УДК 004.056.55

ДОСЛІДЖЕННЯ ПОТОКОВИХ АЛГОРИТМІВ ШИФРУВАННЯ – ПЕРЕМОЖЦІВ МІЖНАРОДНОГО ПРОЕКТУ ESTREAM

О.О. КУЗНЕЦОВ, Д.В. ІВАНЕНКО, Р.І. МОРДВІНОВ

Аналізуються потокові алгоритми шифрування, які визначено переможцями міжнародного проекту eSTREAM. Досліджується базова структура алгоритмів-переможців, застосовані функціональні зав'язки, шари лінійних та нелінійних перетворень тощо. Проводяться експериментальні дослідження статистичної безпеки за методикою тестування NIST Special Publication 800-22. Встановлено, що вихідні послідовності досліджуваних шифрів відповідають сучасним вимогам, розглянуті криптоперетворення можуть бути застосовані під час розробки різних варіантів побудови національного стандарту потокового шифрування України.

Ключові слова: криптографічне перетворення, потоковий шифр, статистичні тести.

ВСТУП

Міжнародний проект eSTREAM проводився у 2004 – 2008 роках Європейським Союзом та був призначений для виявлення нового потокового шифру, придатного для широкого застосування [1–7]. Його було започатковано через невдачу всіх шістьох потокових шифрів, поданих свого часу для проекту NESSIE [8, 9].

Проект eSTREAM складався з декількох фаз і його метою було знаходження алгоритмів, придатних для різних варіантів застосування. Для цього шифри в проекті eSTREAM згруповані за такими профілями:

– профіль 1 – шифри для програмного забезпечення з високими вимогами до пропускної здатності;

– профіль 2 – шифри для апаратного забезпечення з обмеженими ресурсами (пам'яті, кількості транзисторів або інших пристроїв) або споживання енергії.

На сьогоднішній день відібрані 7 шифрів, які відповідають встановленим вимогам (див. табл. 1) [1–7]. Наведені шифри не мають обмежень щодо застосування, в тому числі патентних чи будь-яких інших [1–7].

Таблиця 1

Профіль 1	Профіль 2
HC-128	Grain
Rabbit	MICKEY
Salsa20/12	Trivium
SOSEMANUK	

Метою роботи є аналіз та дослідження структури та основних перетворень потокових алгоритмів шифрування з міжнародного проекту eSTREAM, дослідження показників їхньої статистичної безпеки та обґрунтування можливості застосування певних технічних рішень під час розробки різних варіантів побудови національного стандарту потокового шифрування України.

1. ДОСЛІДЖЕННЯ ПРОГРАМНО-ОРІЄНТОВАНИХ ПОТОКОВИХ ШИФРІВ

До першого профілю міжнародного проекту eSTREAM відносяться криптоалгоритми, які спрямовано на програмну реалізацію з високими вимогами до швидкості формування псевдовипадкових послідовностей (ПВП). До цього профілю віднесено: HC-128; Rabbit; Salsa 20/12; SOSEMANUK.

Розглянемо більш докладно кожен алгоритм, проаналізуємо особливості побудови та можливості застосування.

1.1. Потоковий шифр HC-128

Потоковий шифр HC-128 є спрощеною версією HC-256 [2] для 128-бітного рівня безпеки. Це простий, безпечний, програмно-орієнтований шифр з ефективною реалізацією і може вільно використовуватися. Він складається з двох секретних таблиць, кожна з яких містить 512 32-бітових елементів. На кожному кроці виконується оновлення одного елементу таблиці з нелінійною функцією зворотного зв'язку. Всі елементи двох таблиць оновлюється кожні 1024 кроків. На кожному кроці, один 32-бітовий вихід генерується з нелінійної функції вихідної фільтрації.

Шифр призначений для формування ключового потоку довжиною до 2^{64} бітів, який виробляється з 128-бітного ключа і 128-бітного вектора ініціалізації. В ході реалізації HC-128 використовуються операції з табл. 2.

Дві таблиці, які використовуються в HC-128, позначаються як P і Q. Ключ і вектор ініціалізації позначаються як K і IV. Ключовий потік, який генерується шифром, позначається як s.

Шифр HC-128 використовує шість функцій (так само, як і шифр HC-256). Функції $f_1(x)$ і $f_2(x)$ збігаються з функціями $\sigma_0^{\{256\}}(x)$ і $\sigma_1^{\{256\}}(x)$, які використовуються в алгоритмі ґешування SHA-256. Для функції $h_1(x)$ як S-блок використовується таблиця Q. Для функції $h_2(x)$ як S-блок використовується таблиця P. Функції мають вигляд:

$$f_1(x) = (x >>> 7) \oplus (x >>> 18) \oplus (x >> 3);$$

$$f_2(x) = (x >>> 17) \oplus (x >>> 19) \oplus (x >> 10);$$

$$g_1(x, y, z) = ((x >>> 10) \oplus (z >>> 23)) + (y >>> 8);$$

$$g_2(x, y, z) = ((x <<< 10) \oplus (z <<< 23)) + (y <<< 8);$$

$$h_1(x) = Q[x_0] + Q[256 + x_2];$$

$$h_2(x) = P[x_0] + P[256 + x_2],$$

де $x = x_3 \| x_2 \| x_1 \| x_0$, x – це 32-бітне слово, x_0, x_1, x_2, x_3 – чотири байти.

Байти x_3 і x_0 позначаються як найбільш значущий і молодший байт x, відповідно.

Таблиця 2

	Зміст позначення
+	додавання за 2^{32}
-	віднімання за модулем 512
$\oplus$	побітне виключне АБО (XOR)
$\|\|$	конкатенація
>>	зсув праворуч на вказане число бітів
<<	зсув ліворуч на вказане число бітів
>>>	циклічний зсув праворуч на вказане число бітів
<<<	циклічний зсув ліворуч на вказане число бітів
P	Таблиця з 512 32-бітними елементами. Кожен елемент позначається як $P[i]$, $0 \le i \le 511$
Q	Таблиця з 512 32-бітними елементами. Кожен елемент позначається як $Q[i]$, $0 \le i \le 511$
K	128-бітний ключ потокового шифру HC-128
IV	128-бітний вектор ініціалізації
s	Ключовий потік. На кожному кроці формується 32-бітний вихід s_i, $s = s_0 \| s_1 \| s_2 \| ...$

Алгоритм ініціалізації. Процес ініціалізації складається з розширення ключа і вектора ініціалізації в P і Q (за аналогією з установкою повідомлень в SHA-256) і запуску шифрування 1024 разів без генерації вихідного результату (виходи використовуються для оновлення P і Q):

1. Нехай $K = K_0 \,||\, K_1 \,||\, K_2 \,||\, K_3$ і $IV = IV_0 \,||\, IV_1 \,||\, IV_2 \,||\, IV_3$, де кожен K_i і IV_i позначає 32-розрядне число. Нехай $K_{i+4} = K_i$, і $IV_{i+4} = IV_i$ для $0 \le i < 4$. Ключ і вектор ініціалізації розширюються в масив W_i $(0 \le i \le 1279)$ таким чином:

$$W_i = \begin{cases} K_i, & 0 \le i \le 7; \\ IV_{i-8}, & 8 \le i \le 15; \\ f_2(W_{i-2}) + W_{i-7} + f_1(W_{i-15}) + W_{i-16} + i, & \\ \qquad 16 \le i \le 1279. & \end{cases}$$

2. Оновлення таблиці P і Q з масиву W.

