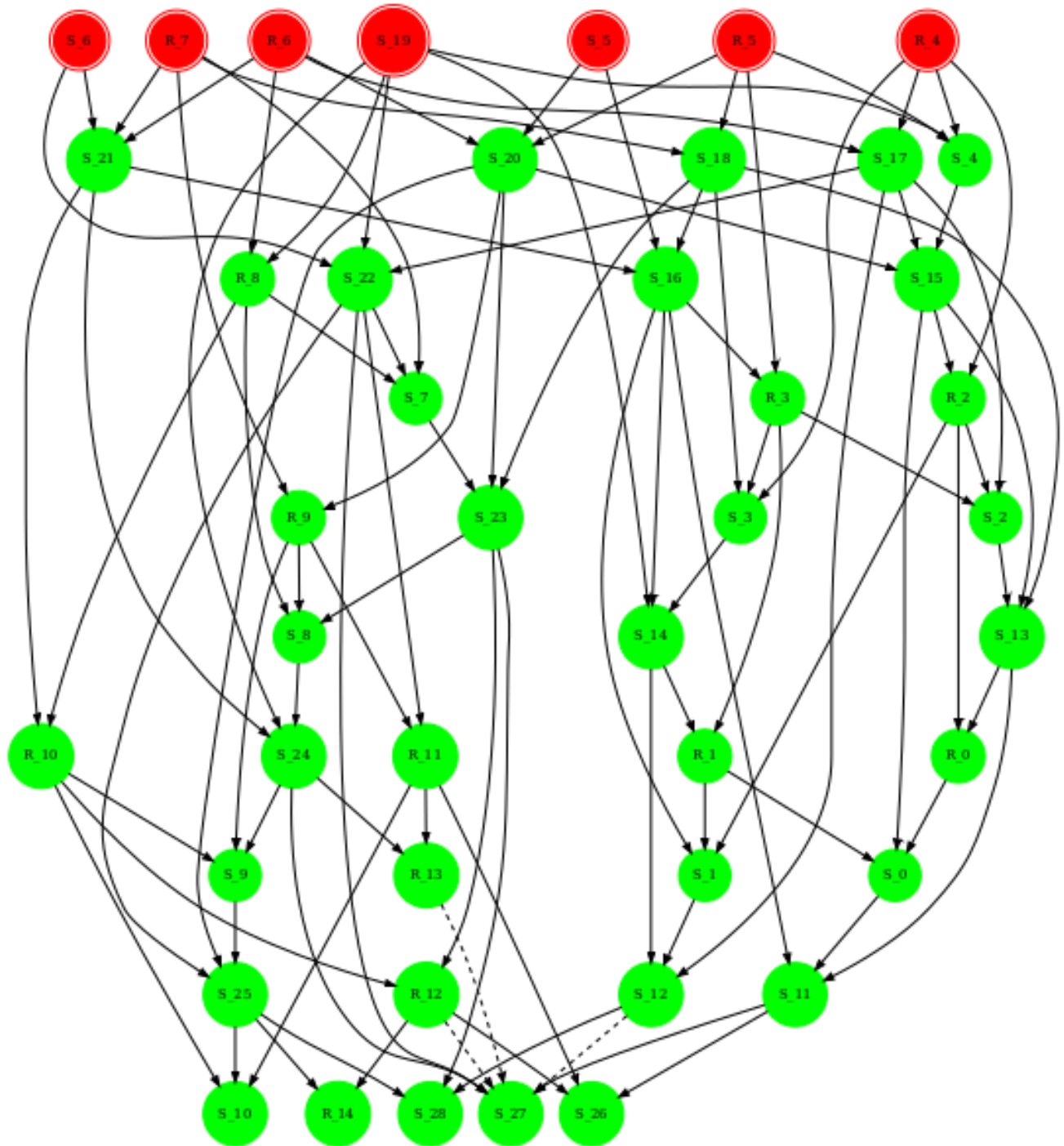


Рисунок 3.1 – Граф розповсюдження знань

то заповнення базиса вгадано неправильно. Тоді криптоаналітик обирає наступне заповнення базиса доти, доки не знайде таке заповнення, при якому згенерована гама співпадає із спостережуваною.

Алгоритм відновлення стану шифру Струмок виглядатиме так (через двокрапку позначено шукану змінну та множину змінних, завдяки яким

можна обчислити її значення):

- На першому кроці будуть відновлені значення:

$$(S_{21} : S_6, R_7, R_6),$$

$$(S_{20} : R_6, S_5, R_5),$$

$$(S_{18} : R_7, R_5),$$

$$(S_{17} : R_6, R_4),$$

$$(S_4 : S_{19}, R_5, R_4).$$

- На другому кроці будуть відновлені значення:

$$(R_8 : R_6, S_{19}),$$

$$(S_{22} : S_6, S_{19}, S_{17}),$$

$$(S_{16} : S_{21}, S_5, S_{18}),$$

$$(S_{15} : S_{20}, S_{17}, S_4).$$

- На третьому:

$$(S_7 : R_8, S_{22}, R_7),$$

$$(R_3 : S_{16}, R_5),$$

$$(R_2 : S_{15}, R_4).$$

- На четвертому:

$$(R_9 : R_7, S_{20}),$$

$$(S_{23} : S_7, S_{20}, S_{18}),$$

$$(S_3 : S_{18}, R_3, R_4),$$

$$(S_2 : R_3, R_2, S_{17}).$$

– На п'ятому:

$$\begin{aligned}(S_8 : R_8, R_9, S_{23}), \\ (S_{14} : S_{19}, S_{16}, S_3), \\ (S_{13} : S_2, S_{15}, S_{18}).\end{aligned}$$

– На шостому:

$$\begin{aligned}(R_{10} : S_{21}, R_8), \\ (S_{24} : S_{21}, S_{19}, S_8), \\ (R_{11} : R_9, S_{22}), \\ (R_1 : S_{14}, R_3), \\ (R_0 : R_2, S_{13}).\end{aligned}$$

– На сьомому:

$$\begin{aligned}(S_9 : R_{10}, R_9, S_{24}), \\ (R_{13} : S_{24}, R_{11}), \\ (S_1 : S_{16}, R_1, R_2), \\ (S_0 : R_1, S_{15}, R_0).\end{aligned}$$

– На восьмому:

$$\begin{aligned}(S_{25} : S_{22}, S_{20}, S_9), \\ (R_{12} : R_{10}, S_{23}), \\ (S_{12} : S_{14}, S_1, S_{17}), \\ (S_{11} : S_{16}, S_0, S_{13}).\end{aligned}$$

– І на останньому дев'ятому кроці будуть відновлені значення:

$$(S_{10} : R_{10}, S_{25}, R_{11}),$$

$$(R_{14} : S_{25}, R_{12}),$$

$$(S_{28} : S_{25}, S_{23}, S_{12}),$$

$$(S_{27} : S_{24}, S_{22}, R_{12}, R_{13}, S_{12}, S_{11}),$$

$$(S_{26} : R_{12}, R_{11}, S_{11}).$$

Причому варто зазначити, що значення комірки S_{27} може бути відновлене кількома різними способами.

Відновлення стану потокового шифру Струмок не є складною задачею, тому її складністю можна нехтувати і оцінити складність атаки часткового вгадування із використанням базиса із семи слів як $O(2^{448})$.

У результаті пошуку базисів вгадування для різної кількості тактів потокового шифру Струмок отримано певний розподіл кількості базисів вгадування.

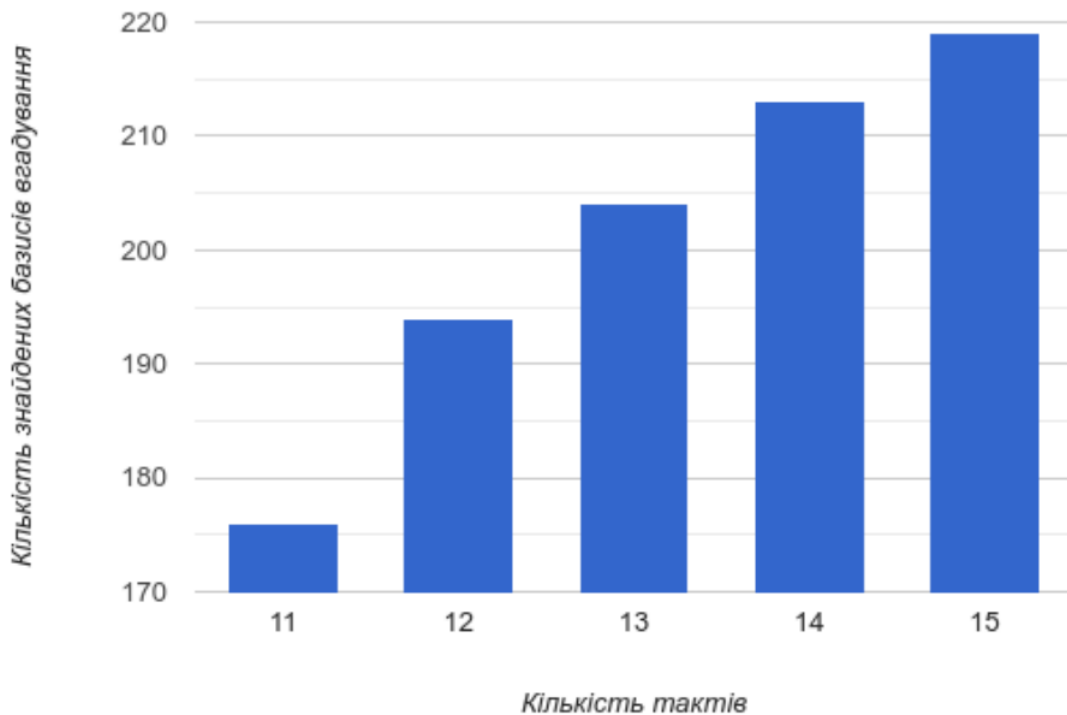


Рисунок 3.2 – Розподіл кількості базисів вгадування

3.2 Оптимальний алгоритм перебору

Складність атаки часткового вгадування на потоковий шифр Струмок можна зменшити шляхом використання оптимального алгоритма перебору значень базиса.

Оптимальний алгоритм перебору наведено у роботі [3]. Автори запропонували застосування рекурсивного пошуку для побудови атаки на потоковий шифр SNOW-V.

Використання рекурсивного пошуку дозволяє зменшити обчислювальну складність алгоритма за рахунок збільшення просторової складності, адже такий алгоритм намагається вгадувати значення слів базиса у оптимальній послідовності, максимально скорочуючи при цьому неправильні шляхи обчислень.

Покращений алгоритм одразу відкидає гілку обчислень, якщо значення слів базиса не відповідають певним критеріям, це дозволяє одразу повернутись на велику кількість кроків назад. Але такий підхід вимагає зберігання в пам'яті великої кількості кроків обчислень, на які можна відразу повернутися. Вгадуючи змінні в певному порядку можна або вгадати меншу кількість змінних, або скоротити деякі шляхи вгадування у яких уже відомі (або вгадані, або визначені) змінні не задовольняють деяким обмеженням рівняння в середині.

Наприклад, розглянемо найпростіший цикл у псевдокоді, де x, y, z є трьома словами довжини 8 біт. Легко встановити, що складність такого циклу становить $T = 2^{24}$.

```

T = 0 ;
for ( x =0; x <256; x++)
    for ( y =0; y <256; y++)
        for ( z =0; z <256; z++){
            T = T + 1 ;
            . . .
        }

```

Однак для іншого циклу, де $L1[x]$ – це списки, що залежать від конкретних значень x , а $L2[x, y]$ – це списки, що залежать від значень x, y , розмір циклу не є фіксований. Він залежить від довжини списків $L1[x]$ і $L2[x, y]$. Наприклад, для певного значення x , після того як ми пройшли через кожне значення $L1[x]$ для y (і, відповідно, наступного z), ми можемо негайно перейти до іншого значення x , замість того, щоб розглядати всі 256 значень y . У цьому випадку складність дорівнює не просто 2^{24} , а кількості дійсних шляхів циклу.

```

T = 0 ;
for ( x =0; x <256; x++)
    for ( y = L1[x].first ; y!=NULL ; y=y->next )
        for ( z = L2[x, y].first ; z !=NULL ; z=z->next ){
            T = T + 1 ;
            . . .
        }

```

Висновки до розділу 3

У розділі описано застосування програми Autoguess для пошуку базиса атаки часткового вгадування на потоковий шифр Струмок. Описано метод створення системи зв'язків між змінними. У результаті проведеного пошуку встановлено, що для потокового шифру Струмок існують базиси вгадування, які складаються із семи слів довжиною 64

біта.

Встановлено емпіричним шляхом, що 11 тактів – це мінімальна кількість тактів потокового шифру Струмок, необхідна для знаходження базиса вгадування, із якого можна відновити внутрішній стан Струмка. Складність атаки часткового вгадування із використанням знайдених базисів вгадування складає $O(2^{448})$ операцій, що є меншою, ніж складність перебору усіх значень ключа для варіанта Струмка із ключем довжиною 512 бітів.

Наведено детальний покроковий приклад побудови атаки часткового вгадування на потоковий шифр Струмок із використанням графа розповсюдження знань.

Розглянуто методи зменшення складності атаки, а саме алгоритм динамічного перебору заповнень базиса вгадування.

ВИСНОВКИ

У ході роботи проведений аналіз опублікованих джерел за тематикою атак часткового вгадування на потокові шифри сімейства SNOW. Досліджено будову поточкових шифрів сімейства SNOW та проведено порівняльний аналіз будови поточкових шифрів Струмок і SNOW-2.0.

Досліджено методи побудови атак часткового вгадування та методи застосування цих атак до поточкових шифрів сімейства SNOW. Розглянуто програмну реалізацію методів аналізу атак часткового вгадування на шифри.

Використано програмний застосунок Autoguess для пошуку базисів атак часткового вгадування на потоковий шифр Струмок. Встановлено, що 11 тактів – це мінімальна кількість тактів потокового шифру Струмок, необхідна для знаходження базисів вгадування, із яких можна побудувати атаки часткового вгадування на потоковий шифр Струмок.

Складність атаки часткового вгадування із використанням знайдених базисів вгадування складає $O(2^{448})$ операцій, що є меншою за складність перебору усіх ключів для варіанта шифру із ключем довжиною 512 бітів. Всього знайдено 176 базисів для 11 тактів шифру. Наведено детальний покроковий приклад побудови атаки часткового вгадування на потоковий шифр Струмок із використанням графу розповсюдження знань. Запропоновано метод зменшення складності атаки, а саме алгоритм динамічного перебору заповнень базиса вгадування. Перспективним методом зниження складності атаки також може бути метод розбиття слів на менші складові частини.

Підтверджено та покращено результати попередніх досліджень наявності атак часткового вгадування на потоковий шифр Струмок.

Рекомендовано використовувати програмні застосунки загального виду в процесі розробки шифру для пошуку наявних вразливостей.

ПЕРЕЛІК ПОСИЛАНЬ

1. Математична структура потокового шифру Струмок / О. Кузнецов, І. Горбенко, Ю. Горбенко, А. Олексійчук, В. Тимченко // Radiotekhnika. — 2018. — No 193. — С. 17—27.
2. Hawkes P., Rose G. G. Guess-and-Determine Attacks on SNOW // Selected Areas in Cryptography / за ред. К. Nyberg, Н. Heys. — Berlin, Heidelberg : Springer Berlin Heidelberg, 2003. — С. 37—46. — ISBN 978-3-540-36492-4.
3. Yang J., Johansson T., Maximov A. Improved guess-and-determine and distinguishing attacks on SNOW-V. — 2021. — URL: <https://eprint.iacr.org/2021/544> ; <https://eprint.iacr.org/2021/544>. Cryptology ePrint Archive, Paper 2021/544.
4. Nia M. S. N., Payandeh A. THE NEW HEURISTIC GUESS AND DETERMINE ATTACK ON SNOW 2.0 STREAM CIPHER. — 2014. — URL: <https://eprint.iacr.org/2014/619> ; <https://eprint.iacr.org/2014/619>. Cryptology ePrint Archive, Paper 2014/619.
5. Ahmadi, H., Eghlidos, T.: Heuristic guess-and-determine attacks on stream ciphers. IET Information Security 3(2), 66–73 (2009)
6. Hadipour H., Eichlseder M. Autoguess: A Tool for Finding Guess-and-Determine Attacks and Key Bridges // Applied Cryptography and Network Security – ACNS 2022. T. 13269 / за ред. G. Ateniese, D. Venturi. — Springer, 2022. — С. 230—250. — (LNCS). — DOI: 10.1007/978-3-031-09234-3-12. — eprint: 2021/1529.
7. A. Fesenko. Cryptanalysis of Strumok Cipher Initialization // Proceedings of Central European Conference on Cryptology, CECC'2022 , Smolenice, Slovakia, June 26-29. — 2022. — pp. 41–45.
8. Patrik Ekdahl, Thomas Johansson, Alexander Maximov and Jing Yang «A new SNOW stream cipher called SNOW-V»
9. Carlos Cid, Matthew Dodd, and Sean Murphy. 2020. A Security

Evaluation of the SNOW-V Stream Cipher. (4 June 2020). Quaternion Security Ltd. <https://www.3gpp.org/ftp/tsg-sa/WG3-Security/TSGS3-101/Docs/S3-202852.zip>.

10. Lin Jiao, Yongqiang Li, and Yonglin Hao. 2020. A Guess-And-Determine Attack On SNOW-V Stream Cipher. *Comput. J.* 63, 12 (03 2020), 1789–1812. <https://doi.org/10.1093/comjnl/bxaa003>

11. Ekdahl P., Johansson T. «SNOW - A new stream cipher», 2000, First open Nessie Workshop.

12. Ekdahl P., Johansson T. «A New Version of the Stream Cipher SNOW», *Selected Areas in Cryptography*, Springer Berlin Heidelberg, 2003. - c. 47-61. - DOI: 10.1007/3 – 540 – 36492 – 7₅.

13. D. Coppersmith, S. Halevi, C. Jutla, «Cryptanalysis of stream ciphers with linear masking», To appear in *Advances in Cryptology - CRYPTO 2002*, *Lecture Notes in Computer Science*, Springer, 2002

14. J. Daemen, V. Rijmen, «The design of Rijndael», Springer Verlag Series on Information Security and Cryptography, Springer Verlag, 2002, ISBN 3-540-42580-2.

15. ETSI/SAGE. Specification of the 3GPP confidentiality and integrity algorithms UEA2& UIA2, document 2: SNOW 3G specification, version 1.1, 2006.

16. Jing Yang and Thomas Johansson and Alexander Maximov. Vectorized linear approximations for attacks on SNOW 3G. *Cryptology ePrint Archive*, Report 2019/991.

17. Kuznetsov O., Lutsenko M., Ivanenko D. «Strumok stream cipher: Specification and basic properties», 2016 Third International Scientific-Practical Conference Problems of Infocommunications Science and Technology (PIC S&T). - IEEE, 10.2016. - DOI: 10.1109/infocommst.2016.7905335.

18. Gorbenko I., Kuznetsov O., Gorbenko Y., Alekseychuk A., Tymchenko V. «Strumok keystream generator», 2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT). - IEEE, 05.2018. 24-27 May, 2018. Kyiv, Ukraine. – P. 292-299. - DOI:

10.1109/dessert.2018.8409147.

19. Danner, J., Kreuzer, M.: A fault attack on KCipher-2. *International Journal of Computer Mathematics: Computer Systems Theory* pp. 1–22 (2020)

20. Een, N.: MiniSat: A SAT solver with conflict-clause minimization. In: *Theory and Applications of Satisfiability Testing – SAT 2005*. pp. 502–518 (2005), available from <http://minisat.se>

21. Bouillaguet, C., Derbez, P., Fouque, P.A.: Automatic search of attacks on round-reduced AES and applications. In: *Advances in Cryptology – CRYPTO 2011*. pp. 169–187. Springer (2011)

22. Gorbenko I., Kuznetsov A., Gorbenko Y., Alekseychuk A., Tymchenko V. «Потоковий шифр «Струмок», Комп'ютерні науки та кібербезпека. – 2018. – No 1. – vol. 9. – С. 4-14.

23. Олексійчук А. М. «Достатня умова стійкості SNOW 2.0-подібних поточкових шифрів відносно певних атак зі зв'язаними ключами», – *Захист інформації*. – 2016. – Т. 18. – No 3. – С. 261–268.50

ДОДАТОК А ТЕКСТИ ПРОГРАМ

A.1 Python скрипт генерації файлів зв'язків

```

import os
from argparse import ArgumentParser

output_dir = os.path.curdir

def strumok(T=13, wordlen=64):
    cipher_name = 'strumok'
    recommended_mg = 7
    recommended_ms = 9
    eqs = f'#{cipher_name} {T} Rounds\n'
    eqs += 'connection relations\n'
    if wordlen == 64:
        for t in range(T):
            eqs += f'S_{t + 16}, S_{t + 13}, S_{t + 11},\n\
S_{t}\n'
            eqs += f'Z_{t}, S_{t + 15}, R_{t + 1}, R_{t},\n\
S_{t}\n'
            eqs += f'R_{t + 2}, S_{t + 13}, R_{t}\n'
        eqs += 'known\n' + '\n'.join([f'Z_{t}' for t in range(T)])
        eqs += 'end'
        eqsfile_path = os.path.join(output_dir, f'{cipher_name}_{T}\n\
_ms{recommended_ms}_wl{wordlen}.txt')
        with open(eqsfile_path, 'w') as relation_file:
            relation_file.write(eqs)
    elif wordlen == 32:
        for t in range(T):
            # LFSR:
            eqs += f'S_{t+16}_0, S_{t+13}_0, S_{t+11}_0,\n\

```

```

S{t}_0\n'

eqs += f'S_{t+16}_1, S_{t+13}_1, S_{t+11}_0,\n\
S_{t+11}_1, S_{t}_0, S_{t}_1\n'

# Z:
eqs += f'Z_{t}_0, S_{t+15}_0, R1_{t}_0, R2_{t}_0,\n\
S_{t}_0\n'
eqs += f'Z_{t}_1, S_{t+15}_1, S_{t+15}_0, R1_{t}_1,\n\
R1_{t}_0, R2_{t}_1, S_{t}_1\n'

# FSM R1:
eqs += f'R1_{t+1}_0, R2_{t}_0, S_{t+13}_0\n'
eqs += f'R1_{t+1}_1, R2_{t}_1, R2_{t}_0, S_{t+13}_1,\n\
S_{t+13}_0\n'

# FSM R2:
eqs += f'R2_{t+1}_0, R1_{t}_0, R1_{t}_1\n'
eqs += f'R2_{t+1}_1, R1_{t}_0, R1_{t}_1\n'
eqs += 'known\n' + '\n'.join([f'Z_{t}_{i}' for i in\
range(2) for t in range(T)]) + '\n'
eqs += 'end'

eqsfile_path = os.path.join(output_dir, f'{cipher_name}\n\
_{T}clk_mg{recommended_mg}_ms{recommended_ms}\n\
_wl{wordlen}.txt')

with open(eqsfile_path, 'w') as relation_file:
    relation_file.write(eqs)

elif wordlen == 8:
    for t in range(T):
        for i in range(wordlen):
            # LFSR:

```

```

eqs += f'S_{t+16}_{i}', '\\
+ ', '.join([f'S_{t}_{k}', '\\
S_{t + 11}_{k}' for k in range(i + 1)])\\
+ f', S_{t + 13}_{i}\\n'

# Z:
eqs += f'Z_{t}_{i}, S_{t}_{i}', '\\
+ ', '.join([f'S_{t + 15}_{k}', '\\
R1_{t}_{k}' for k in range(i + 1)]) + f',\\
R2_{t}_{i}\\n'

# FSM R1:
eqs += f'R1_{t + 1}_{i}', '\\
+ ', '.join([f'S_{t + 13}_{k}', '\\
R2_{t}_{k}' for k in range(i + 1)]) + '\\n'

# FSM R2:
eqs += f'R2_{t + 1}_{i}', '\\
+ ', '.join([f'R1_{t}_{k}' for k in range(8)])\\
+ '\\n'

eqs += 'known\\n' + '\\n'.join([f'Z_{t}_{k}' for k\\
in range(8) for t in range(T)]) + '\\n'
eqs += 'end'

eqsfile_path = os.path.join(output_dir, f'{cipher_name}\\
_{T}clk_mg{recommended_mg}_ms{recommended_ms}\\
_wl{wordlen}.txt')

with open(eqsfile_path, 'w') as relation_file:
    relation_file.write(eqs)

def main():
    parser = ArgumentParser()

```

```

parser.add_argument(
    '-t',
    '--ticks',
    nargs=1,
    help='Number of cipher ticks'
)
parser.add_argument(
    '-wl',
    '--wordlen',
    nargs=1,
    help='Bit length of word'
)
args = parser.parse_args()
T = int(args.ticks[0])
wordlen = int(args.wordlen[0])
strumok(T=T, wordlen=wordlen)

if __name__ == '__main__':
    main()

```

A.2 Файл зв'язків для 11 тактів шифру Струмок

```

#strumok 11 Rounds
connection relations
S_16, S_13, S_11, S_0
Z_0, S_15, R_1, R_0, S_0
R_2, S_13, R_0
S_17, S_14, S_12, S_1
Z_1, S_16, R_2, R_1, S_1
R_3, S_14, R_1
S_18, S_15, S_13, S_2

```

Z_2, S_17, R_3, R_2, S_2
 R_4, S_15, R_2
 S_19, S_16, S_14, S_3
 Z_3, S_18, R_4, R_3, S_3
 R_5, S_16, R_3
 S_20, S_17, S_15, S_4
 Z_4, S_19, R_5, R_4, S_4
 R_6, S_17, R_4
 S_21, S_18, S_16, S_5
 Z_5, S_20, R_6, R_5, S_5
 R_7, S_18, R_5
 S_22, S_19, S_17, S_6
 Z_6, S_21, R_7, R_6, S_6
 R_8, S_19, R_6
 S_23, S_20, S_18, S_7
 Z_7, S_22, R_8, R_7, S_7
 R_9, S_20, R_7
 S_24, S_21, S_19, S_8
 Z_8, S_23, R_9, R_8, S_8
 R_10, S_21, R_8
 S_25, S_22, S_20, S_9
 Z_9, S_24, R_10, R_9, S_9
 R_11, S_22, R_9
 S_26, S_23, S_21, S_10
 Z_10, S_25, R_11, R_10, S_10
 R_12, S_23, R_10
 known
 Z_0
 Z_1
 Z_2
 Z_3

Z_4
 Z_5
 Z_6
 Z_7
 Z_8
 Z_9
 Z_10
 end

A.3 Файл зв'язків для 13 тактів шифру Струмок

```

#strumok 13 Rounds
connection relations
S_16, S_13, S_11, S_0
Z_0, S_15, R_1, R_0, S_0
R_2, S_13, R_0
S_17, S_14, S_12, S_1
Z_1, S_16, R_2, R_1, S_1
R_3, S_14, R_1
S_18, S_15, S_13, S_2
Z_2, S_17, R_3, R_2, S_2
R_4, S_15, R_2
S_19, S_16, S_14, S_3
Z_3, S_18, R_4, R_3, S_3
R_5, S_16, R_3
S_20, S_17, S_15, S_4
Z_4, S_19, R_5, R_4, S_4
R_6, S_17, R_4
S_21, S_18, S_16, S_5
Z_5, S_20, R_6, R_5, S_5
R_7, S_18, R_5

```

S_22, S_19, S_17, S_6
 Z_6, S_21, R_7, R_6, S_6
 R_8, S_19, R_6
 S_23, S_20, S_18, S_7
 Z_7, S_22, R_8, R_7, S_7
 R_9, S_20, R_7
 S_24, S_21, S_19, S_8
 Z_8, S_23, R_9, R_8, S_8
 R_10, S_21, R_8
 S_25, S_22, S_20, S_9
 Z_9, S_24, R_10, R_9, S_9
 R_11, S_22, R_9
 S_26, S_23, S_21, S_10
 Z_10, S_25, R_11, R_10, S_10
 R_12, S_23, R_10
 S_27, S_24, S_22, S_11
 Z_11, S_26, R_12, R_11, S_11
 R_13, S_24, R_11
 S_28, S_25, S_23, S_12
 Z_12, S_27, R_13, R_12, S_12
 R_14, S_25, R_12
 known
 Z_0
 Z_1
 Z_2
 Z_3
 Z_4
 Z_5
 Z_6
 Z_7
 Z_8

Z_9

Z_10

Z_11

Z_12

end

A.4 Знайдені базиси вгадування для 11 тактів шифра

Струмук

$R_5 * S_3 * S_{19} * R_3 * R_0 * S_{15} * S_{11}$

$R_5 * S_3 * S_{19} * R_3 * S_1 * S_0 * S_{11}$

$R_5 * S_3 * S_{19} * R_3 * S_1 * R_0 * S_{11}$

$S_4 * R_5 * S_{19} * R_2 * R_0 * R_1 * S_{11}$

$S_4 * S_{20} * R_5 * S_2 * R_3 * S_1 * S_{14}$

$R_6 * R_4 * R_3 * S_1 * S_{12} * R_0 * S_0$

$R_6 * S_3 * R_4 * S_2 * S_{18} * S_{14} * S_{11}$

$R_6 * S_3 * R_4 * S_2 * S_{18} * S_1 * S_{12}$

$R_6 * R_5 * R_4 * S_2 * R_3 * S_0 * S_{11}$

$R_6 * R_5 * R_4 * S_2 * R_3 * S_1 * S_0$

$R_6 * S_4 * S_{20} * R_4 * S_2 * S_1 * S_{12}$

$S_5 * R_6 * S_{20} * R_3 * S_1 * S_{14} * S_{15}$

$S_5 * R_6 * S_{20} * S_2 * R_3 * S_1 * S_{14}$

$S_5 * R_6 * S_{20} * R_4 * R_3 * S_1 * S_{12}$

$R_7 * R_5 * S_2 * R_3 * S_1 * S_{14} * R_0$

$R_7 * R_5 * R_4 * S_2 * R_2 * R_1 * S_{11}$

$R_7 * R_5 * S_3 * R_4 * R_0 * S_0 * S_{11}$

$R_7 * R_5 * S_3 * R_4 * R_2 * S_0 * S_{11}$

$R_7 * R_5 * S_3 * R_4 * S_2 * S_0 * S_{11}$

$R_7 * S_4 * R_5 * S_3 * S_{19} * R_0 * S_0$

$R_7 * S_4 * R_5 * S_3 * S_{19} * S_1 * S_{13}$

$R_7*S_4*R_5*S_3*S_{19}*S_1*S_0$
 $R_7*S_4*R_5*S_3*S_{19}*S_1*R_0$
 $R_7*S_4*R_5*S_3*S_{19}*S_1*S_{12}$
 $R_7*S_4*R_5*S_3*S_{19}*S_2*S_1$
 $R_7*R_6*R_5*S_3*R_4*S_1*S_{12}$
 $R_7*R_6*R_5*S_3*R_4*S_2*S_{11}$
 $R_7*R_6*R_5*S_3*R_4*S_2*S_0$
 $R_7*R_6*R_5*S_3*R_4*S_2*S_1$
 $R_7*S_5*S_{21}*R_5*S_2*S_{17}*S_{15}$
 $R_7*S_5*S_{21}*R_5*S_2*S_{17}*R_0$
 $R_7*S_5*S_{21}*R_5*S_3*R_2*R_0$
 $S_6*R_7*S_{21}*R_4*S_2*R_3*S_{13}$
 $S_6*R_7*S_{21}*R_4*S_2*R_3*R_0$
 $R_8*R_6*S_3*R_4*S_1*S_{12}*R_2$
 $R_8*R_6*S_3*R_4*S_{18}*S_1*S_{12}$
 $R_8*R_6*R_5*S_3*R_3*S_1*S_{15}$
 $R_8*R_6*R_5*S_3*R_3*S_1*S_{12}$
 $R_8*R_6*R_5*S_3*S_2*R_3*S_1$
 $R_8*R_6*S_4*R_5*S_1*S_{12}*R_1$
 $R_8*R_6*S_4*R_5*R_3*S_1*S_{12}$
 $R_8*R_6*S_4*R_5*S_3*S_1*S_{12}$
 $R_8*S_5*R_6*S_4*S_{20}*S_1*R_1$
 $R_8*S_5*R_6*S_4*S_{20}*S_2*R_1$
 $R_8*S_5*R_6*S_4*S_{20}*S_2*R_0$
 $R_8*S_5*R_6*S_4*S_{20}*S_2*S_{14}$
 $R_8*S_5*R_6*S_4*S_{20}*S_2*S_1$
 $R_8*S_5*R_6*S_4*S_{20}*S_3*S_2$
 $R_8*R_7*R_6*S_4*R_5*S_3*S_{12}$
 $R_8*R_7*R_6*S_4*R_5*S_3*S_1$
 $R_8*R_7*R_6*S_4*R_5*S_3*S_2$
 $R_8*S_6*S_{22}*R_6*S_3*S_{18}*R_1$