$P[i] = W_{i+256}$ для $0 \le i \le 511$;

$Q[i] = W_{i+768}$ для $0 \le i \le 511$.

3. Виконання 1024 кроків шифрування і використання виходів для заміни елементів таблиці таким чином:

Для всіх від i = 0 до 511:

$P[i] = (P[i] + g1(P[i-3],$

$P[i-10], P[i-511])) \oplus h1(P[i-12]);$

Для всіх від $i = 0$ до 511:

$Q[i] = (Q[i] + g2(Q[i-3],$

$Q[i-10], Q[i-511])) \oplus h2(Q[i-12]).$

Після виконання наведених кроків процес ініціалізації завершується і шифр готовий генерувати ключовий потік.

Алгоритм генерації ключового потоку. На кожному кроці оновлюється один з елементів таблиці і генерується один 32-бітовий вихід. Кожен S-блок використовується для генерації тільки 512 виходів, то він оновлюється протягом наступних 512 кроків.

Головною особливістю в структурі потокового шифру HC-128 (та його більш потужної версії HC-256) є застосування логічних функцій $f_1(x)$ і $f_2(x)$ та процесу ініціалізації, які запозичені з алгоритму гешування SHA-256. Це дає можливість застосувати вже досліджені та добре перевірені часом обчислювально ефективні елементи криптоперетворення та досягти певних показників компактності реалізації та можливості розпаралелювання. Як результат, потоковий шифр HC-128 є дуже зручним для реалізації на сучасних мікропроцесорах, причому залежність між окремими операціями зведена до мінімуму: три послідовні кроки алгоритму можуть бути обчислені паралельно. Отримана можливість розпаралелювання призводить до високих показників ефективності на сучасних процесорах [2].

1.2. Потоковий шифр Rabbit

Алгоритм Rabbit – високошвидкісний потоковий шифр, який вперше був представлений в лютому 2003 року на 10-му симпозіумі FSE. У травні 2005, цей шифр був відправлений на конкурс eStream [4]. Розробниками алгоритму Rabbit є Martin Boesgaard, Mette Vesterager, Thomas Pedersen, Jesper Christiansen і Ove Scavenius.

Основним компонентом шифру є генератор бітового потоку, який формує 128 бітів за ітерацію. Перевага шифру Rabbit полягає в ретельному перемішуванні його внутрішніх станів між двома послідовними ітераціями. Функція перемішування повністю заснована на арифметичних операціях, доступних на сучасних процесорах, тобто S-блоки підстановок і пошукові таблиці не потрібні для реалізації шифру.

Алгоритм Rabbit можна стисло описати наступним чином. Як вхідні дані алгоритм приймає 128-бітний секретний ключ і 64-бітний вектор ініціалізації IV (за бажанням) і формує для кожної ітерації вихідний блок 128 псевдовипадкових бітів з комбінації бітів внутрішнього стану. Шифрування / розшифрування виконується операцією XOR псевдовипадкових даних з відкритим / зашифрованим текстом. При викладенні специфікації потокового шифру Rabbit використовують позначення з табл. 4 [4].

Таблиця 4

	Зміст позначення
$\oplus$	Операція логічного виключного АБО (XOR)
&	Операція логічного І (AND)
>>	Оператор зсуву праворуч на вказане число бітів
<<	Оператор зсуву ліворуч на вказане число бітів
>>>	Оператор циклічного зсуву праворуч на вказане число бітів
<<<	Оператор циклічного зсуву ліворуч на вказане число бітів
◊	Конкатенація двох бітових послідовностей
$A^{[g..h]}$	Позначення послідовності бітів з g по h змінної A

Внутрішній стан потокового шифру складається з 513 бітів. З них 512 бітів розділені між вісьмома 32-бітовими змінними стану $x_{j,i}$ і вісьмома 32-бітовими змінними лічильника $c_{j,i}$. В цих позначеннях $x_{j,i}$ є змінною стану j-ї підсистеми на i-й ітерації, а $c_{j,i}$ є змінною лічильника j-ї підсистеми на i-й ітерації, відповідно. Застосовується також один лічильник бітів перенесення, що по-

значається як $\phi_{7,i}$ і який має зберігатися між ітераціями. Цей лічильник бітів перенесення встановлюється на нуль. Вісім змінних стану і вісім змінних лічильників встановлюються з ключа при ініціалізації.

Схема установки ключа. Алгоритм шифрування Rabbit ініціюється розширенням 128-бітного ключа в обидві складові: вісім змінних стану і вісім змінних лічильників. Існує взаємно однозначна відповідність між ключем і початковим станом цих змінних.

Установку ключа можна розділити на три етапи: розширення ключа, система ітерацій, модифікація лічильника.

Етап розширення ключа гарантує взаємно однозначну відповідність між ключем стану і ключем лічильника, який запобігає надлишку ключів. Він також розподіляє біти ключа оптимальним чином для всіх ітерацій.

На цьому етапі ключ поточного шифрування, який позначається як $K^{[127...0]}$, ділиться на вісім частин, т.з. субключів:

$$k_0 = K^{[15..0]},\ k_1 = K^{[31..16]},\ \ldots,\ k_7 = K^{[127..112]}.$$

Система ітерацій гарантує, що після однієї ітерації функції наступного стану кожен біт ключа вплине на всі вісім змінних стану. Це також гарантує, що після другої ітерації функції наступного стану всі біти ключа вплинуть на всі біти стану з імовірністю 0,5.

Змінні стану і лічильників зініціюються з цих субключів таким чином:

$$x_{j,0} = \begin{cases} k_{(j+1 \bmod 8)} \lozenge k_j, & \text{для всіх парних } j; \\ k_{(j+5 \bmod 8)} \lozenge k_{(j+4 \bmod 8)}, & \text{для всіх непарних } j; \end{cases}$$

$$A_{j,0} = \begin{cases} k_{(j+4 \bmod 8)} \lozenge k_{(j+5 \bmod 8)}, & \text{для всіх парних } j; \\ k_j \lozenge k_{(j+1 \bmod 8)}, & \text{для всіх непарних } j. \end{cases}$$

Схема установки ключа повторюється чотири рази, відповідно до функції наступного стану (визначеної далі). Це спрямовано на зменшення кореляції між бітами ключа і бітами змінних внутрішнього стану.