$R_8*S_6*S_{22}*R_6*S_3*S_{18}*S_{14}$
 $R_8*S_6*S_{22}*R_6*S_4*R_3*S_{14}$
 $R_8*S_6*S_{22}*R_6*S_4*S_3*S_{14}$
 $S_7*R_8*S_{22}*R_5*S_3*R_4*R_1$
 $S_7*R_8*S_{22}*R_5*S_3*R_4*S_{14}$
 $R_9*R_7*S_4*R_5*S_2*S_{17}*R_0$
 $R_9*R_7*S_4*R_5*S_3*S_{19}*S_1$
 $R_9*R_7*R_6*S_4*R_4*S_2*S_{16}$
 $R_9*R_7*R_6*S_4*R_4*S_2*R_0$
 $R_9*R_7*R_6*S_4*S_3*R_4*S_2$
 $R_9*R_7*S_5*R_6*S_2*R_0*S_{15}$
 $R_9*R_7*S_5*R_6*S_2*R_2*R_0$
 $R_9*S_6*R_7*S_5*S_{21}*S_2*R_2$
 $R_9*S_6*R_7*S_5*S_{21}*S_3*S_{15}$
 $R_9*S_6*R_7*S_5*S_{21}*S_3*R_2$
 $R_9*S_6*R_7*S_5*S_{21}*S_3*S_{14}$
 $R_9*S_6*R_7*S_5*S_{21}*S_3*S_2$
 $R_9*S_6*R_7*S_5*S_{21}*S_4*S_3$
 $R_9*R_8*R_7*S_5*R_6*S_3*S_{14}$
 $R_9*R_8*R_7*S_5*R_6*S_4*S_{13}$
 $R_9*R_8*R_7*S_5*R_6*S_4*R_0$
 $R_9*R_8*R_7*S_5*R_6*S_4*S_2$
 $R_9*R_8*R_7*S_5*R_6*S_4*S_3$
 $R_9*S_7*S_{23}*R_7*S_4*S_{19}*R_2$
 $R_9*S_7*S_{23}*R_7*S_4*S_{19}*S_{17}$
 $R_9*S_7*S_{23}*R_7*S_5*R_4*R_2$
 $S_8*R_9*S_{23}*R_6*S_4*R_5*S_{15}$
 $S_8*R_9*S_{23}*R_6*S_4*R_5*R_2$
 $R_{10}*R_8*S_5*R_6*S_3*S_{18}*R_1$
 $R_{10}*R_8*S_5*R_6*S_4*S_{20}*S_2$
 $R_{10}*R_8*R_7*S_5*R_5*S_3*S_{17}$

$R_{10} * R_8 * R_7 * S_5 * R_5 * S_3 * S_{14}$
 $R_{10} * R_8 * R_7 * S_5 * S_4 * R_5 * S_3$
 $R_{10} * R_8 * S_6 * R_7 * S_3 * R_3 * S_{14}$
 $R_{10} * R_8 * S_6 * R_7 * R_5 * S_3 * S_{14}$
 $R_{10} * R_8 * S_6 * R_7 * S_5 * S_3 * S_{14}$
 $R_{10} * S_7 * R_8 * S_6 * S_{22} * S_3 * R_3$
 $R_{10} * S_7 * R_8 * S_6 * S_{22} * S_4 * S_{16}$
 $R_{10} * S_7 * R_8 * S_6 * S_{22} * S_4 * R_2$
 $R_{10} * S_7 * R_8 * S_6 * S_{22} * S_4 * R_3$
 $R_{10} * S_7 * R_8 * S_6 * S_{22} * S_4 * S_3$
 $R_{10} * S_7 * R_8 * S_6 * S_{22} * S_5 * S_4$
 $R_{10} * R_9 * R_8 * S_6 * R_7 * S_5 * R_1$
 $R_{10} * R_9 * R_8 * S_6 * R_7 * S_5 * S_{14}$
 $R_{10} * R_9 * R_8 * S_6 * R_7 * S_5 * S_3$
 $R_{10} * R_9 * R_8 * S_6 * R_7 * S_5 * S_4$
 $R_{10} * S_8 * S_{24} * R_8 * S_5 * S_{20} * R_3$
 $R_{10} * S_8 * S_{24} * R_8 * S_5 * S_{20} * S_{18}$
 $R_{10} * S_8 * S_{24} * R_8 * S_6 * R_5 * R_3$
 $S_9 * R_{10} * S_{24} * R_7 * S_5 * R_6 * S_{16}$
 $S_9 * R_{10} * S_{24} * R_7 * S_5 * R_6 * R_3$
 $R_{11} * R_9 * S_6 * R_7 * S_4 * S_{19} * R_2$
 $R_{11} * R_9 * S_6 * R_7 * S_5 * S_{21} * S_3$
 $R_{11} * R_9 * R_8 * S_6 * R_6 * S_4 * R_2$
 $R_{11} * R_9 * R_8 * S_6 * R_6 * S_4 * S_{18}$
 $R_{11} * R_9 * R_8 * S_6 * S_5 * R_6 * S_4$
 $R_{11} * R_9 * S_7 * R_8 * S_4 * S_{17} * R_2$
 $R_{11} * R_9 * S_7 * R_8 * S_4 * R_4 * R_2$
 $R_{11} * S_8 * R_9 * S_7 * S_{23} * S_4 * R_4$
 $R_{11} * S_8 * R_9 * S_7 * S_{23} * S_5 * S_{17}$
 $R_{11} * S_8 * R_9 * S_7 * S_{23} * S_5 * R_3$
 $R_{11} * S_8 * R_9 * S_7 * S_{23} * S_5 * R_4$

$R_{11} * S_8 * R_9 * S_7 * S_{23} * S_5 * S_4$
 $R_{11} * S_8 * R_9 * S_7 * S_{23} * S_6 * S_5$
 $R_{11} * S_8 * S_{24} * R_9 * S_6 * S_{21} * R_4$
 $R_{11} * S_8 * S_{24} * R_9 * S_7 * S_{23} * S_5$
 $R_{11} * R_{10} * R_9 * S_7 * R_8 * S_6 * S_{15}$
 $R_{11} * R_{10} * R_9 * S_7 * R_8 * S_6 * R_2$
 $R_{11} * R_{10} * R_9 * S_7 * R_8 * S_6 * S_4$
 $R_{11} * R_{10} * R_9 * S_7 * R_8 * S_6 * S_5$
 $R_{11} * R_{10} * S_8 * S_{24} * S_7 * R_8 * S_6$
 $R_{11} * S_9 * R_{10} * S_{24} * S_6 * S_{19} * R_4$
 $R_{11} * S_9 * R_{10} * S_{24} * S_6 * R_6 * R_4$
 $R_{11} * S_9 * S_{25} * R_9 * S_6 * S_{21} * R_4$
 $R_{11} * S_9 * S_{25} * R_9 * S_6 * S_{21} * S_{19}$
 $R_{11} * S_9 * S_{25} * R_9 * S_7 * R_6 * R_4$
 $S_{10} * R_{11} * S_{25} * R_8 * S_6 * R_7 * S_{17}$
 $S_{10} * R_{11} * S_{25} * R_8 * S_6 * R_7 * R_4$
 $S_{10} * R_{11} * S_9 * S_{25} * S_{24} * S_6 * R_6$
 $S_{10} * R_{11} * S_9 * S_{25} * S_{24} * S_7 * S_{19}$
 $S_{10} * R_{11} * S_9 * S_{25} * S_{24} * S_7 * R_5$
 $S_{10} * R_{11} * S_9 * S_{25} * S_{24} * S_7 * R_6$
 $S_{10} * R_{11} * S_9 * S_{25} * S_{24} * S_7 * S_6$
 $S_{10} * R_{11} * S_9 * S_{25} * S_8 * S_{24} * S_7$
 $S_{10} * S_{26} * R_{11} * S_9 * S_{25} * S_{24} * S_7$
 $R_{12} * R_{10} * S_7 * R_8 * S_5 * S_{20} * R_3$
 $R_{12} * R_{10} * S_7 * R_8 * S_6 * S_{22} * S_4$
 $R_{12} * R_{10} * R_9 * S_7 * R_7 * S_5 * R_3$
 $R_{12} * R_{10} * R_9 * S_7 * R_7 * S_5 * S_{19}$
 $R_{12} * R_{10} * R_9 * S_7 * S_6 * R_7 * S_5$
 $R_{12} * R_{10} * S_8 * R_9 * S_5 * S_{18} * R_3$
 $R_{12} * R_{10} * S_8 * R_9 * S_5 * R_5 * R_3$
 $R_{12} * S_9 * R_{10} * S_8 * S_{24} * S_5 * R_5$

$R_{12} * S_9 * R_{10} * S_8 * S_{24} * S_6 * S_{18}$
 $R_{12} * S_9 * R_{10} * S_8 * S_{24} * S_6 * R_4$
 $R_{12} * S_9 * R_{10} * S_8 * S_{24} * S_6 * R_5$
 $R_{12} * S_9 * R_{10} * S_8 * S_{24} * S_6 * S_5$
 $R_{12} * S_9 * R_{10} * S_8 * S_{24} * S_7 * S_6$
 $R_{12} * S_9 * S_{25} * R_{10} * S_7 * S_{22} * R_5$
 $R_{12} * S_9 * S_{25} * R_{10} * S_8 * S_{24} * S_6$
 $R_{12} * R_{11} * R_{10} * S_8 * R_9 * S_7 * S_{16}$
 $R_{12} * R_{11} * R_{10} * S_8 * R_9 * S_7 * R_3$
 $R_{12} * R_{11} * R_{10} * S_8 * R_9 * S_7 * S_5$
 $R_{12} * R_{11} * R_{10} * S_8 * R_9 * S_7 * S_6$
 $R_{12} * R_{11} * S_9 * R_{10} * S_8 * S_{24} * R_4$
 $R_{12} * R_{11} * S_9 * R_{10} * S_8 * S_{24} * S_6$
 $R_{12} * R_{11} * S_9 * R_{10} * S_8 * S_{24} * S_7$
 $R_{12} * R_{11} * S_9 * S_{25} * S_8 * R_9 * S_7$
 $R_{12} * S_{10} * R_{11} * S_{25} * S_7 * S_{20} * R_5$
 $R_{12} * S_{10} * R_{11} * S_{25} * S_7 * R_7 * R_5$
 $R_{12} * S_{10} * R_{11} * S_9 * S_{25} * S_{24} * R_5$
 $R_{12} * S_{10} * R_{11} * S_9 * S_{25} * S_{24} * S_7$
 $R_{12} * S_{10} * R_{11} * S_9 * S_{25} * S_8 * S_{24}$
 $R_{12} * S_{10} * S_{26} * R_{10} * S_7 * S_{22} * R_5$
 $R_{12} * S_{10} * S_{26} * R_{10} * S_7 * S_{22} * S_{20}$
 $R_{12} * S_{10} * S_{26} * R_{10} * S_8 * R_7 * R_5$
 $R_{12} * S_{10} * S_{26} * R_{11} * S_{25} * S_7 * R_7$
 $R_{12} * S_{10} * S_{26} * R_{11} * S_{25} * S_{24} * R_6$
 $R_{12} * S_{10} * S_{26} * R_{11} * S_{25} * S_8 * S_{20}$
 $R_{12} * S_{10} * S_{26} * R_{11} * S_{25} * S_8 * R_6$
 $R_{12} * S_{10} * S_{26} * R_{11} * S_{25} * S_8 * R_7$
 $R_{12} * S_{10} * S_{26} * R_{11} * S_{25} * S_8 * S_7$
 $R_{12} * S_{10} * S_{26} * R_{11} * S_{25} * S_8 * S_{24}$
 $R_{12} * S_{10} * S_{26} * R_{11} * S_9 * S_{25} * S_{24}$

$R_{12} * S_{10} * S_{26} * R_{11} * S_9 * S_{25} * S_8$

A.5 Знайдені базиси вгадування для 12 тактів шифра Струмок

$R_5 * S_3 * S_{19} * R_3 * R_0 * S_{15} * S_{11}$

$R_5 * S_3 * S_{19} * R_3 * S_1 * S_0 * S_{11}$

$R_5 * S_3 * S_{19} * R_3 * S_1 * R_0 * S_{11}$

$S_4 * R_5 * S_{19} * R_2 * R_0 * R_1 * S_{11}$

$S_4 * S_{20} * R_5 * S_2 * R_3 * S_1 * S_{14}$

$R_6 * R_4 * R_3 * S_1 * S_{12} * R_0 * S_0$

$R_6 * S_3 * R_4 * S_2 * S_{18} * S_{14} * S_{11}$

$R_6 * S_3 * R_4 * S_2 * S_{18} * S_1 * S_{12}$

$R_6 * R_5 * R_4 * S_2 * R_3 * S_0 * S_{11}$

$R_6 * R_5 * R_4 * S_2 * R_3 * S_1 * S_0$

$R_6 * S_4 * S_{20} * R_4 * S_2 * S_1 * S_{12}$

$S_5 * R_6 * S_{20} * R_3 * S_1 * S_{14} * S_{15}$

$S_5 * R_6 * S_{20} * S_2 * R_3 * S_1 * S_{14}$

$S_5 * R_6 * S_{20} * R_4 * R_3 * S_1 * S_{12}$

$R_7 * R_5 * S_2 * R_3 * S_1 * S_{14} * R_0$

$R_7 * R_5 * R_4 * S_2 * R_2 * R_1 * S_{11}$

$R_7 * R_5 * S_3 * R_4 * R_0 * S_0 * S_{11}$

$R_7 * R_5 * S_3 * R_4 * R_2 * S_0 * S_{11}$

$R_7 * R_5 * S_3 * R_4 * S_2 * S_0 * S_{11}$

$R_7 * S_4 * R_5 * S_3 * S_{19} * R_0 * S_0$

$R_7 * S_4 * R_5 * S_3 * S_{19} * S_1 * S_{13}$

$R_7 * S_4 * R_5 * S_3 * S_{19} * S_1 * S_0$

$R_7 * S_4 * R_5 * S_3 * S_{19} * S_1 * R_0$

$R_7 * S_4 * R_5 * S_3 * S_{19} * S_1 * S_{12}$

$R_7 * S_4 * R_5 * S_3 * S_{19} * S_2 * S_1$

$R_7 * R_6 * R_5 * S_3 * R_4 * S_1 * S_{12}$
 $R_7 * R_6 * R_5 * S_3 * R_4 * S_2 * S_{11}$
 $R_7 * R_6 * R_5 * S_3 * R_4 * S_2 * S_0$
 $R_7 * R_6 * R_5 * S_3 * R_4 * S_2 * S_1$
 $R_7 * S_5 * S_{21} * R_5 * S_2 * S_{17} * S_{15}$
 $R_7 * S_5 * S_{21} * R_5 * S_2 * S_{17} * R_0$
 $R_7 * S_5 * S_{21} * R_5 * S_3 * R_2 * R_0$
 $S_6 * R_7 * S_{21} * R_4 * S_2 * R_3 * S_{13}$
 $S_6 * R_7 * S_{21} * R_4 * S_2 * R_3 * R_0$
 $R_8 * R_6 * S_3 * R_4 * S_1 * S_{12} * R_2$
 $R_8 * R_6 * S_3 * R_4 * S_{18} * S_1 * S_{12}$
 $R_8 * R_6 * R_5 * S_3 * R_3 * S_1 * S_{15}$
 $R_8 * R_6 * R_5 * S_3 * R_3 * S_1 * S_{12}$
 $R_8 * R_6 * R_5 * S_3 * S_2 * R_3 * S_1$
 $R_8 * R_6 * S_4 * R_5 * S_1 * S_{12} * R_1$
 $R_8 * R_6 * S_4 * R_5 * R_3 * S_1 * S_{12}$
 $R_8 * R_6 * S_4 * R_5 * S_3 * S_1 * S_{12}$
 $R_8 * S_5 * R_6 * S_4 * S_{20} * S_1 * R_1$
 $R_8 * S_5 * R_6 * S_4 * S_{20} * S_2 * R_1$
 $R_8 * S_5 * R_6 * S_4 * S_{20} * S_2 * R_0$
 $R_8 * S_5 * R_6 * S_4 * S_{20} * S_2 * S_{14}$
 $R_8 * S_5 * R_6 * S_4 * S_{20} * S_2 * S_1$
 $R_8 * S_5 * R_6 * S_4 * S_{20} * S_3 * S_2$
 $R_8 * R_7 * R_6 * S_4 * R_5 * S_3 * S_{12}$
 $R_8 * R_7 * R_6 * S_4 * R_5 * S_3 * S_1$
 $R_8 * R_7 * R_6 * S_4 * R_5 * S_3 * S_2$
 $R_8 * S_6 * S_{22} * R_6 * S_3 * S_{18} * R_1$
 $R_8 * S_6 * S_{22} * R_6 * S_3 * S_{18} * S_{14}$
 $R_8 * S_6 * S_{22} * R_6 * S_4 * R_3 * S_{14}$
 $R_8 * S_6 * S_{22} * R_6 * S_4 * S_3 * S_{14}$
 $S_7 * R_8 * S_{22} * R_5 * S_3 * R_4 * R_1$

$S_7 * R_8 * S_{22} * R_5 * S_3 * R_4 * S_{14}$
 $R_9 * R_7 * S_4 * R_5 * S_2 * S_{17} * R_0$
 $R_9 * R_7 * S_4 * R_5 * S_3 * S_{19} * S_1$
 $R_9 * R_7 * R_6 * S_4 * R_4 * S_2 * S_{16}$
 $R_9 * R_7 * R_6 * S_4 * R_4 * S_2 * R_0$
 $R_9 * R_7 * R_6 * S_4 * S_3 * R_4 * S_2$
 $R_9 * R_7 * S_5 * R_6 * S_2 * R_0 * S_{15}$
 $R_9 * R_7 * S_5 * R_6 * S_2 * R_2 * R_0$
 $R_9 * S_6 * R_7 * S_5 * S_{21} * S_2 * R_2$
 $R_9 * S_6 * R_7 * S_5 * S_{21} * S_3 * S_{15}$
 $R_9 * S_6 * R_7 * S_5 * S_{21} * S_3 * R_2$
 $R_9 * S_6 * R_7 * S_5 * S_{21} * S_3 * S_{14}$
 $R_9 * S_6 * R_7 * S_5 * S_{21} * S_3 * S_2$
 $R_9 * S_6 * R_7 * S_5 * S_{21} * S_4 * S_3$
 $R_9 * R_8 * R_7 * S_5 * R_6 * S_3 * S_{14}$
 $R_9 * R_8 * R_7 * S_5 * R_6 * S_4 * S_{13}$
 $R_9 * R_8 * R_7 * S_5 * R_6 * S_4 * R_0$
 $R_9 * R_8 * R_7 * S_5 * R_6 * S_4 * S_2$
 $R_9 * R_8 * R_7 * S_5 * R_6 * S_4 * S_3$
 $R_9 * S_7 * S_{23} * R_7 * S_4 * S_{19} * R_2$
 $R_9 * S_7 * S_{23} * R_7 * S_4 * S_{19} * S_{17}$
 $R_9 * S_7 * S_{23} * R_7 * S_5 * R_4 * R_2$
 $S_8 * R_9 * S_{23} * R_6 * S_4 * R_5 * S_{15}$
 $S_8 * R_9 * S_{23} * R_6 * S_4 * R_5 * R_2$
 $R_{10} * R_8 * S_5 * R_6 * S_3 * S_{18} * R_1$
 $R_{10} * R_8 * S_5 * R_6 * S_4 * S_{20} * S_2$
 $R_{10} * R_8 * R_7 * S_5 * R_5 * S_3 * S_{17}$
 $R_{10} * R_8 * R_7 * S_5 * R_5 * S_3 * S_{14}$
 $R_{10} * R_8 * R_7 * S_5 * S_4 * R_5 * S_3$
 $R_{10} * R_8 * S_6 * R_7 * S_3 * R_3 * S_{14}$
 $R_{10} * R_8 * S_6 * R_7 * R_5 * S_3 * S_{14}$

$R_{10} * R_8 * S_6 * R_7 * S_5 * S_3 * S_{14}$
 $R_{10} * S_7 * R_8 * S_6 * S_{22} * S_3 * R_3$
 $R_{10} * S_7 * R_8 * S_6 * S_{22} * S_4 * S_{16}$
 $R_{10} * S_7 * R_8 * S_6 * S_{22} * S_4 * R_2$
 $R_{10} * S_7 * R_8 * S_6 * S_{22} * S_4 * R_3$
 $R_{10} * S_7 * R_8 * S_6 * S_{22} * S_4 * S_3$
 $R_{10} * S_7 * R_8 * S_6 * S_{22} * S_5 * S_4$
 $R_{10} * R_9 * R_8 * S_6 * R_7 * S_5 * R_1$
 $R_{10} * R_9 * R_8 * S_6 * R_7 * S_5 * S_{14}$
 $R_{10} * R_9 * R_8 * S_6 * R_7 * S_5 * S_3$
 $R_{10} * R_9 * R_8 * S_6 * R_7 * S_5 * S_4$
 $R_{10} * S_8 * S_{24} * R_8 * S_5 * S_{20} * R_3$
 $R_{10} * S_8 * S_{24} * R_8 * S_5 * S_{20} * S_{18}$
 $R_{10} * S_8 * S_{24} * R_8 * S_6 * R_5 * R_3$
 $S_9 * R_{10} * S_{24} * R_7 * S_5 * R_6 * S_{16}$
 $S_9 * R_{10} * S_{24} * R_7 * S_5 * R_6 * R_3$
 $R_{11} * R_9 * S_6 * R_7 * S_4 * S_{19} * R_2$
 $R_{11} * R_9 * S_6 * R_7 * S_5 * S_{21} * S_3$
 $R_{11} * R_9 * R_8 * S_6 * R_6 * S_4 * R_2$
 $R_{11} * R_9 * R_8 * S_6 * R_6 * S_4 * S_{18}$
 $R_{11} * R_9 * R_8 * S_6 * S_5 * R_6 * S_4$
 $R_{11} * R_9 * S_7 * R_8 * S_4 * S_{17} * R_2$
 $R_{11} * R_9 * S_7 * R_8 * S_4 * R_4 * R_2$
 $R_{11} * S_8 * R_9 * S_7 * S_{23} * S_4 * R_4$
 $R_{11} * S_8 * R_9 * S_7 * S_{23} * S_5 * S_{17}$
 $R_{11} * S_8 * R_9 * S_7 * S_{23} * S_5 * R_3$
 $R_{11} * S_8 * R_9 * S_7 * S_{23} * S_5 * R_4$
 $R_{11} * S_8 * R_9 * S_7 * S_{23} * S_5 * S_4$
 $R_{11} * S_8 * R_9 * S_7 * S_{23} * S_6 * S_5$
 $R_{11} * R_{10} * R_9 * S_7 * R_8 * S_6 * S_{15}$
 $R_{11} * R_{10} * R_9 * S_7 * R_8 * S_6 * R_2$

$R_{11} * R_{10} * R_9 * S_7 * R_8 * S_6 * S_4$
 $R_{11} * R_{10} * R_9 * S_7 * R_8 * S_6 * S_5$
 $R_{11} * S_9 * S_{25} * R_9 * S_6 * S_{21} * R_4$
 $R_{11} * S_9 * S_{25} * R_9 * S_6 * S_{21} * S_{19}$
 $R_{11} * S_9 * S_{25} * R_9 * S_7 * R_6 * R_4$
 $S_{10} * R_{11} * S_{25} * R_8 * S_6 * R_7 * S_{17}$
 $S_{10} * R_{11} * S_{25} * R_8 * S_6 * R_7 * R_4$
 $R_{12} * R_{10} * S_7 * R_8 * S_5 * S_{20} * R_3$
 $R_{12} * R_{10} * S_7 * R_8 * S_6 * S_{22} * S_4$
 $R_{12} * R_{10} * R_9 * S_7 * R_7 * S_5 * R_3$
 $R_{12} * R_{10} * R_9 * S_7 * R_7 * S_5 * S_{19}$
 $R_{12} * R_{10} * R_9 * S_7 * S_6 * R_7 * S_5$
 $R_{12} * R_{10} * S_8 * R_9 * S_5 * S_{18} * R_3$
 $R_{12} * R_{10} * S_8 * R_9 * S_5 * R_5 * R_3$
 $R_{12} * S_9 * R_{10} * S_8 * S_{24} * S_5 * R_5$
 $R_{12} * S_9 * R_{10} * S_8 * S_{24} * S_6 * S_{18}$
 $R_{12} * S_9 * R_{10} * S_8 * S_{24} * S_6 * R_4$
 $R_{12} * S_9 * R_{10} * S_8 * S_{24} * S_6 * R_5$
 $R_{12} * S_9 * R_{10} * S_8 * S_{24} * S_6 * S_5$
 $R_{12} * S_9 * R_{10} * S_8 * S_{24} * S_7 * S_6$
 $R_{12} * S_9 * S_{25} * R_{10} * S_7 * S_{22} * R_5$
 $R_{12} * S_9 * S_{25} * R_{10} * S_8 * S_{24} * S_6$
 $R_{12} * R_{11} * R_{10} * S_8 * R_9 * S_7 * S_{16}$
 $R_{12} * R_{11} * R_{10} * S_8 * R_9 * S_7 * R_3$
 $R_{12} * R_{11} * R_{10} * S_8 * R_9 * S_7 * S_5$
 $R_{12} * R_{11} * R_{10} * S_8 * R_9 * S_7 * S_6$
 $R_{12} * R_{11} * S_9 * S_{25} * S_8 * R_9 * S_7$
 $R_{12} * S_{10} * R_{11} * S_{25} * S_7 * S_{20} * R_5$
 $R_{12} * S_{10} * R_{11} * S_{25} * S_7 * R_7 * R_5$
 $R_{12} * S_{10} * S_{26} * R_{10} * S_7 * S_{22} * R_5$
 $R_{12} * S_{10} * S_{26} * R_{10} * S_7 * S_{22} * S_{20}$

$R_{12} * S_{10} * S_{26} * R_{10} * S_8 * R_7 * R_5$
 $R_{12} * S_{10} * S_{26} * R_{11} * S_{25} * S_7 * R_7$
 $R_{12} * S_{10} * S_{26} * R_{11} * S_{25} * S_8 * S_{20}$
 $R_{12} * S_{10} * S_{26} * R_{11} * S_{25} * S_8 * R_6$
 $R_{12} * S_{10} * S_{26} * R_{11} * S_{25} * S_8 * R_7$
 $R_{12} * S_{10} * S_{26} * R_{11} * S_{25} * S_8 * S_7$
 $R_{12} * S_{10} * S_{26} * R_{11} * S_9 * S_{25} * S_8$
 $S_{27} * R_{12} * S_{10} * S_{26} * R_{11} * S_{25} * S_8$
 $R_{13} * R_{11} * S_8 * R_9 * S_6 * S_{21} * R_4$
 $R_{13} * R_{11} * S_8 * R_9 * S_7 * S_{23} * S_5$
 $R_{13} * R_{11} * R_{10} * S_8 * R_8 * S_6 * R_4$
 $R_{13} * R_{11} * R_{10} * S_8 * R_8 * S_6 * S_{20}$
 $R_{13} * R_{11} * R_{10} * S_8 * S_7 * R_8 * S_6$
 $R_{13} * R_{11} * S_9 * R_{10} * S_6 * S_{19} * R_4$
 $R_{13} * R_{11} * S_9 * R_{10} * S_6 * R_6 * R_4$
 $R_{13} * S_{10} * R_{11} * S_9 * S_{25} * S_6 * R_6$
 $R_{13} * S_{10} * R_{11} * S_9 * S_{25} * S_7 * S_{19}$
 $R_{13} * S_{10} * R_{11} * S_9 * S_{25} * S_7 * R_5$
 $R_{13} * S_{10} * R_{11} * S_9 * S_{25} * S_7 * R_6$
 $R_{13} * S_{10} * R_{11} * S_9 * S_{25} * S_7 * S_6$
 $R_{13} * S_{10} * R_{11} * S_9 * S_{25} * S_8 * S_7$
 $R_{13} * S_{10} * S_{26} * R_{11} * S_8 * S_{23} * R_6$
 $R_{13} * S_{10} * S_{26} * R_{11} * S_9 * S_{25} * S_7$
 $R_{13} * R_{12} * R_{11} * S_9 * R_{10} * S_8 * S_{17}$
 $R_{13} * R_{12} * R_{11} * S_9 * R_{10} * S_8 * R_4$
 $R_{13} * R_{12} * R_{11} * S_9 * R_{10} * S_8 * S_6$
 $R_{13} * R_{12} * R_{11} * S_9 * R_{10} * S_8 * S_7$
 $R_{13} * R_{12} * S_{10} * R_{11} * S_9 * S_{25} * R_5$
 $R_{13} * R_{12} * S_{10} * R_{11} * S_9 * S_{25} * S_7$
 $R_{13} * R_{12} * S_{10} * R_{11} * S_9 * S_{25} * S_8$
 $R_{13} * R_{12} * S_{10} * S_{26} * S_9 * R_{10} * S_8$