Модифікація лічильника. Навіть, якщо лічильники будуть відомі зловмисникові, модифікація лічильника значно ускладнює відновлення ключа шляхом інвертування лічильника системи, оскільки в цьому випадку будуть потрібні відомості щодо змінних стану. Крім того, модифікація порушує взаємно однозначне співвідношення між ключем і лічильником.

Змінні лічильника модифікуються відповідно до такого правила:

$$c_{j,4} = c_{j,4} \oplus x_{(j+4 \bmod 8),4},$$

яке виконується для всіх j.

Схема установки вектора ініціалізації. Позначимо внутрішній стан після схеми установки ключа як майстер стан і нехай копія цього майстер стану буте модифікована згідно зі схемою вектора ініціалізації *IV*. Схема установки *IV* функціонує шляхом зміни стану лічильника як функція від *IV*. Це робиться шляхом операції XOR над 64-бітним вектором ініціалізації *IV* на всіх 256 бітах стану лічильника. Ці 64 біти вектора ініціалізації *IV* позначаються як $IV^{[63..0]}$. Лічильники модифікуються таким чином:

$$\begin{aligned} c_{0,4} &= c_{0,4} \oplus IV[31..0], \\ c_{1,4} &= c_{1,4} \oplus (IV[63..48] \lozenge IV[31..16]), \\ c_{2,4} &= c_{2,4} \oplus IV[63..32], \\ c_{3,4} &= c_{3,4} \oplus (IV[47..32] \lozenge IV[15..0]), \\ c_{4,4} &= c_{4,4} \oplus IV[31..0], \\ c_{5,4} &= c_{5,4} \oplus (IV[63..48] \lozenge IV[31..16]), \\ c_{6,4} &= c_{6,4} \oplus IV[63..32], \\ c_{7,4} &= c_{7,4} \oplus (IV[47..32] \lozenge IV[15..0]). \end{aligned}$$

Наведена схема модифікації повторюється чотири рази, щоб усі біти стану стали нелінійно залежними від усіх бітів вектора ініціалізації *IV*. Модифікація змінних лічильника за вектором ініціалізації *IV* гарантує, що застосування всіх 2^{64} різних векторів ініціалізації *IV* призведе до формування унікальних ключових потоків.

Функція наступного стану. Ядром алгоритму є ітеративна схема, яка визначена за допомогою таких рівнянь:

$$\begin{aligned} x_{0,i+1} &= g_{0,i} + (g_{7,}{}_i <<< 16) + (g_{6,i} <<< 16); \\ x_{1,i+1} &= g_{1,i} + (g_{0,i} <<< 8) + g_{7,i}; \\ x_{2,i+1} &= g_{2,i} + (g_{1,\,i} <<< 16) + (g_{0,i} <<< 16); \\ x_{3,i+1} &= g_{3,i} + (g_{2,\,i} <<< 8) + g_{1,i}; \\ x_{4,i+1} &= g_{4,i} + (g_{3,\,i} <<< 16) + (g_{2,\,i} <<< 16) \\ x_{5,i+1} &= g_{5,i} + (g_{4,i} <<< 8) + g_{3,i}; \\ x_{6,i+1} &= g_{6,i} + (g_{5,\,i} <<< 16) + (g_{4,i} <<< 16); \\ x_{7,i+1} &= g_{7,i} + (g_{6,\,i} <<< 8) + g_{5,i}, \end{aligned} \qquad (5)$$

де

$$g_{j,i} = ((x_{j,i} + c_{j,i+1})^2 \oplus ((x_{j,i} + c_{j,i} + 1)^2 >> 32)) \bmod 2^{32}$$

і всі додавання виконуються за модулем 2^{32}.

Наведені аналітичні вирази в поєднанні між собою можна в графічному вигляді подати на рис. 1. Перед ітерацією лічильники збільшуються, як це описано далі.

Система лічильників. Динаміка зміни лічильників визначається таким чином:

$$\begin{aligned} c_{0,i+1} &= c_{0,i} + a_0 + \phi_{7,i} \bmod 2^{32}; \\ c_{1,i+1} &= c_{1,i} + a_1 + \phi_{0,i+1} \bmod 2^{32}; \\ c_{2,i+1} &= c_{2,i} + a_2 + \phi_{1,i+1} \bmod 2^{32}; \\ c_{3,i+1} &= c_{3,i} + a_3 + \phi_{2,i+1} \bmod 2^{32}; \\ c_{4,i+1} &= c_{4,i} + a_4 + \phi_{3,i+1} \bmod 2^{32}; \\ c_{5,i+1} &= c_{5,i} + a_5 + \phi_{4,i+1} \bmod 2^{32}; \\ c_{6,i+1} &= c_{6,i} + a_6 + \phi_{5,i+1} \bmod 2^{32}; \\ c_{7,i+1} &= c_{7,i} + a_7 + \phi_{6,i+1} \bmod 2^{32}, \end{aligned}$$

де лічильник біту перенесення $\phi_{j,i+1}$ задається як

$$\phi_{j,i+1} = \begin{cases} 1, & c_{0,i} + a_0 + \phi_{7,i} \ge 2^{32} \wedge j = 0; \\ 1, & c_{j,i} + a_j + \phi_{j-1,i+1} \ge 2^{32} \wedge j > 0; \\ 0, & \textit{otherwise}. \end{cases}$$

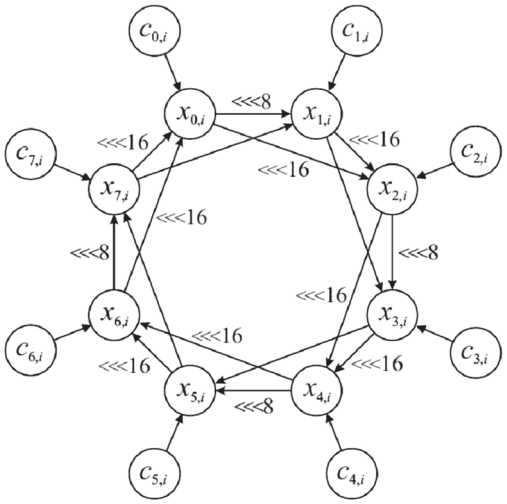

Рис. 1

Постійні a_j визначаються як:

$a_0 = 0x4D34D34D$; $a_1 = 0xD34D34D3$;

$a_2 = 0x34D34D34$; $a_3 = 0x4D34D34D$;

$a_4 = 0xD34D34D3$; $a_5 = 0x34D34D34$;

$a_6 = 0x4D34D34D$; $a_7 = 0xD34D34D3$.