$R_{13} * R_{12} * S_{10} * S_{26} * R_{11} * S_{25} * R_6$
 $R_{13} * R_{12} * S_{10} * S_{26} * R_{11} * S_{25} * S_8$
 $R_{13} * R_{12} * S_{10} * S_{26} * R_{11} * S_9 * S_{25}$
 $R_{13} * S_{27} * S_{10} * S_{26} * R_{11} * S_9 * S_{25}$
 $R_{13} * S_{27} * R_{12} * S_{26} * R_{11} * S_8 * R_8$
 $R_{13} * S_{27} * R_{12} * S_{26} * R_{11} * S_{25} * R_7$
 $R_{13} * S_{27} * R_{12} * S_{26} * R_{11} * S_9 * S_{21}$
 $R_{13} * S_{27} * R_{12} * S_{26} * R_{11} * S_9 * R_7$
 $R_{13} * S_{27} * R_{12} * S_{26} * R_{11} * S_9 * R_8$
 $R_{13} * S_{27} * R_{12} * S_{26} * R_{11} * S_9 * S_8$
 $R_{13} * S_{27} * R_{12} * S_{26} * R_{11} * S_9 * S_{25}$
 $R_{13} * S_{27} * R_{12} * S_{10} * R_{11} * S_9 * S_{25}$
 $R_{13} * S_{27} * R_{12} * S_{10} * S_{26} * R_{11} * S_{25}$
 $R_{13} * S_{27} * R_{12} * S_{10} * S_{26} * R_{11} * S_9$

А.6 Знайдені базиси вгадування для 13 тактів шифра

Струмок

$R_5 * S_3 * S_{19} * R_3 * R_0 * S_{15} * S_{11}$
 $R_5 * S_3 * S_{19} * R_3 * S_1 * S_0 * S_{11}$
 $R_5 * S_3 * S_{19} * R_3 * S_1 * R_0 * S_{11}$
 $S_4 * R_5 * S_{19} * R_2 * R_0 * R_1 * S_{11}$
 $S_4 * S_{20} * R_5 * S_2 * R_3 * S_1 * S_{14}$
 $R_6 * R_4 * R_3 * S_1 * S_{12} * R_0 * S_0$
 $R_6 * S_3 * R_4 * S_2 * S_{18} * S_{14} * S_{11}$
 $R_6 * S_3 * R_4 * S_2 * S_{18} * S_1 * S_{12}$
 $R_6 * R_5 * R_4 * S_2 * R_3 * S_0 * S_{11}$
 $R_6 * R_5 * R_4 * S_2 * R_3 * S_1 * S_0$
 $R_6 * S_4 * S_{20} * R_4 * S_2 * S_1 * S_{12}$
 $S_5 * R_6 * S_{20} * R_3 * S_1 * S_{14} * S_{15}$

$S_5 * R_6 * S_{20} * S_2 * R_3 * S_1 * S_{14}$
 $S_5 * R_6 * S_{20} * R_4 * R_3 * S_1 * S_{12}$
 $R_7 * R_5 * S_2 * R_3 * S_1 * S_{14} * R_0$
 $R_7 * R_5 * R_4 * S_2 * R_2 * R_1 * S_{11}$
 $R_7 * R_5 * S_3 * R_4 * R_0 * S_0 * S_{11}$
 $R_7 * R_5 * S_3 * R_4 * R_2 * S_0 * S_{11}$
 $R_7 * R_5 * S_3 * R_4 * S_2 * S_0 * S_{11}$
 $R_7 * S_4 * R_5 * S_3 * S_{19} * R_0 * S_0$
 $R_7 * S_4 * R_5 * S_3 * S_{19} * S_1 * S_{13}$
 $R_7 * S_4 * R_5 * S_3 * S_{19} * S_1 * S_0$
 $R_7 * S_4 * R_5 * S_3 * S_{19} * S_1 * R_0$
 $R_7 * S_4 * R_5 * S_3 * S_{19} * S_1 * S_{12}$
 $R_7 * S_4 * R_5 * S_3 * S_{19} * S_2 * S_1$
 $R_7 * R_6 * R_5 * S_3 * R_4 * S_1 * S_{12}$
 $R_7 * R_6 * R_5 * S_3 * R_4 * S_2 * S_{11}$
 $R_7 * R_6 * R_5 * S_3 * R_4 * S_2 * S_0$
 $R_7 * R_6 * R_5 * S_3 * R_4 * S_2 * S_1$
 $R_7 * S_5 * S_{21} * R_5 * S_2 * S_{17} * S_{15}$
 $R_7 * S_5 * S_{21} * R_5 * S_2 * S_{17} * R_0$
 $R_7 * S_5 * S_{21} * R_5 * S_3 * R_2 * R_0$
 $S_6 * R_7 * S_{21} * R_4 * S_2 * R_3 * S_{13}$
 $S_6 * R_7 * S_{21} * R_4 * S_2 * R_3 * R_0$
 $R_8 * R_6 * S_3 * R_4 * S_1 * S_{12} * R_2$
 $R_8 * R_6 * S_3 * R_4 * S_{18} * S_1 * S_{12}$
 $R_8 * R_6 * R_5 * S_3 * R_3 * S_1 * S_{15}$
 $R_8 * R_6 * R_5 * S_3 * R_3 * S_1 * S_{12}$
 $R_8 * R_6 * R_5 * S_3 * S_2 * R_3 * S_1$
 $R_8 * R_6 * S_4 * R_5 * S_1 * S_{12} * R_1$
 $R_8 * R_6 * S_4 * R_5 * R_3 * S_1 * S_{12}$
 $R_8 * R_6 * S_4 * R_5 * S_3 * S_1 * S_{12}$
 $R_8 * S_5 * R_6 * S_4 * S_{20} * S_1 * R_1$

$R_8*S_5*R_6*S_4*S_{20}*S_2*R_1$
 $R_8*S_5*R_6*S_4*S_{20}*S_2*R_0$
 $R_8*S_5*R_6*S_4*S_{20}*S_2*S_{14}$
 $R_8*S_5*R_6*S_4*S_{20}*S_2*S_1$
 $R_8*S_5*R_6*S_4*S_{20}*S_3*S_2$
 $R_8*R_7*R_6*S_4*R_5*S_3*S_{12}$
 $R_8*R_7*R_6*S_4*R_5*S_3*S_1$
 $R_8*R_7*R_6*S_4*R_5*S_3*S_2$
 $R_8*S_6*S_{22}*R_6*S_3*S_{18}*R_1$
 $R_8*S_6*S_{22}*R_6*S_3*S_{18}*S_{14}$
 $R_8*S_6*S_{22}*R_6*S_4*R_3*S_{14}$
 $R_8*S_6*S_{22}*R_6*S_4*S_3*S_{14}$
 $S_7*R_8*S_{22}*R_5*S_3*R_4*R_1$
 $S_7*R_8*S_{22}*R_5*S_3*R_4*S_{14}$
 $R_9*R_7*S_4*R_5*S_2*S_{17}*R_0$
 $R_9*R_7*S_4*R_5*S_3*S_{19}*S_1$
 $R_9*R_7*R_6*S_4*R_4*S_2*S_{16}$
 $R_9*R_7*R_6*S_4*R_4*S_2*R_0$
 $R_9*R_7*R_6*S_4*S_3*R_4*S_2$
 $R_9*R_7*S_5*R_6*S_2*R_0*S_{15}$
 $R_9*R_7*S_5*R_6*S_2*R_2*R_0$
 $R_9*S_6*R_7*S_5*S_{21}*S_2*R_2$
 $R_9*S_6*R_7*S_5*S_{21}*S_3*S_{15}$
 $R_9*S_6*R_7*S_5*S_{21}*S_3*R_2$
 $R_9*S_6*R_7*S_5*S_{21}*S_3*S_{14}$
 $R_9*S_6*R_7*S_5*S_{21}*S_3*S_2$
 $R_9*S_6*R_7*S_5*S_{21}*S_4*S_3$
 $R_9*R_8*R_7*S_5*R_6*S_3*S_{14}$
 $R_9*R_8*R_7*S_5*R_6*S_4*S_{13}$
 $R_9*R_8*R_7*S_5*R_6*S_4*R_0$
 $R_9*R_8*R_7*S_5*R_6*S_4*S_2$

$R_9 * R_8 * R_7 * S_5 * R_6 * S_4 * S_3$
 $R_9 * S_7 * S_{23} * R_7 * S_4 * S_{19} * R_2$
 $R_9 * S_7 * S_{23} * R_7 * S_4 * S_{19} * S_{17}$
 $R_9 * S_7 * S_{23} * R_7 * S_5 * R_4 * R_2$
 $S_8 * R_9 * S_{23} * R_6 * S_4 * R_5 * S_{15}$
 $S_8 * R_9 * S_{23} * R_6 * S_4 * R_5 * R_2$
 $R_{10} * R_8 * S_5 * R_6 * S_3 * S_{18} * R_1$
 $R_{10} * R_8 * S_5 * R_6 * S_4 * S_{20} * S_2$
 $R_{10} * R_8 * R_7 * S_5 * R_5 * S_3 * S_{17}$
 $R_{10} * R_8 * R_7 * S_5 * R_5 * S_3 * S_{14}$
 $R_{10} * R_8 * R_7 * S_5 * S_4 * R_5 * S_3$
 $R_{10} * R_8 * S_6 * R_7 * S_3 * R_3 * S_{14}$
 $R_{10} * R_8 * S_6 * R_7 * R_5 * S_3 * S_{14}$
 $R_{10} * R_8 * S_6 * R_7 * S_5 * S_3 * S_{14}$
 $R_{10} * S_7 * R_8 * S_6 * S_{22} * S_3 * R_3$
 $R_{10} * S_7 * R_8 * S_6 * S_{22} * S_4 * S_{16}$
 $R_{10} * S_7 * R_8 * S_6 * S_{22} * S_4 * R_2$
 $R_{10} * S_7 * R_8 * S_6 * S_{22} * S_4 * R_3$
 $R_{10} * S_7 * R_8 * S_6 * S_{22} * S_4 * S_3$
 $R_{10} * S_7 * R_8 * S_6 * S_{22} * S_5 * S_4$
 $R_{10} * R_9 * R_8 * S_6 * R_7 * S_5 * R_1$
 $R_{10} * R_9 * R_8 * S_6 * R_7 * S_5 * S_{14}$
 $R_{10} * R_9 * R_8 * S_6 * R_7 * S_5 * S_3$
 $R_{10} * R_9 * R_8 * S_6 * R_7 * S_5 * S_4$
 $R_{10} * S_8 * S_{24} * R_8 * S_5 * S_{20} * R_3$
 $R_{10} * S_8 * S_{24} * R_8 * S_5 * S_{20} * S_{18}$
 $R_{10} * S_8 * S_{24} * R_8 * S_6 * R_5 * R_3$
 $S_9 * R_{10} * S_{24} * R_7 * S_5 * R_6 * S_{16}$
 $S_9 * R_{10} * S_{24} * R_7 * S_5 * R_6 * R_3$
 $R_{11} * R_9 * S_6 * R_7 * S_4 * S_{19} * R_2$
 $R_{11} * R_9 * S_6 * R_7 * S_5 * S_{21} * S_3$

$R_{11} \cdot R_9 \cdot R_8 \cdot S_6 \cdot R_6 \cdot S_4 \cdot R_2$
 $R_{11} \cdot R_9 \cdot R_8 \cdot S_6 \cdot R_6 \cdot S_4 \cdot S_{18}$
 $R_{11} \cdot R_9 \cdot R_8 \cdot S_6 \cdot S_5 \cdot R_6 \cdot S_4$
 $R_{11} \cdot R_9 \cdot S_7 \cdot R_8 \cdot S_4 \cdot S_{17} \cdot R_2$
 $R_{11} \cdot R_9 \cdot S_7 \cdot R_8 \cdot S_4 \cdot R_4 \cdot R_2$
 $R_{11} \cdot S_8 \cdot R_9 \cdot S_7 \cdot S_{23} \cdot S_4 \cdot R_4$
 $R_{11} \cdot S_8 \cdot R_9 \cdot S_7 \cdot S_{23} \cdot S_5 \cdot S_{17}$
 $R_{11} \cdot S_8 \cdot R_9 \cdot S_7 \cdot S_{23} \cdot S_5 \cdot R_3$
 $R_{11} \cdot S_8 \cdot R_9 \cdot S_7 \cdot S_{23} \cdot S_5 \cdot R_4$
 $R_{11} \cdot S_8 \cdot R_9 \cdot S_7 \cdot S_{23} \cdot S_5 \cdot S_4$
 $R_{11} \cdot S_8 \cdot R_9 \cdot S_7 \cdot S_{23} \cdot S_6 \cdot S_5$
 $R_{11} \cdot R_{10} \cdot R_9 \cdot S_7 \cdot R_8 \cdot S_6 \cdot S_{15}$
 $R_{11} \cdot R_{10} \cdot R_9 \cdot S_7 \cdot R_8 \cdot S_6 \cdot R_2$
 $R_{11} \cdot R_{10} \cdot R_9 \cdot S_7 \cdot R_8 \cdot S_6 \cdot S_4$
 $R_{11} \cdot R_{10} \cdot R_9 \cdot S_7 \cdot R_8 \cdot S_6 \cdot S_5$
 $R_{11} \cdot S_9 \cdot S_{25} \cdot R_9 \cdot S_6 \cdot S_{21} \cdot R_4$
 $R_{11} \cdot S_9 \cdot S_{25} \cdot R_9 \cdot S_6 \cdot S_{21} \cdot S_{19}$
 $R_{11} \cdot S_9 \cdot S_{25} \cdot R_9 \cdot S_7 \cdot R_6 \cdot R_4$
 $S_{10} \cdot R_{11} \cdot S_{25} \cdot R_8 \cdot S_6 \cdot R_7 \cdot S_{17}$
 $S_{10} \cdot R_{11} \cdot S_{25} \cdot R_8 \cdot S_6 \cdot R_7 \cdot R_4$
 $R_{12} \cdot R_{10} \cdot S_7 \cdot R_8 \cdot S_5 \cdot S_{20} \cdot R_3$
 $R_{12} \cdot R_{10} \cdot S_7 \cdot R_8 \cdot S_6 \cdot S_{22} \cdot S_4$
 $R_{12} \cdot R_{10} \cdot R_9 \cdot S_7 \cdot R_7 \cdot S_5 \cdot R_3$
 $R_{12} \cdot R_{10} \cdot R_9 \cdot S_7 \cdot R_7 \cdot S_5 \cdot S_{19}$
 $R_{12} \cdot R_{10} \cdot R_9 \cdot S_7 \cdot S_6 \cdot R_7 \cdot S_5$
 $R_{12} \cdot R_{10} \cdot S_8 \cdot R_9 \cdot S_5 \cdot S_{18} \cdot R_3$
 $R_{12} \cdot R_{10} \cdot S_8 \cdot R_9 \cdot S_5 \cdot R_5 \cdot R_3$
 $R_{12} \cdot S_9 \cdot R_{10} \cdot S_8 \cdot S_{24} \cdot S_5 \cdot R_5$
 $R_{12} \cdot S_9 \cdot R_{10} \cdot S_8 \cdot S_{24} \cdot S_6 \cdot S_{18}$
 $R_{12} \cdot S_9 \cdot R_{10} \cdot S_8 \cdot S_{24} \cdot S_6 \cdot R_4$
 $R_{12} \cdot S_9 \cdot R_{10} \cdot S_8 \cdot S_{24} \cdot S_6 \cdot R_5$