Схема виходу. Після кожної ітерації вихідний ключовий потік формується таким чином:

$$s_i^{[15..0]} = x_{0,i}^{[15..0]} \oplus x_{5,i}^{[31..16]};\ s_i^{[31..16]} = x_{0,i}^{[31..16]} \oplus x_{3,i}^{[15..0]};$$

$$s_i^{[47..32]} = x_{2,i}^{[15..0]} \oplus x_{7,i}^{[31..16]};\ s_i^{[63..48]} = x_{2,i}^{[31..16]} \oplus x_{5,i}^{[15..0]};$$

$$s_i^{[78..64]} = x_{4,i}^{[15..0]} \oplus x_{1,i}^{[31..16]};\ s_i^{[95..80]} = x_{4,i}^{[31..16]} \oplus x_{7,i}^{[15..0]};$$

$$s_i^{[111..96]} = x_{6,i}^{[15..0]} \oplus x_{3,i}^{[31..16]};\ s_i^{[127..112]} = x_{6,i}^{[31..16]} \oplus x_{1,i}^{[15..0]},$$

де s_i є 128-бітний блок ключового потоку на i-й ітерації.

Схема шифрування/розшифрування. Сформовані біти ключового потоку додаються (операцією XOR) до відкритого/зашифрованого тексту для шифрування/розшифрування:

$$c_i = p_i \oplus s_i,\ p_i = c_i \oplus s_i,$$

де c_i і p_i позначають i-й 128-бітний блок зашифрованого тексту і блок відкритого тексту, відповідно.

Головною особливістю в структурі потокового шифру Rabbit є застосування логічних функцій (5), які використовують модульне додавання та піднесення до квадрата (за модулем 2^{32}). Це дозволяє значно спростити схему обчислення значень вихідного ключового потоку, акцентувати приріст швидкості генерації гами.

Таким чином, алгоритм Rabbit був розроблений для забезпечення високої швидкодії, зокрема для того, щоб бути швидшим, ніж зазвичай використовувані шифри із розміром ключа 128 біт для шифрування до 2^{64} блоків тексту. Для зловмисника, який не знає ключ, можливість розрізнити 2^{64} блоків шифру від виходу генератора справжніх випадкових біт має потребувати не менше дій, ніж потрібно було б для перебору 2^{128} ключів [4].

1.3. Потоковий шифр Salsa 20

Salsa 20 є потоковим шифром, який був розроблений Daniel J. Bernstein [5]. Він має дві модифікації Salsa 20/8 і Salsa 20/12, 8-м і 12-ть раундів за шифрування, відповідно, замість 20 оригінальних в Salsa 20. Вихідний алгоритм Salsa 20 був зроблений з великим запасом стійкості, тоді як Salsa 20/8 показує гарні результати у швидкодії, випереджаючи у більшості випадків багато інших шифрів, у тому числі AES і RC4 [5]. Алгоритм використовує геш-функцію з 20 циклами. Основні її перетворення нагадують алгоритм AES. При викладенні специфікації потокового шифру Salsa 20 використовують позначення, що наведено в табл. 5 [5].

Таблиця 5

	Зміст позначення
$\oplus$	Операція побітного додавання за модулем 2 (XOR)
$+$	Операція додавання 32-х бітних чисел за модулем 2^{32}
$<<<$	Оператор циклічного зсуву ліворуч на вказане число бітів

Функція ***quarterround***. Основним функціональним блоком системи шифрування є перетворення *quarterround*(y) над 4-ма словами. З цього перетворення будуються більш загальні та складні перетворення.

Сутність перетворення *quarterround*(y) полягає в тому, що для кожного слова складають два попередніх, отриману суму циклічно зсувають на певну кількість біт і побітово підсумовують результат з обраним словом. Наступні операції проводяться з новими значеннями слів.

Якщо y – це послідовність 4-х слів, тобто $y = (y_0, y_1, y_2, y_3)$, тоді обчислення функції *quarterround*(y) $= (z_0, z_1, z_2, z_3)$ полягає в тому, що:

$$z_1 = y_1 \oplus ((y_0 + y_3) <<< 7),$$
$$z_2 = y_2 \oplus ((z_1 + y_0) <<< 9),$$
$$z_3 = y_3 \oplus ((z_2 + z_1) <<< 13),$$
$$z_0 = y_0 \oplus ((z_3 + z_2) <<< 18).$$

Схема обчислення функції *quarterround*(y) $= (z_0, z_1, z_2, z_3)$ наведена на рис. 2. Можна розглядати цю функцію як перетворення слів y_1, y_2, y_3 і y_4. Кожне з перетворень обернене, як і функція в цілому.

Функція ***rowround***. На вхід перетворення поступає 16 слів у вигляді матриці 4×4. Кожен рядок цієї «матриці» обробляється функцією *quarterround*(y). Слова з рядка беруться по порядку, починаючи з i-го для i-го рядка, де $i = \{0, 1, 2, 3\}$.

Якщо $y = (y_0, y_1, y_2, ..., y_{15})$ – послідовність з 16 слів, тоді обчислення функції *rowround*(y) $= (z_0, z_1, z_2, ..., z_{15})$ полягає в наступному:

$$(z_0, z_1, z_2, z_3) = quarterround(y_0, y_1, y_2, y_3),$$
$$(z_5, z_6, z_7, z_4) = quarterround(y_5, y_6, y_7, y_4),$$
$$(z_{10}, z_{11}, z_8, z_9) = quarterround(y_{10}, y_{11}, y_8, y_9),$$
$$(z_{15}, z_{12}, z_{13}, z_{14}) = quarterround(y_{15}, y_{12}, y_{13}, y_{14}).$$

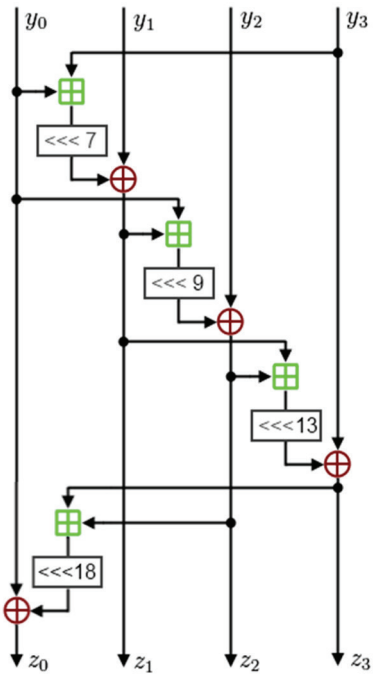

Рис. 2

Функція ***columnround***. Перетворення аналогічне функції *quarterround(y)*, аргументом якої є значення стовпців (колонок) матриці, починаючи з *j*-го для *j*-го стовпця, де $j = \{0,1,2,3\}$.