$R_{12} * S_9 * R_{10} * S_8 * S_{24} * S_6 * S_5$
 $R_{12} * S_9 * R_{10} * S_8 * S_{24} * S_7 * S_6$
 $R_{12} * R_{11} * R_{10} * S_8 * R_9 * S_7 * S_{16}$
 $R_{12} * R_{11} * R_{10} * S_8 * R_9 * S_7 * R_3$
 $R_{12} * R_{11} * R_{10} * S_8 * R_9 * S_7 * S_5$
 $R_{12} * R_{11} * R_{10} * S_8 * R_9 * S_7 * S_6$
 $R_{12} * S_{10} * S_{26} * R_{10} * S_7 * S_{22} * R_5$
 $R_{12} * S_{10} * S_{26} * R_{10} * S_7 * S_{22} * S_{20}$
 $R_{12} * S_{10} * S_{26} * R_{10} * S_8 * R_7 * R_5$
 $R_{13} * R_{11} * S_8 * R_9 * S_6 * S_{21} * R_4$
 $R_{13} * R_{11} * S_8 * R_9 * S_7 * S_{23} * S_5$
 $R_{13} * R_{11} * R_{10} * S_8 * R_8 * S_6 * R_4$
 $R_{13} * R_{11} * R_{10} * S_8 * R_8 * S_6 * S_{20}$
 $R_{13} * R_{11} * R_{10} * S_8 * S_7 * R_8 * S_6$
 $R_{13} * R_{11} * S_9 * R_{10} * S_6 * S_{19} * R_4$
 $R_{13} * R_{11} * S_9 * R_{10} * S_6 * R_6 * R_4$
 $R_{13} * S_{10} * R_{11} * S_9 * S_{25} * S_6 * R_6$
 $R_{13} * S_{10} * R_{11} * S_9 * S_{25} * S_7 * S_{19}$
 $R_{13} * S_{10} * R_{11} * S_9 * S_{25} * S_7 * R_5$
 $R_{13} * S_{10} * R_{11} * S_9 * S_{25} * S_7 * R_6$
 $R_{13} * S_{10} * R_{11} * S_9 * S_{25} * S_7 * S_6$
 $R_{13} * S_{10} * R_{11} * S_9 * S_{25} * S_8 * S_7$
 $R_{13} * S_{10} * S_{26} * R_{11} * S_8 * S_{23} * R_6$
 $R_{13} * S_{10} * S_{26} * R_{11} * S_9 * S_{25} * S_7$
 $R_{13} * R_{12} * R_{11} * S_9 * R_{10} * S_8 * S_{17}$
 $R_{13} * R_{12} * R_{11} * S_9 * R_{10} * S_8 * R_4$
 $R_{13} * R_{12} * R_{11} * S_9 * R_{10} * S_8 * S_6$
 $R_{13} * R_{12} * R_{11} * S_9 * R_{10} * S_8 * S_7$
 $R_{13} * R_{12} * S_{10} * S_{26} * S_9 * R_{10} * S_8$
 $R_{13} * S_{27} * S_{10} * S_{26} * R_{11} * S_9 * S_{25}$
 $R_{13} * S_{27} * R_{12} * S_{26} * R_{11} * S_8 * R_8$

$R_{13} * S_{27} * R_{12} * S_{26} * R_{11} * S_9 * S_{21}$
 $R_{13} * S_{27} * R_{12} * S_{26} * R_{11} * S_9 * R_7$
 $R_{13} * S_{27} * R_{12} * S_{26} * R_{11} * S_9 * R_8$
 $R_{13} * S_{27} * R_{12} * S_{26} * R_{11} * S_9 * S_8$
 $R_{13} * S_{27} * R_{12} * S_{10} * S_{26} * R_{11} * S_9$
 $S_{28} * R_{13} * S_{27} * R_{12} * S_{26} * R_{11} * S_9$
 $R_{14} * R_{12} * S_9 * R_{10} * S_7 * S_{22} * R_5$
 $R_{14} * R_{12} * S_9 * R_{10} * S_8 * S_{24} * S_6$
 $R_{14} * R_{12} * R_{11} * S_9 * R_9 * S_7 * R_5$
 $R_{14} * R_{12} * R_{11} * S_9 * R_9 * S_7 * S_{21}$
 $R_{14} * R_{12} * R_{11} * S_9 * S_8 * R_9 * S_7$
 $R_{14} * R_{12} * S_{10} * R_{11} * S_7 * S_{20} * R_5$
 $R_{14} * R_{12} * S_{10} * R_{11} * S_7 * R_7 * R_5$
 $R_{14} * R_{12} * S_{10} * S_{26} * R_{11} * S_7 * R_7$
 $R_{14} * R_{12} * S_{10} * S_{26} * R_{11} * S_8 * S_{20}$
 $R_{14} * R_{12} * S_{10} * S_{26} * R_{11} * S_8 * R_6$
 $R_{14} * R_{12} * S_{10} * S_{26} * R_{11} * S_8 * R_7$
 $R_{14} * R_{12} * S_{10} * S_{26} * R_{11} * S_8 * S_7$
 $R_{14} * R_{12} * S_{10} * S_{26} * R_{11} * S_9 * S_8$
 $R_{14} * S_{27} * R_{12} * S_{10} * S_{26} * R_{11} * S_8$
 $R_{14} * R_{13} * R_{12} * S_{10} * R_{11} * S_9 * S_{18}$
 $R_{14} * R_{13} * R_{12} * S_{10} * R_{11} * S_9 * R_5$
 $R_{14} * R_{13} * R_{12} * S_{10} * R_{11} * S_9 * S_7$
 $R_{14} * R_{13} * R_{12} * S_{10} * R_{11} * S_9 * S_8$
 $R_{14} * R_{13} * R_{12} * S_{10} * S_{26} * R_{11} * R_6$
 $R_{14} * R_{13} * R_{12} * S_{10} * S_{26} * R_{11} * S_8$
 $R_{14} * R_{13} * R_{12} * S_{10} * S_{26} * R_{11} * S_9$
 $R_{14} * R_{13} * S_{27} * R_{12} * S_{26} * R_{11} * R_7$
 $R_{14} * R_{13} * S_{27} * R_{12} * S_{26} * R_{11} * S_9$
 $R_{14} * S_{28} * R_{13} * S_{27} * R_{12} * S_9 * R_9$
 $R_{14} * S_{28} * R_{13} * S_{27} * R_{12} * S_{26} * R_8$

$R_{14} * S_{28} * R_{13} * S_{27} * R_{12} * S_{10} * S_{11}$
 $R_{14} * S_{28} * R_{13} * S_{27} * R_{12} * S_{10} * S_{22}$
 $R_{14} * S_{28} * R_{13} * S_{27} * R_{12} * S_{10} * R_8$
 $R_{14} * S_{28} * R_{13} * S_{27} * R_{12} * S_{10} * R_9$
 $R_{14} * S_{28} * R_{13} * S_{27} * R_{12} * S_{10} * S_9$
 $R_{14} * S_{28} * R_{13} * S_{27} * R_{12} * S_{10} * S_{26}$

А.7 Знайдені базиси вгадування для 14 тактів шифра Струмок

$R_5 * S_3 * S_{19} * R_3 * R_0 * S_{15} * S_{11}$
 $R_5 * S_3 * S_{19} * R_3 * S_1 * S_0 * S_{11}$
 $R_5 * S_3 * S_{19} * R_3 * S_1 * R_0 * S_{11}$
 $S_4 * R_5 * S_{19} * R_2 * R_0 * R_1 * S_{11}$
 $S_4 * S_{20} * R_5 * S_2 * R_3 * S_1 * S_{14}$
 $R_6 * R_4 * R_3 * S_1 * S_{12} * R_0 * S_0$
 $R_6 * S_3 * R_4 * S_2 * S_{18} * S_{14} * S_{11}$
 $R_6 * S_3 * R_4 * S_2 * S_{18} * S_1 * S_{12}$
 $R_6 * R_5 * R_4 * S_2 * R_3 * S_0 * S_{11}$
 $R_6 * R_5 * R_4 * S_2 * R_3 * S_1 * S_0$
 $R_6 * S_4 * S_{20} * R_4 * S_2 * S_1 * S_{12}$
 $S_5 * R_6 * S_{20} * R_3 * S_1 * S_{14} * S_{15}$
 $S_5 * R_6 * S_{20} * S_2 * R_3 * S_1 * S_{14}$
 $S_5 * R_6 * S_{20} * R_4 * R_3 * S_1 * S_{12}$
 $R_7 * R_5 * S_2 * R_3 * S_1 * S_{14} * R_0$
 $R_7 * R_5 * R_4 * S_2 * R_2 * R_1 * S_{11}$
 $R_7 * R_5 * S_3 * R_4 * R_0 * S_0 * S_{11}$
 $R_7 * R_5 * S_3 * R_4 * R_2 * S_0 * S_{11}$
 $R_7 * R_5 * S_3 * R_4 * S_2 * S_0 * S_{11}$
 $R_7 * S_4 * R_5 * S_3 * S_{19} * R_0 * S_0$

$R_7*S_4*R_5*S_3*S_{19}*S_1*S_{13}$
 $R_7*S_4*R_5*S_3*S_{19}*S_1*S_0$
 $R_7*S_4*R_5*S_3*S_{19}*S_1*R_0$
 $R_7*S_4*R_5*S_3*S_{19}*S_1*S_{12}$
 $R_7*S_4*R_5*S_3*S_{19}*S_2*S_1$
 $R_7*R_6*R_5*S_3*R_4*S_1*S_{12}$
 $R_7*R_6*R_5*S_3*R_4*S_2*S_{11}$
 $R_7*R_6*R_5*S_3*R_4*S_2*S_0$
 $R_7*R_6*R_5*S_3*R_4*S_2*S_1$
 $R_7*S_5*S_{21}*R_5*S_2*S_{17}*S_{15}$
 $R_7*S_5*S_{21}*R_5*S_2*S_{17}*R_0$
 $R_7*S_5*S_{21}*R_5*S_3*R_2*R_0$
 $S_6*R_7*S_{21}*R_4*S_2*R_3*S_{13}$
 $S_6*R_7*S_{21}*R_4*S_2*R_3*R_0$
 $R_8*R_6*S_3*R_4*S_1*S_{12}*R_2$
 $R_8*R_6*S_3*R_4*S_{18}*S_1*S_{12}$
 $R_8*R_6*R_5*S_3*R_3*S_1*S_{15}$
 $R_8*R_6*R_5*S_3*R_3*S_1*S_{12}$
 $R_8*R_6*R_5*S_3*S_2*R_3*S_1$
 $R_8*R_6*S_4*R_5*S_1*S_{12}*R_1$
 $R_8*R_6*S_4*R_5*R_3*S_1*S_{12}$
 $R_8*R_6*S_4*R_5*S_3*S_1*S_{12}$
 $R_8*S_5*R_6*S_4*S_{20}*S_1*R_1$
 $R_8*S_5*R_6*S_4*S_{20}*S_2*R_1$
 $R_8*S_5*R_6*S_4*S_{20}*S_2*R_0$
 $R_8*S_5*R_6*S_4*S_{20}*S_2*S_{14}$
 $R_8*S_5*R_6*S_4*S_{20}*S_2*S_1$
 $R_8*S_5*R_6*S_4*S_{20}*S_3*S_2$
 $R_8*R_7*R_6*S_4*R_5*S_3*S_{12}$
 $R_8*R_7*R_6*S_4*R_5*S_3*S_1$
 $R_8*R_7*R_6*S_4*R_5*S_3*S_2$

$R_8*S_6*S_{22}*R_6*S_3*S_{18}*R_1$
 $R_8*S_6*S_{22}*R_6*S_3*S_{18}*S_{14}$
 $R_8*S_6*S_{22}*R_6*S_4*R_3*S_{14}$
 $R_8*S_6*S_{22}*R_6*S_4*S_3*S_{14}$
 $S_7*R_8*S_{22}*R_5*S_3*R_4*R_1$
 $S_7*R_8*S_{22}*R_5*S_3*R_4*S_{14}$
 $R_9*R_7*S_4*R_5*S_2*S_{17}*R_0$
 $R_9*R_7*S_4*R_5*S_3*S_{19}*S_1$
 $R_9*R_7*R_6*S_4*R_4*S_2*S_{16}$
 $R_9*R_7*R_6*S_4*R_4*S_2*R_0$
 $R_9*R_7*R_6*S_4*S_3*R_4*S_2$
 $R_9*R_7*S_5*R_6*S_2*R_0*S_{15}$
 $R_9*R_7*S_5*R_6*S_2*R_2*R_0$
 $R_9*S_6*R_7*S_5*S_{21}*S_2*R_2$
 $R_9*S_6*R_7*S_5*S_{21}*S_3*S_{15}$
 $R_9*S_6*R_7*S_5*S_{21}*S_3*R_2$
 $R_9*S_6*R_7*S_5*S_{21}*S_3*S_{14}$
 $R_9*S_6*R_7*S_5*S_{21}*S_3*S_2$
 $R_9*S_6*R_7*S_5*S_{21}*S_4*S_3$
 $R_9*R_8*R_7*S_5*R_6*S_3*S_{14}$
 $R_9*R_8*R_7*S_5*R_6*S_4*S_{13}$
 $R_9*R_8*R_7*S_5*R_6*S_4*R_0$
 $R_9*R_8*R_7*S_5*R_6*S_4*S_2$
 $R_9*R_8*R_7*S_5*R_6*S_4*S_3$
 $R_9*S_7*S_{23}*R_7*S_4*S_{19}*R_2$
 $R_9*S_7*S_{23}*R_7*S_4*S_{19}*S_{17}$
 $R_9*S_7*S_{23}*R_7*S_5*R_4*R_2$
 $S_8*R_9*S_{23}*R_6*S_4*R_5*S_{15}$
 $S_8*R_9*S_{23}*R_6*S_4*R_5*R_2$
 $R_{10}*R_8*S_5*R_6*S_3*S_{18}*R_1$
 $R_{10}*R_8*S_5*R_6*S_4*S_{20}*S_2$

$R_{10}R_8R_7S_5R_5S_3S_{17}$
 $R_{10}R_8R_7S_5R_5S_3S_{14}$
 $R_{10}R_8R_7S_5S_4R_5S_3$
 $R_{10}R_8S_6R_7S_3R_3S_{14}$
 $R_{10}R_8S_6R_7R_5S_3S_{14}$
 $R_{10}R_8S_6R_7S_5S_3S_{14}$
 $R_{10}S_7R_8S_6S_{22}S_3R_3$
 $R_{10}S_7R_8S_6S_{22}S_4S_{16}$
 $R_{10}S_7R_8S_6S_{22}S_4R_2$
 $R_{10}S_7R_8S_6S_{22}S_4R_3$
 $R_{10}S_7R_8S_6S_{22}S_4S_3$
 $R_{10}S_7R_8S_6S_{22}S_5S_4$
 $R_{10}R_9R_8S_6R_7S_5R_1$
 $R_{10}R_9R_8S_6R_7S_5S_{14}$
 $R_{10}R_9R_8S_6R_7S_5S_3$
 $R_{10}R_9R_8S_6R_7S_5S_4$
 $R_{10}S_8S_{24}R_8S_5S_{20}R_3$
 $R_{10}S_8S_{24}R_8S_5S_{20}S_{18}$
 $R_{10}S_8S_{24}R_8S_6R_5R_3$
 $S_9R_{10}S_{24}R_7S_5R_6S_{16}$
 $S_9R_{10}S_{24}R_7S_5R_6R_3$
 $R_{11}R_9S_6R_7S_4S_{19}R_2$
 $R_{11}R_9S_6R_7S_5S_{21}S_3$
 $R_{11}R_9R_8S_6R_6S_4R_2$
 $R_{11}R_9R_8S_6R_6S_4S_{18}$
 $R_{11}R_9R_8S_6S_5R_6S_4$
 $R_{11}R_9S_7R_8S_4S_{17}R_2$
 $R_{11}R_9S_7R_8S_4R_4R_2$
 $R_{11}S_8R_9S_7S_{23}S_4R_4$
 $R_{11}S_8R_9S_7S_{23}S_5S_{17}$
 $R_{11}S_8R_9S_7S_{23}S_5R_3$

$R_{11} * S_8 * R_9 * S_7 * S_{23} * S_5 * R_4$
 $R_{11} * S_8 * R_9 * S_7 * S_{23} * S_5 * S_4$
 $R_{11} * S_8 * R_9 * S_7 * S_{23} * S_6 * S_5$
 $R_{11} * R_{10} * R_9 * S_7 * R_8 * S_6 * S_{15}$
 $R_{11} * R_{10} * R_9 * S_7 * R_8 * S_6 * R_2$
 $R_{11} * R_{10} * R_9 * S_7 * R_8 * S_6 * S_4$
 $R_{11} * R_{10} * R_9 * S_7 * R_8 * S_6 * S_5$
 $R_{11} * S_9 * S_{25} * R_9 * S_6 * S_{21} * R_4$
 $R_{11} * S_9 * S_{25} * R_9 * S_6 * S_{21} * S_{19}$
 $R_{11} * S_9 * S_{25} * R_9 * S_7 * R_6 * R_4$
 $S_{10} * R_{11} * S_{25} * R_8 * S_6 * R_7 * S_{17}$
 $S_{10} * R_{11} * S_{25} * R_8 * S_6 * R_7 * R_4$
 $R_{12} * R_{10} * S_7 * R_8 * S_5 * S_{20} * R_3$
 $R_{12} * R_{10} * S_7 * R_8 * S_6 * S_{22} * S_4$
 $R_{12} * R_{10} * R_9 * S_7 * R_7 * S_5 * R_3$
 $R_{12} * R_{10} * R_9 * S_7 * R_7 * S_5 * S_{19}$
 $R_{12} * R_{10} * R_9 * S_7 * S_6 * R_7 * S_5$
 $R_{12} * R_{10} * S_8 * R_9 * S_5 * S_{18} * R_3$
 $R_{12} * R_{10} * S_8 * R_9 * S_5 * R_5 * R_3$
 $R_{12} * S_9 * R_{10} * S_8 * S_{24} * S_5 * R_5$
 $R_{12} * S_9 * R_{10} * S_8 * S_{24} * S_6 * S_{18}$
 $R_{12} * S_9 * R_{10} * S_8 * S_{24} * S_6 * R_4$
 $R_{12} * S_9 * R_{10} * S_8 * S_{24} * S_6 * R_5$
 $R_{12} * S_9 * R_{10} * S_8 * S_{24} * S_6 * S_5$
 $R_{12} * S_9 * R_{10} * S_8 * S_{24} * S_7 * S_6$
 $R_{12} * R_{11} * R_{10} * S_8 * R_9 * S_7 * S_{16}$
 $R_{12} * R_{11} * R_{10} * S_8 * R_9 * S_7 * R_3$
 $R_{12} * R_{11} * R_{10} * S_8 * R_9 * S_7 * S_5$
 $R_{12} * R_{11} * R_{10} * S_8 * R_9 * S_7 * S_6$
 $R_{12} * S_{10} * S_{26} * R_{10} * S_7 * S_{22} * R_5$
 $R_{12} * S_{10} * S_{26} * R_{10} * S_7 * S_{22} * S_{20}$

$R_{12} * S_{10} * S_{26} * R_{10} * S_8 * R_7 * R_5$
 $R_{13} * R_{11} * S_8 * R_9 * S_6 * S_{21} * R_4$
 $R_{13} * R_{11} * S_8 * R_9 * S_7 * S_{23} * S_5$
 $R_{13} * R_{11} * R_{10} * S_8 * R_8 * S_6 * R_4$
 $R_{13} * R_{11} * R_{10} * S_8 * R_8 * S_6 * S_{20}$
 $R_{13} * R_{11} * R_{10} * S_8 * S_7 * R_8 * S_6$
 $R_{13} * R_{11} * S_9 * R_{10} * S_6 * S_{19} * R_4$
 $R_{13} * R_{11} * S_9 * R_{10} * S_6 * R_6 * R_4$
 $R_{13} * S_{10} * R_{11} * S_9 * S_{25} * S_6 * R_6$
 $R_{13} * S_{10} * R_{11} * S_9 * S_{25} * S_7 * S_{19}$
 $R_{13} * S_{10} * R_{11} * S_9 * S_{25} * S_7 * R_5$
 $R_{13} * S_{10} * R_{11} * S_9 * S_{25} * S_7 * R_6$
 $R_{13} * S_{10} * R_{11} * S_9 * S_{25} * S_7 * S_6$
 $R_{13} * S_{10} * R_{11} * S_9 * S_{25} * S_8 * S_7$
 $R_{13} * R_{12} * R_{11} * S_9 * R_{10} * S_8 * S_{17}$
 $R_{13} * R_{12} * R_{11} * S_9 * R_{10} * S_8 * R_4$
 $R_{13} * R_{12} * R_{11} * S_9 * R_{10} * S_8 * S_6$
 $R_{13} * R_{12} * R_{11} * S_9 * R_{10} * S_8 * S_7$
 $R_{14} * R_{12} * S_9 * R_{10} * S_7 * S_{22} * R_5$
 $R_{14} * R_{12} * S_9 * R_{10} * S_8 * S_{24} * S_6$
 $R_{14} * R_{12} * R_{11} * S_9 * R_9 * S_7 * R_5$
 $R_{14} * R_{12} * R_{11} * S_9 * R_9 * S_7 * S_{21}$
 $R_{14} * R_{12} * R_{11} * S_9 * S_8 * R_9 * S_7$
 $R_{14} * R_{12} * S_{10} * R_{11} * S_7 * S_{20} * R_5$
 $R_{14} * R_{12} * S_{10} * R_{11} * S_7 * R_7 * R_5$
 $R_{14} * R_{12} * S_{10} * S_{26} * R_{11} * S_7 * R_7$
 $R_{14} * R_{12} * S_{10} * S_{26} * R_{11} * S_8 * S_{20}$
 $R_{14} * R_{12} * S_{10} * S_{26} * R_{11} * S_8 * R_6$
 $R_{14} * R_{12} * S_{10} * S_{26} * R_{11} * S_8 * R_7$
 $R_{14} * R_{12} * S_{10} * S_{26} * R_{11} * S_8 * S_7$
 $R_{14} * R_{12} * S_{10} * S_{26} * R_{11} * S_9 * S_8$

$R_{14} * S_{27} * R_{12} * S_{10} * S_{26} * R_{11} * S_8$
 $R_{14} * R_{13} * R_{12} * S_{10} * R_{11} * S_9 * S_{18}$
 $R_{14} * R_{13} * R_{12} * S_{10} * R_{11} * S_9 * R_5$
 $R_{14} * R_{13} * R_{12} * S_{10} * R_{11} * S_9 * S_7$
 $R_{14} * R_{13} * R_{12} * S_{10} * R_{11} * S_9 * S_8$
 $R_{14} * S_{28} * R_{13} * S_{27} * R_{12} * S_9 * R_9$
 $R_{14} * S_{28} * R_{13} * S_{27} * R_{12} * S_{10} * S_{11}$
 $R_{14} * S_{28} * R_{13} * S_{27} * R_{12} * S_{10} * S_{22}$
 $R_{14} * S_{28} * R_{13} * S_{27} * R_{12} * S_{10} * R_8$
 $R_{14} * S_{28} * R_{13} * S_{27} * R_{12} * S_{10} * R_9$
 $R_{14} * S_{28} * R_{13} * S_{27} * R_{12} * S_{10} * S_9$
 $S_{29} * R_{14} * S_{28} * R_{13} * S_{27} * R_{12} * S_{10}$
 $R_{15} * R_{13} * S_{10} * R_{11} * S_8 * S_{23} * R_6$
 $R_{15} * R_{13} * S_{10} * R_{11} * S_9 * S_{25} * S_7$
 $R_{15} * R_{13} * R_{12} * S_{10} * R_{10} * S_8 * R_6$
 $R_{15} * R_{13} * R_{12} * S_{10} * R_{10} * S_8 * S_{22}$
 $R_{15} * R_{13} * R_{12} * S_{10} * S_9 * R_{10} * S_8$
 $R_{15} * R_{13} * S_{27} * S_{10} * R_{11} * S_9 * S_{25}$
 $R_{15} * R_{13} * S_{27} * R_{12} * R_{11} * S_8 * R_8$
 $R_{15} * R_{13} * S_{27} * R_{12} * R_{11} * S_9 * S_{21}$
 $R_{15} * R_{13} * S_{27} * R_{12} * R_{11} * S_9 * R_7$
 $R_{15} * R_{13} * S_{27} * R_{12} * R_{11} * S_9 * R_8$
 $R_{15} * R_{13} * S_{27} * R_{12} * R_{11} * S_9 * S_8$
 $R_{15} * R_{13} * S_{27} * R_{12} * S_{10} * R_{11} * S_9$
 $R_{15} * S_{28} * R_{13} * S_{27} * R_{12} * R_{11} * S_9$
 $R_{15} * R_{14} * R_{13} * R_{12} * S_{10} * R_{11} * S_{19}$
 $R_{15} * R_{14} * R_{13} * R_{12} * S_{10} * R_{11} * R_6$
 $R_{15} * R_{14} * R_{13} * R_{12} * S_{10} * R_{11} * S_8$
 $R_{15} * R_{14} * R_{13} * R_{12} * S_{10} * R_{11} * S_9$
 $R_{15} * R_{14} * R_{13} * S_{27} * R_{12} * R_{11} * R_7$
 $R_{15} * R_{14} * R_{13} * S_{27} * R_{12} * R_{11} * S_9$

$R_{15} * R_{14} * S_{28} * R_{13} * S_{27} * R_{12} * R_8$
 $R_{15} * R_{14} * S_{28} * R_{13} * S_{27} * R_{12} * S_{10}$
 $R_{15} * S_{29} * R_{14} * S_{28} * R_{13} * S_{10} * R_{10}$
 $R_{15} * S_{29} * R_{14} * S_{28} * R_{13} * R_{12} * R_9$
 $R_{15} * S_{29} * R_{14} * S_{28} * R_{13} * R_{12} * S_{10}$
 $R_{15} * S_{29} * R_{14} * S_{28} * R_{13} * S_{27} * R_9$
 $R_{15} * S_{29} * R_{14} * S_{28} * R_{13} * S_{27} * R_{12}$

A.8 Знайдені базиси вгадування для 15 тактів шифра

Струмук

$R_5 * S_3 * S_{19} * R_3 * R_0 * S_{15} * S_{11}$
 $R_5 * S_3 * S_{19} * R_3 * S_1 * S_0 * S_{11}$
 $R_5 * S_3 * S_{19} * R_3 * S_1 * R_0 * S_{11}$
 $S_4 * R_5 * S_{19} * R_2 * R_0 * R_1 * S_{11}$
 $S_4 * S_{20} * R_5 * S_2 * R_3 * S_1 * S_{14}$
 $R_6 * R_4 * R_3 * S_1 * S_{12} * R_0 * S_0$
 $R_6 * S_3 * R_4 * S_2 * S_{18} * S_{14} * S_{11}$
 $R_6 * S_3 * R_4 * S_2 * S_{18} * S_1 * S_{12}$
 $R_6 * R_5 * R_4 * S_2 * R_3 * S_0 * S_{11}$
 $R_6 * R_5 * R_4 * S_2 * R_3 * S_1 * S_0$
 $R_6 * S_4 * S_{20} * R_4 * S_2 * S_1 * S_{12}$
 $S_5 * R_6 * S_{20} * R_3 * S_1 * S_{14} * S_{15}$
 $S_5 * R_6 * S_{20} * S_2 * R_3 * S_1 * S_{14}$
 $S_5 * R_6 * S_{20} * R_4 * R_3 * S_1 * S_{12}$
 $R_7 * R_5 * S_2 * R_3 * S_1 * S_{14} * R_0$
 $R_7 * R_5 * R_4 * S_2 * R_2 * R_1 * S_{11}$
 $R_7 * R_5 * S_3 * R_4 * R_0 * S_0 * S_{11}$
 $R_7 * R_5 * S_3 * R_4 * R_2 * S_0 * S_{11}$
 $R_7 * R_5 * S_3 * R_4 * S_2 * S_0 * S_{11}$

$R_7*S_4*R_5*S_3*S_{19}*R_0*S_0$
 $R_7*S_4*R_5*S_3*S_{19}*S_1*S_{13}$
 $R_7*S_4*R_5*S_3*S_{19}*S_1*S_0$
 $R_7*S_4*R_5*S_3*S_{19}*S_1*R_0$
 $R_7*S_4*R_5*S_3*S_{19}*S_1*S_{12}$
 $R_7*S_4*R_5*S_3*S_{19}*S_2*S_1$
 $R_7*R_6*R_5*S_3*R_4*S_1*S_{12}$
 $R_7*R_6*R_5*S_3*R_4*S_2*S_{11}$
 $R_7*R_6*R_5*S_3*R_4*S_2*S_0$
 $R_7*R_6*R_5*S_3*R_4*S_2*S_1$
 $R_7*S_5*S_{21}*R_5*S_2*S_{17}*S_{15}$
 $R_7*S_5*S_{21}*R_5*S_2*S_{17}*R_0$
 $R_7*S_5*S_{21}*R_5*S_3*R_2*R_0$
 $S_6*R_7*S_{21}*R_4*S_2*R_3*S_{13}$
 $S_6*R_7*S_{21}*R_4*S_2*R_3*R_0$
 $R_8*R_6*S_3*R_4*S_1*S_{12}*R_2$
 $R_8*R_6*S_3*R_4*S_{18}*S_1*S_{12}$
 $R_8*R_6*R_5*S_3*R_3*S_1*S_{15}$
 $R_8*R_6*R_5*S_3*R_3*S_1*S_{12}$
 $R_8*R_6*R_5*S_3*S_2*R_3*S_1$
 $R_8*R_6*S_4*R_5*S_1*S_{12}*R_1$
 $R_8*R_6*S_4*R_5*R_3*S_1*S_{12}$
 $R_8*R_6*S_4*R_5*S_3*S_1*S_{12}$
 $R_8*S_5*R_6*S_4*S_{20}*S_1*R_1$
 $R_8*S_5*R_6*S_4*S_{20}*S_2*R_1$
 $R_8*S_5*R_6*S_4*S_{20}*S_2*R_0$
 $R_8*S_5*R_6*S_4*S_{20}*S_2*S_{14}$
 $R_8*S_5*R_6*S_4*S_{20}*S_2*S_1$
 $R_8*S_5*R_6*S_4*S_{20}*S_3*S_2$
 $R_8*R_7*R_6*S_4*R_5*S_3*S_{12}$
 $R_8*R_7*R_6*S_4*R_5*S_3*S_1$