Функція $columnround(y) = (z)$ також оперує послідовністю 16-и слів так, що

$$(z_0, z_4, z_8, z_{12}) = quarterround(y_0, y_4, y_8, y_{12}),$$
$$(z_5, z_9, z_{13}, z_1) = quarterround(y_5, y_9, y_{13}, y_1),$$
$$(z_{10}, z_{14}, z_2, z_6) = quarterround(y_{10}, y_{14}, y_2, y_6),$$
$$(z_{15}, z_3, z_7, z_{11}) = quarterround(y_{15}, y_3, y_7, y_{11}).$$

Еквівалентна формула:

$$(z_0, z_4, z_8, z_{12}, z_1, z_5, z_9, z_{13}, z_2, z_6, z_{10}, z_{14}, z_3, z_7, z_{11}, z_{15}) = rowround(y_0, y_4, y_8, y_{12}, y_1, y_5, y_9, y_{13}, y_2, y_6, y_{10}, y_{14}, y_3, y_7, y_{11}, y_{15}).$$

Функція ***doubleround***. Функція *doubleround(y)* є послідовним застосуванням функцій *columnround* а потім *rowround*, тобто:

$$doubleround(y) = rowround(columnround(y)).$$

Функція ***littleendian***. Salsa 20 використовує запис слова, яке починається з молодшого байта. Для цього введено окреме перетворення *littleendian(b)*.

Нехай $b = (b_0, b_1, b_2, b_3)$ – 4-байтова послідовність, тоді функція $littleendian(b)$ повертає слово, таке що $littleendian(b) = b_0 + 2^8 b_1 + 2^{16} b_2 + 2^{24} b_3$. Слід відмітити, що ця функція є оберненою.

Геш-функція ***Salsa20(x)***. На вхід функції подається 64-байтова послідовність і повертає 64-бійтову послідовність. Обчислення функції $Salsa20(x)$ починається з вектора

$$x = (x[0], x[1], ..., x[63]),$$

де

$$x_0 = littleendian(x[0], x[1], x[2], x[3]),$$
$$x_1 = littleendian(x[4], x[5], x[6], x[7]),$$
$$x_2 = littleendian(x[8], x[9], x[10], x[11]),$$
$$...$$
$$x_{15} = littleendian(x[60], x[61], x[62], x[63]).$$

Тут застосовано такі позначення: $x[i]$ – байти вхідної послідовності x, x_j – слова, що використовуються в описаних вище функціях.

Визначимо $(z_0, z_1, ..., z_{15}) = doubleround^{10}(x_0, x_1, ..., x_{15})$, тоді значення функції $Salsa20(x)$ обчислюється як:

$$littleendian^{-1}(z_0 + x_0);$$
$$littleendian^{-1}(z_1 + x_1);$$
$$littleendian^{-1}(z_2 + x_2);$$
$$...$$
$$littleendian^{-1}(z_{15} + x_{15}).$$

Функція поширення ***Salsa20$_k$(n)***. На вхід функції $Salsa20_k(n)$ поступає 32-байтова або 16-байтова послідовність k і 16-байтова послідовність n та повертає 64-байтову послідовність. Для цього використовуються константи:

$$\sigma_0 = (101, 120, 112, 97),$$
$$\sigma_1 = (110, 100, 32, 51),$$
$$\sigma_2 = (50, 45, 98, 121),$$
$$\sigma_3 = (116, 101, 32, 107).$$

Якщо k_0, k_1, n є 16-байтовими послідовностями, тоді

$$Salsa20_{k0,k1}(n) = Salsa20(\sigma_0, k_0, \sigma_1, n, \sigma_2, k_1, \sigma_3).$$

Позначимо

$$\tau_0 = (101, 120, 112, 97),$$
$$\tau_1 = (110, 100, 32, 49),$$
$$\tau_2 = (54, 45, 98, 121),$$
$$\tau_3 = (116, 101, 32, 107).$$

Якщо k, n є 16-байтовими послідовностями, тоді

$$Salsa20_k(n) = Salsa20(\tau_0, k, \tau_1, n, \tau_2, k).$$

Функція шифрування ***Salsa20***. Шифртекстом l-байтової послідовності m для $l \in \{0,1, ... 2^{70}\}$ буде

$$Salsa20_k(v) \oplus m,$$

де v – унікальний 8-байтовий номер (*nonce*).

Шифртекст має розмір l байт, так само як і вихідний текст.

$Salsa20_k(v)$ – ключовий потік 2^{70} байт:

$$Salsa20_k(v, \underline{0}), Salsa20_k(v, \underline{1}), Salsa20_k(v, \underline{2}), ... , Salsa20_k(v, \underline{2^{64} - 1}),$$

де $\underline{i}$ – унікальна послідовність з 8 байт $(i_0, i_1, ..., i_7)$ така, що

$$i = i_0 + 2^8 i_1 + ... + 2^{56} i_7.$$

Відповідно

$$Salsa20_k(v) \oplus (m[0], m[1], .., m[l\text{-}1]) = (c[0], c[1], ..., c[l\text{-}1]),$$

де $c[i] = m[i] \oplus Salsa20_k(v, \lfloor i/64 \rfloor)[i \bmod 64]$.

Таким чином, алгоритм *Salsa*20 практично не має накладних обчислень. Багато шифросистем розраховують попередні обчислення, результати яких мають зберігатися в кеші першого рівня (L1) процесора. Оскільки вони залежать від ключа у випадку його оновлення всі обчислення потрібно здійснити заново. У *Salsa*20 ж досить просто завантажити ключ у пам'ять.

1.4. Потоковий шифр SOSEMANUK

Sosemanuk – це новий синхронний програмно-орієнтований потоковий шифр, який відповідає першому профілю міжнародного конкурсу eECRYPT [6]. Його довжина ключа може бути обрана між 128 і 256 бітами. Шифр працює з 128 бітовим початковим значенням, при цьому, як стверджується розробниками алгоритму [6], будь-яка довжина ключа досягає 128-бітного захисту.

Алгоритм Sosemanuk використовує деякі основні принципи потокового шифру Snow 2.0 [10] і деякі перетворення, отримані з блокового симетричного шифру (БСШ) SERPENT [11]. Sosemanuk спрямований на поліпшення Snow 2.0 як в сенсі безпеки, так і в сенсі ефективності реалізації. Зокрема, Sosemanuk використовує швидку процедуру установки вектора ініціалізації IV. Він також вимагає зменшеної кількості статистичних даних, що дає більш високу продуктивність на декількох архітектурах.