$R_8 * R_7 * R_6 * S_4 * R_5 * S_3 * S_2$
 $R_8 * S_6 * S_{22} * R_6 * S_3 * S_{18} * R_1$
 $R_8 * S_6 * S_{22} * R_6 * S_3 * S_{18} * S_{14}$
 $R_8 * S_6 * S_{22} * R_6 * S_4 * R_3 * S_{14}$
 $R_8 * S_6 * S_{22} * R_6 * S_4 * S_3 * S_{14}$
 $S_7 * R_8 * S_{22} * R_5 * S_3 * R_4 * R_1$
 $S_7 * R_8 * S_{22} * R_5 * S_3 * R_4 * S_{14}$
 $R_9 * R_7 * S_4 * R_5 * S_2 * S_{17} * R_0$
 $R_9 * R_7 * S_4 * R_5 * S_3 * S_{19} * S_1$
 $R_9 * R_7 * R_6 * S_4 * R_4 * S_2 * S_{16}$
 $R_9 * R_7 * R_6 * S_4 * R_4 * S_2 * R_0$
 $R_9 * R_7 * R_6 * S_4 * S_3 * R_4 * S_2$
 $R_9 * R_7 * S_5 * R_6 * S_2 * R_0 * S_{15}$
 $R_9 * R_7 * S_5 * R_6 * S_2 * R_2 * R_0$
 $R_9 * S_6 * R_7 * S_5 * S_{21} * S_2 * R_2$
 $R_9 * S_6 * R_7 * S_5 * S_{21} * S_3 * S_{15}$
 $R_9 * S_6 * R_7 * S_5 * S_{21} * S_3 * R_2$
 $R_9 * S_6 * R_7 * S_5 * S_{21} * S_3 * S_{14}$
 $R_9 * S_6 * R_7 * S_5 * S_{21} * S_3 * S_2$
 $R_9 * S_6 * R_7 * S_5 * S_{21} * S_4 * S_3$
 $R_9 * R_8 * R_7 * S_5 * R_6 * S_3 * S_{14}$
 $R_9 * R_8 * R_7 * S_5 * R_6 * S_4 * S_{13}$
 $R_9 * R_8 * R_7 * S_5 * R_6 * S_4 * R_0$
 $R_9 * R_8 * R_7 * S_5 * R_6 * S_4 * S_2$
 $R_9 * R_8 * R_7 * S_5 * R_6 * S_4 * S_3$
 $R_9 * S_7 * S_{23} * R_7 * S_4 * S_{19} * R_2$
 $R_9 * S_7 * S_{23} * R_7 * S_4 * S_{19} * S_{17}$
 $R_9 * S_7 * S_{23} * R_7 * S_5 * R_4 * R_2$
 $S_8 * R_9 * S_{23} * R_6 * S_4 * R_5 * S_{15}$
 $S_8 * R_9 * S_{23} * R_6 * S_4 * R_5 * R_2$
 $R_{10} * R_8 * S_5 * R_6 * S_3 * S_{18} * R_1$

$R_{10}R_8S_5R_6S_4S_{20}S_2$
 $R_{10}R_8R_7S_5R_5S_3S_{17}$
 $R_{10}R_8R_7S_5R_5S_3S_{14}$
 $R_{10}R_8R_7S_5S_4R_5S_3$
 $R_{10}R_8S_6R_7S_3R_3S_{14}$
 $R_{10}R_8S_6R_7R_5S_3S_{14}$
 $R_{10}R_8S_6R_7S_5S_3S_{14}$
 $R_{10}S_7R_8S_6S_{22}S_3R_3$
 $R_{10}S_7R_8S_6S_{22}S_4S_{16}$
 $R_{10}S_7R_8S_6S_{22}S_4R_2$
 $R_{10}S_7R_8S_6S_{22}S_4R_3$
 $R_{10}S_7R_8S_6S_{22}S_4S_3$
 $R_{10}S_7R_8S_6S_{22}S_5S_4$
 $R_{10}R_9R_8S_6R_7S_5R_1$
 $R_{10}R_9R_8S_6R_7S_5S_{14}$
 $R_{10}R_9R_8S_6R_7S_5S_3$
 $R_{10}R_9R_8S_6R_7S_5S_4$
 $R_{10}S_8S_{24}R_8S_5S_{20}R_3$
 $R_{10}S_8S_{24}R_8S_5S_{20}S_{18}$
 $R_{10}S_8S_{24}R_8S_6R_5R_3$
 $S_9R_{10}S_{24}R_7S_5R_6S_{16}$
 $S_9R_{10}S_{24}R_7S_5R_6R_3$
 $R_{11}R_9S_6R_7S_4S_{19}R_2$
 $R_{11}R_9S_6R_7S_5S_{21}S_3$
 $R_{11}R_9R_8S_6R_6S_4R_2$
 $R_{11}R_9R_8S_6R_6S_4S_{18}$
 $R_{11}R_9R_8S_6S_5R_6S_4$
 $R_{11}R_9S_7R_8S_4S_{17}R_2$
 $R_{11}R_9S_7R_8S_4R_4R_2$
 $R_{11}S_8R_9S_7S_{23}S_4R_4$
 $R_{11}S_8R_9S_7S_{23}S_5S_{17}$

$R_{11} * S_8 * R_9 * S_7 * S_{23} * S_5 * R_3$
 $R_{11} * S_8 * R_9 * S_7 * S_{23} * S_5 * R_4$
 $R_{11} * S_8 * R_9 * S_7 * S_{23} * S_5 * S_4$
 $R_{11} * S_8 * R_9 * S_7 * S_{23} * S_6 * S_5$
 $R_{11} * R_{10} * R_9 * S_7 * R_8 * S_6 * S_{15}$
 $R_{11} * R_{10} * R_9 * S_7 * R_8 * S_6 * R_2$
 $R_{11} * R_{10} * R_9 * S_7 * R_8 * S_6 * S_4$
 $R_{11} * R_{10} * R_9 * S_7 * R_8 * S_6 * S_5$
 $R_{11} * S_9 * S_{25} * R_9 * S_6 * S_{21} * R_4$
 $R_{11} * S_9 * S_{25} * R_9 * S_6 * S_{21} * S_{19}$
 $R_{11} * S_9 * S_{25} * R_9 * S_7 * R_6 * R_4$
 $S_{10} * R_{11} * S_{25} * R_8 * S_6 * R_7 * S_{17}$
 $S_{10} * R_{11} * S_{25} * R_8 * S_6 * R_7 * R_4$
 $R_{12} * R_{10} * S_7 * R_8 * S_5 * S_{20} * R_3$
 $R_{12} * R_{10} * S_7 * R_8 * S_6 * S_{22} * S_4$
 $R_{12} * R_{10} * R_9 * S_7 * R_7 * S_5 * R_3$
 $R_{12} * R_{10} * R_9 * S_7 * R_7 * S_5 * S_{19}$
 $R_{12} * R_{10} * R_9 * S_7 * S_6 * R_7 * S_5$
 $R_{12} * R_{10} * S_8 * R_9 * S_5 * S_{18} * R_3$
 $R_{12} * R_{10} * S_8 * R_9 * S_5 * R_5 * R_3$
 $R_{12} * S_9 * R_{10} * S_8 * S_{24} * S_5 * R_5$
 $R_{12} * S_9 * R_{10} * S_8 * S_{24} * S_6 * S_{18}$
 $R_{12} * S_9 * R_{10} * S_8 * S_{24} * S_6 * R_4$
 $R_{12} * S_9 * R_{10} * S_8 * S_{24} * S_6 * R_5$
 $R_{12} * S_9 * R_{10} * S_8 * S_{24} * S_6 * S_5$
 $R_{12} * S_9 * R_{10} * S_8 * S_{24} * S_7 * S_6$
 $R_{12} * R_{11} * R_{10} * S_8 * R_9 * S_7 * S_{16}$
 $R_{12} * R_{11} * R_{10} * S_8 * R_9 * S_7 * R_3$
 $R_{12} * R_{11} * R_{10} * S_8 * R_9 * S_7 * S_5$
 $R_{12} * R_{11} * R_{10} * S_8 * R_9 * S_7 * S_6$
 $R_{12} * S_{10} * S_{26} * R_{10} * S_7 * S_{22} * R_5$

$R_{12} * S_{10} * S_{26} * R_{10} * S_7 * S_{22} * S_{20}$
 $R_{12} * S_{10} * S_{26} * R_{10} * S_8 * R_7 * R_5$
 $R_{13} * R_{11} * S_8 * R_9 * S_6 * S_{21} * R_4$
 $R_{13} * R_{11} * S_8 * R_9 * S_7 * S_{23} * S_5$
 $R_{13} * R_{11} * R_{10} * S_8 * R_8 * S_6 * R_4$
 $R_{13} * R_{11} * R_{10} * S_8 * R_8 * S_6 * S_{20}$
 $R_{13} * R_{11} * R_{10} * S_8 * S_7 * R_8 * S_6$
 $R_{13} * R_{11} * S_9 * R_{10} * S_6 * S_{19} * R_4$
 $R_{13} * R_{11} * S_9 * R_{10} * S_6 * R_6 * R_4$
 $R_{13} * S_{10} * R_{11} * S_9 * S_{25} * S_6 * R_6$
 $R_{13} * S_{10} * R_{11} * S_9 * S_{25} * S_7 * S_{19}$
 $R_{13} * S_{10} * R_{11} * S_9 * S_{25} * S_7 * R_5$
 $R_{13} * S_{10} * R_{11} * S_9 * S_{25} * S_7 * R_6$
 $R_{13} * S_{10} * R_{11} * S_9 * S_{25} * S_7 * S_6$
 $R_{13} * S_{10} * R_{11} * S_9 * S_{25} * S_8 * S_7$
 $R_{13} * R_{12} * R_{11} * S_9 * R_{10} * S_8 * S_{17}$
 $R_{13} * R_{12} * R_{11} * S_9 * R_{10} * S_8 * R_4$
 $R_{13} * R_{12} * R_{11} * S_9 * R_{10} * S_8 * S_6$
 $R_{13} * R_{12} * R_{11} * S_9 * R_{10} * S_8 * S_7$
 $R_{14} * R_{12} * S_9 * R_{10} * S_7 * S_{22} * R_5$
 $R_{14} * R_{12} * S_9 * R_{10} * S_8 * S_{24} * S_6$
 $R_{14} * R_{12} * R_{11} * S_9 * R_9 * S_7 * R_5$
 $R_{14} * R_{12} * R_{11} * S_9 * R_9 * S_7 * S_{21}$
 $R_{14} * R_{12} * R_{11} * S_9 * S_8 * R_9 * S_7$
 $R_{14} * R_{12} * S_{10} * R_{11} * S_7 * S_{20} * R_5$
 $R_{14} * R_{12} * S_{10} * R_{11} * S_7 * R_7 * R_5$
 $R_{14} * R_{12} * S_{10} * S_{26} * R_{11} * S_7 * R_7$
 $R_{14} * R_{12} * S_{10} * S_{26} * R_{11} * S_8 * S_{20}$
 $R_{14} * R_{12} * S_{10} * S_{26} * R_{11} * S_8 * R_6$
 $R_{14} * R_{12} * S_{10} * S_{26} * R_{11} * S_8 * R_7$
 $R_{14} * R_{12} * S_{10} * S_{26} * R_{11} * S_8 * S_7$

$R_{14} * R_{12} * S_{10} * S_{26} * R_{11} * S_9 * S_8$
 $R_{14} * R_{13} * R_{12} * S_{10} * R_{11} * S_9 * S_{18}$
 $R_{14} * R_{13} * R_{12} * S_{10} * R_{11} * S_9 * R_5$
 $R_{14} * R_{13} * R_{12} * S_{10} * R_{11} * S_9 * S_7$
 $R_{14} * R_{13} * R_{12} * S_{10} * R_{11} * S_9 * S_8$
 $R_{15} * R_{13} * S_{10} * R_{11} * S_8 * S_{23} * R_6$
 $R_{15} * R_{13} * S_{10} * R_{11} * S_9 * S_{25} * S_7$
 $R_{15} * R_{13} * R_{12} * S_{10} * R_{10} * S_8 * R_6$
 $R_{15} * R_{13} * R_{12} * S_{10} * R_{10} * S_8 * S_{22}$
 $R_{15} * R_{13} * R_{12} * S_{10} * S_9 * R_{10} * S_8$
 $R_{15} * R_{13} * S_{27} * S_{10} * R_{11} * S_9 * S_{25}$
 $R_{15} * R_{13} * S_{27} * R_{12} * R_{11} * S_8 * R_8$
 $R_{15} * R_{13} * S_{27} * R_{12} * R_{11} * S_9 * S_{21}$
 $R_{15} * R_{13} * S_{27} * R_{12} * R_{11} * S_9 * R_7$
 $R_{15} * R_{13} * S_{27} * R_{12} * R_{11} * S_9 * R_8$
 $R_{15} * R_{13} * S_{27} * R_{12} * R_{11} * S_9 * S_8$
 $R_{15} * R_{13} * S_{27} * R_{12} * S_{10} * R_{11} * S_9$
 $R_{15} * S_{28} * R_{13} * S_{27} * R_{12} * R_{11} * S_9$
 $R_{15} * R_{14} * R_{13} * R_{12} * S_{10} * R_{11} * S_{19}$
 $R_{15} * R_{14} * R_{13} * R_{12} * S_{10} * R_{11} * R_6$
 $R_{15} * R_{14} * R_{13} * R_{12} * S_{10} * R_{11} * S_8$
 $R_{15} * R_{14} * R_{13} * R_{12} * S_{10} * R_{11} * S_9$
 $R_{15} * S_{29} * R_{14} * S_{28} * R_{13} * S_{10} * R_{10}$
 $R_{15} * S_{29} * R_{14} * S_{28} * R_{13} * R_{12} * R_9$
 $R_{15} * S_{29} * R_{14} * S_{28} * R_{13} * R_{12} * S_{10}$
 $R_{16} * R_{14} * R_{12} * S_{10} * S_{26} * R_{11} * S_8$
 $R_{16} * R_{14} * S_{28} * R_{13} * R_{12} * S_9 * R_9$
 $R_{16} * R_{14} * S_{28} * R_{13} * R_{12} * S_{10} * S_{11}$
 $R_{16} * R_{14} * S_{28} * R_{13} * R_{12} * S_{10} * S_{22}$
 $R_{16} * R_{14} * S_{28} * R_{13} * R_{12} * S_{10} * R_8$
 $R_{16} * R_{14} * S_{28} * R_{13} * R_{12} * S_{10} * R_9$

$R_{16} * R_{14} * S_{28} * R_{13} * R_{12} * S_{10} * S_9$
 $R_{16} * S_{29} * R_{14} * S_{28} * R_{13} * R_{12} * S_{10}$
 $R_{16} * R_{15} * R_{14} * R_{13} * R_{11} * S_9 * R_{10}$
 $R_{16} * R_{15} * R_{14} * R_{13} * R_{12} * R_{11} * S_{20}$
 $R_{16} * R_{15} * R_{14} * R_{13} * R_{12} * R_{11} * R_7$
 $R_{16} * R_{15} * R_{14} * R_{13} * R_{12} * R_{11} * S_9$
 $R_{16} * R_{15} * R_{14} * S_{28} * R_{13} * R_{12} * R_8$
 $R_{16} * R_{15} * R_{14} * S_{28} * R_{13} * R_{12} * S_{10}$
 $R_{16} * R_{15} * S_{29} * R_{14} * S_{28} * R_{13} * R_9$
 $R_{16} * S_{30} * R_{15} * S_{29} * R_{14} * R_{11} * S_{11}$
 $R_{16} * S_{30} * R_{15} * S_{29} * R_{14} * R_{13} * S_{11}$
 $R_{16} * S_{30} * R_{15} * S_{29} * R_{14} * R_{13} * R_{10}$
 $R_{16} * S_{30} * R_{15} * S_{29} * R_{14} * S_{28} * R_{10}$
 $R_{16} * S_{30} * R_{15} * S_{29} * R_{14} * S_{28} * R_{13}$