SERPENT і його похідні. SERPENT [11] – є БСШ, який відповідає умовам конкурсу AES на визначення стандарту БСШ США. Алгоритм SERPENT працює над блоками з 128 біт, які розділені на чотири 32-бітові блоки кожен. Пронумеруємо вхідні і вихідні четвірки алгоритму SERPENT від 0 до 3 і запишемо їх в порядку: (Y_3, Y_2, Y_1, Y_0). Y_0 є мінімальним значущим словом і містить мінімальні значущі біти з 32-х 4-бітових слів, що входять в S-блок шифру SERPENT. У той час як вихід алгоритму SERPENT записується в 16 байт, значення Y_i записуються від молодшого до старшого (перший – найменш значимий байт), Y_0 виводиться першим, далі Y_1, і т. д.

З алгоритму SERPENT визначаються два примітива, це Serpent1 і Serpent24 [11].

Serpent1. Раунди алгоритму БСШ SERPENT складаються в такому порядку:

– побітова операція XOR;

– застосування S-блоку (що виражається як набір бітових перестановок між чотирма використовуваними 32-бітними словами);

– лінійна бієктивна трансформація (яка складається з декількох операцій XOR, зсувів і поворотів).

Таким чином, примітив Serpent1 – це один раунд алгоритму БСШ SERPENT без додавання ключа і лінійної трансформації. SERPENT використовує 8 унікальних S-блоків, пронумерованих, відповідно, з S_0 до S_7 на 4-бітових словах. Serpent1 визначається як застосування S_2, в режимі «bitslice», він приймає чотири 32-бітних слова на вхід і надає чотири 32-бітних слова на вихід.

Serpent24. Serpent24 – це алгоритм SERPENT, зменшений до 24 раундів замість 32 раундів повної версії SERPENT. Serpent24 еквівалентний першим 24 раундів SERPENT, де останній, 24-й раунд є повним і включає повний раунд з лінійною трансформації і операцію XOR з 25 підключами. Іншими словами, 24-й раунд Serpent24 є еквівалентним тридцять другому раунду алгоритму SERPENT, за винятком того, що він містить лінійну трансформацію і того, що тут використовуються 24-й та 25-й підключі (32-й і 33-й підключі в алгоритмі SERPENT). Таким чином, рівнянням останнього раунду є:

$$R23\,(X) = L\,(S23\,(X \oplus K23)) \oplus K24.$$

Примітив Serpent24 використовує тільки 25 128-бітних підключів, які є першими 25 підключами алгоритму SERPENT. Serpent24 в Sosemanuk використовується на етапі ініціалізації, тільки в режимі зашифрування. Розшифрування не використовується.

Регістр зсуву з лінійним зворотним зв'язком (РЗЛЗЗ). Для великої частини потокових шифрів внутрішній стан зберігається та оновлюється в РЗЛЗЗ. В шифрі Sosemanuk застосовується РЗЛЗЗ, який містить 10 елементів кінцевого поля $GF(2^{32})$.

Базове кінцеве поле. Елементи поля $GF(2^{32})$ представлені так само, як і в специфікації шифру SNOW 2.0. Наведемо коротко тут ці позначення.

Нехай $GF(2)$ позначає кінцеве поле з двома елементами. Нехай β є коренем примітивного полінома:

$$Q(X) = X^8 + X^7 + X^5 + X^3 + 1$$

в кільці поліномів $GF(2)[X]$. Визначимо поле $GF(2^8)$ як кільце $GF(2)[X] / Q(X)$. Кожен елемент в полі $GF(2^8)$ можемо визначити, використовуючи базис (β^7, β^6, ... β, 1). Оскільки обраний поліном примітивний, тоді β є мультиплікативним породжувальним елементом поля $GF(2^8)$: кожен ненульовий елемент з $GF(2^8)$ дорівнює β^k для деякого невід'ємного цілого k ($0 \le k \le 254$). Будь-який елемент з $GF(2^8)$ ідентифікується з 8-бітовим числом наступною бієктивною функцією:

$$\phi : GF(2^8) \to \{0,1,...,255\},\ x = \sum_{i=0}^{7} x_i \beta^i \mapsto \sum_{i=0}^{7} x_i 2^i\ ,$$

де кожен x_i є або 0 або 1. Наприклад, β^{23} представляється цілим числом $\phi\,(\beta^{23}) = 0xE1$ (в шістнадцятковій системі). Отже, додавання двох елементів в $GF(2^8)$ відповідає бітовій операції XOR між відповідними цілочисельними уявленнями. Множення на β рівнозначно лівому зсуву на один біт цілочисельного уявлення. Він виконується за операцією XOR з фіксованою маскою, якщо найстарший біт, який був видалений за допомогою зсуву, дорівнює 1.

Нехай α є коренем примітивного полінома:

$$P(X) = X^4 + \beta^{23} X^3 + \beta^{245} X^2 + \beta^{48} X + \beta^{239}$$

з $GF(2^8)[X]$. Поле $GF(2^{32})$ далі визначається як кільце $GF(2^8)[X]\ /\ P(X)$, тобто, його елементи представляються в базисі (α^3, α^2, α, 1). Будь-який елемент з $GF(2^{32})$ ідентифікується з 32-бітовим цілим числом такою бієктивною функцією:

$$\psi : GF(2^{32}) \to \{0,1,...,2^{32}-1\},$$

$$x = \sum_{i=0}^{3} y_i \alpha^i \mapsto \sum_{i=0}^{3} \phi(y_i) 2^{8i}.$$

Таким чином, додавання двох елементів в $GF(2^{32})$ відповідає побітовій операції XOR між їх цілочисельними уявленнями. Надалі ця операція буде позначена $\oplus$. Sosemanuk також використовує операції множення і ділення елементів в $GF(2^{32})$ на α. Множення $z \in GF(2^{32})$ на α відповідає лівому зсуву на 8 біт $\psi(z)$, яка слідує за операцією XOR з 32-бітною маскою, яка залежить тільки від найбільш значущих байт $\psi(z)$.

Визначення РЗЛЗЗ. РЗЛЗЗ працює з елементами $GF(2^{32})$. Початковий стан в момент часу $t = 0$ відповідає десяти 32-бітовим значенням $s_1 - s_{10}$. На кожному етапі, нове значення обчислюється за допомогою такої рекурсії:

$$s_{t+10} = s_{t+9} \oplus \alpha^{-1} s_{t+3} \oplus \alpha_{st}, \ \forall\, t \geq 1$$

і регістр зсувається (див. рис. 3 для ілюстрації роботи РЗЛЗЗ).

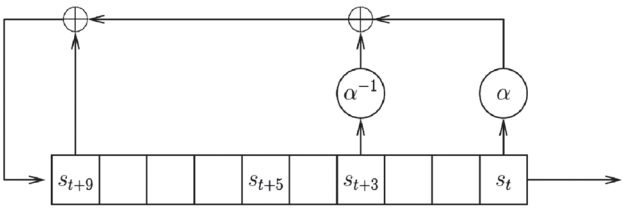

Рис. 3

Для РЗЛЗЗ, схема якого наведена на рис. 3, застосовується такий поліномом зворотного зв'язку:

$$\varpi(X) = \alpha X^{10} + \alpha^{-1} X^7 + X + 1 \in GF(2^{32})[X].$$

Оскільки РЗЛЗЗ є несингулярним, і оскільки ϖ є простим поліномом, то послідовність 32-бітових слів (s_t) $t \geq 1$ є періодичною і має максимальний період, що дорівнює 2^{320}-1.

Finite State Machine (кінцевий автомат). Кінцевий автомат (FSM) є компонентом з 64-ма бітами пам'яті, який відповідає двом 32-бітовим регістрам: $R1$ і $R2$. На кожному етапі FSM приймає на вхід кілька слів зі стану РЗЛЗЗ; він оновлює біти пам'яті і виробляє 32-біта на вихід. FSM оперує над станами РЗЛЗЗ в момент часу $t \geq 1$, як показано нижче:

$$\text{FSM}t\text{: } (R1_{t-1}, R2_{t-1}, s_{t+1}, s_{t+8}, s_{t+9})\ 1 \rightarrow (R1_t, R2_t, f_t),$$

де

$$R1_t = (R2_{t-1} + \text{mux}\,(\text{lsb}\,(R1_{t-1}),$$
$$s_{t+1}, s_{t+1} \oplus s_{t+8})) \bmod 2^{32},$$
$$R2_t = \text{Trans}\,(R1_{t-1}),$$
$$f_t = (s_{t+9} + R1_t \bmod 2^{32}) \oplus R2_t,$$

де $\text{lsb}(x)$ – найменший значущий біт x, $\text{mux}\,(c, x, y)$ дорівнює x, якщо $c = 0$, або y, якщо $c = 1$.

Внутрішня перехідна функція *Trans* на $GF(2^{32})$ визначається

$$Trans\,(z) = (M \times z \bmod 2^{32}) <<< 7,$$

де M є постійним значенням 0x54655307 і $<<<$ означає бітовий зсув 32-бітного значення (в даному випадку, на 7 біт).

Вихідне перетворення. Виходи FSM згруповані по чотири і Serpent1 застосовується до кожної групи. Далі результат комбінується операцією XOR з відповідними значеннями стану РЗЛЗЗ, для отримання на виході значень z_t:

$$(z_{t+3}, z_{t+2}, z_{t+1}, zt) =$$
$$= \text{Serpent1}\,(f_{t+3}, f_{t+2}, f_{t+1}, f_t) \oplus (s_{t+3}, s_{t+2}, s_{t+1}, s_t).$$

Чотири послідовних раунди Sosemanuk схематично зображені на рис. 4.

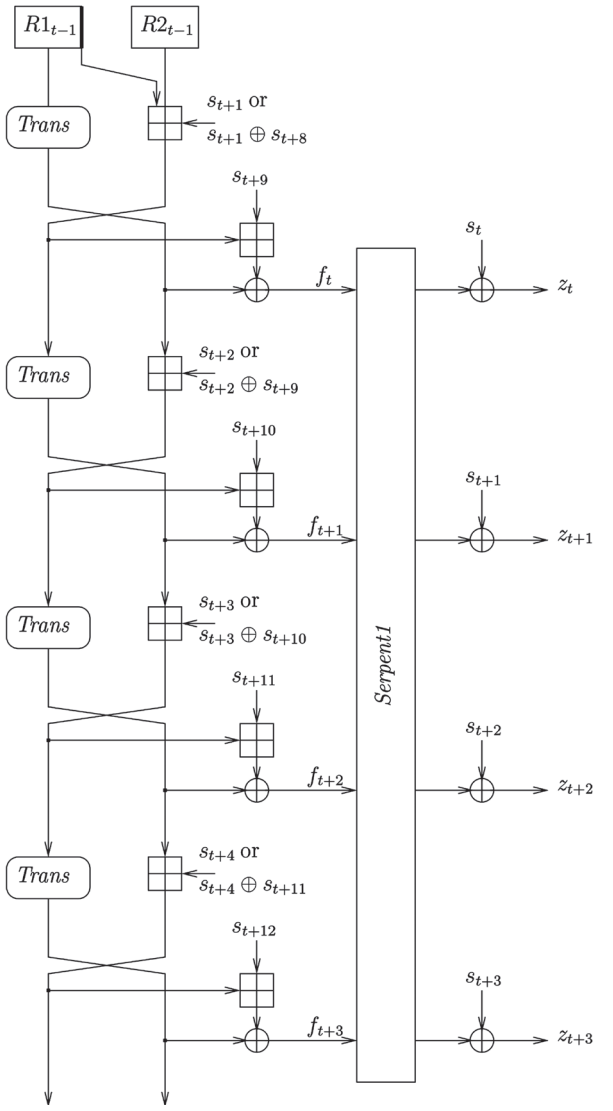

Рис. 4

Робочий процес Sosemanuk. Шифр Sosemanuk комбінує в собі FSM і РЗЛЗЗ для отримання вихідних значень z_t. Момент часу $t = 0$ позначає внутрішній стан після ініціалізації; перше вихідне значення – z_1. На рис. 5 наведено графічне позначення шифру Sosemanuk.

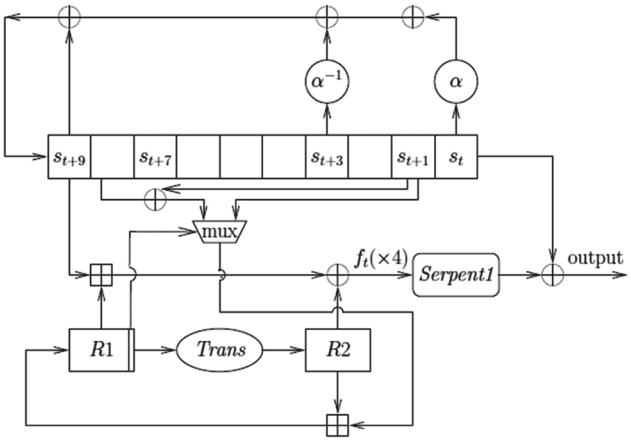

Рис. 5

У момент часу $t \geq 1$ виконуються такі операції:

– оновлюється FSM: $R1_t$, $R2_t$ і проміжне значення f_t обчислюється з $R1t-1$, $R2_{t-1}$, s_{t+1}, s_{t+8} та s_{t+9};

– оновлюється РЗЛЗЗ: s_{t+10} обчислюється виходячи з s_t, s_{t+3} та s_{t+9}. Значення s_t відправляється у внутрішній буфер і РЗЛЗЗ зсувається.

Раз на чотири кроки чотири вихідних значення z_t, z_{t+1}, z_{t+2}, z_{t+3} виходять з накопичених значень $f_t, f_{t+1}, f_{t+2}, f_{t+3}$ та s_t, s_{t+1}, s_{t+2}, s_{t+3}.

Таким чином, алгоритм Sosemanuk виробляє 32-бітові значення. Рекомендується зашифровувати їх у групи по чотири байти, використовуючи порядок байтів від молодшого до старшого, оскільки останній спосіб має більшу швидкість на більш широко використовуваному програмному забезпеченні – платформах високого класу (x86-сумісних). Крім того, цей метод використовує алгоритм SERPENT.

Отже, перші чотири ітерації Sosemanuk є такими:

– початковий стан РЗЛЗЗ містить значення $s_1 - s_{10}$; s_0 не містить ніякого значення. Початковий стан FSM містить $R1_0$ і $R2_0$;

– протягом першого етапу $R1_1$, $R2_1$ і f_1 обчислюються виходячи з $R1_0$, $R2_0$, s_2, s_9 і s_{10};

– з першого етапу отримують буферизовані проміжні значення s_1 і f_1;

– протягом першого етапу зворотне слово s_{11} обчислюється з s_{10}, s_4 і s_1, потім оновлюється внутрішній стан РЗЛЗЗ, що призводить до нового стану, який складається з $s_2 - s_{11}$;

– першими чотирма вихідними значеннями є z_1, z_2, z_3 і z_4, і обчислюються, використовуючи Serpent1 над (f_4, f_3, f_2, f_1), де вихід комбінується операціями XOR з (s_4, s_3, s_2, s_1).

Ключ ініціалізації і ін'єкція IV. Процес ініціалізації алгоритму Sosemanuk розділений на два етапи:

– генерація ключів, яка обробляє секретний ключ, але є незалежною від IV;

– ін'єкція IV, яка використовує результат попередньої операції і IV. Це ініціює внутрішній стан потокового шифру.

Генерація ключів. Генерація ключів відповідає генерації ключів Serpent24, який виробляє 25 128-бітових підключів, як 100 32-бітних слова. Ці 25 128-бітних підключа ідентичні першим 25 128-бітовим підключам, які виробляє генерація ключів алгоритму SERPENT.

Алгоритм SERPENT приймає будь-яку довжину ключа від 1 до 256 біт, отже Sosemanuk може працювати точно з такими ж ключами. Однак, оскільки Sosemanuk спрямований на 128-бітове шифрування, його довжини ключів мають бути принаймні 128 біт. Таким чином, 128 біт є стандартною довжиною ключа. Підтримується будь-яка довжина ключа від 128 біт до 256 біт. Але рівень безпеки як і раніше є 128-бітним. Іншими словами, використання більш довгого секретного ключа не гарантує забезпечення рівня захищеності, такого ж як і довжина ключа.

Ін'єкція IV. Вектор ініціалізації IV – це 128-бітове значення. Воно використовується як вхідні дані блокового шифру Serpent24, який ініційовано попередньою операцією (генерацією ключів). Алгоритм Serpent24 складається з 24 раундів, причому використовуються виходи 12, 18 і 24 раундів. Позначимо ці виходи таким чином:

– $(Y_3^{12}, Y_2^{12}, Y_1^{12}, Y_0^{12})$: вихід 12-го раунду;

– $(Y_3^{18}, Y_2^{18}, Y_1^{18}, Y_0^{18})$: вихід 18-го раунду;

– $(Y_3^{24}, Y_2^{24}, Y_1^{24}, Y_0^{24})$: вихід 24-го раунду.

Вихід кожного раунду складається з чотирьох 32-бітових слів відразу після лінійного перетворення, за винятком 24-го раунду, для якого вихід виходить тільки після складання з двадцять п'ятим підключем. Ці значення використовуються для ініціалізації внутрішнього стану Sosemanuk, з такими значеннями:

$$(s_7, s_8, s_9, s_{10}) = (Y_3^{12}, Y_2^{12}, Y_1^{12}, Y_0^{12});\ (s_5, s_6) = (Y_1^{18}, Y_3^{18});$$

$$(s_1, s_2, s_3, s_4) = (Y_3^{24}, Y_2^{24}, Y_1^{24}, Y_0^{24});\ R1_0 = Y_0^{18};$$

$$R2_0 = Y_2^{18}.$$

Перша покращена властивість процесу ініціалізації Sosemanuk полягає в тому, що він розділений на дві окремі стадії: генерація ключів, що не залежать від IV, і ін'єкція IV, яка генерує початковий стан генератора з вектора ініціалізації IV і виходу схеми генерації ключів. Потім настройка IV для фіксованого ключа є менш затратною, ніж повна установка ключа. Це покращує конструкцію, оскільки зміна вектора ініціалізації IV відбувається частіше, ніж зміна секретного ключа. Другою покращеною характеристикою є те, процес установки вектора ініціалізації IV одержано із застосуванням блокового шифру над IV. Якщо ми розглянемо функцію, яка відображає F_K n-бітний вектор ініціалізації IV у перші n бітів вихідного потоку (що генерується від ключа K і IV), тоді F_K має бути обчислювально невідрізнюваною від випадкової функції.

2. ДОСЛІДЖЕННЯ АПАРАТНО-ОРІЄНТОВАНИХ ПОТОКОВИХ ШИФРІВ

Переможцями у категорії апаратно-орієнтованих шифрів є такі алгоритми: Grain; MICKEY; Trivium. Розглянемо їх структуру, проаналізуємо основні перетворення.

2.1. Потоковий шифр Grain

Потоковий шифр Grain було запропоновано в [1]. Шифр націлений на апаратні середовища з дуже обмеженою кількістю вентилів (gate), споживаної потужності і пам'яті.

Алгоритм шифрування Grain заснований на двох регістрах зсуву і нелінійній функції виводу. Шифр має додаткову функцію, в результаті чого швидкість може бути збільшена за рахунок додаткового обладнання. Складність апаратної реалізації і швидкодії вигідно відрізняє цей алго-

ритм від інших апаратно орієнтованих потокових шифрів, наприклад, таких як E0 і A5 / 1.

Алгоритм Grain є синхронним потоковим шифром, у якому ключовий потік генерується незалежно від тексту. Конструкція заснована на двох регістрах зсуву, один з лінійним зворотним зв'язком (РЗЛЗЗ) і один регістр зсуву з нелінійним зворотним зв'язком (РЗНЛЗЗ). Застосування РЗЛЗЗ гарантує мінімальний період для гами, а також забезпечує збалансованість на виході. Застосування РЗНЛЗЗ разом з нелінійною функцією виходу вводить нелінійність до шифру. Вхід РЗНЛЗЗ маскується з виходу РЗЛЗЗ таким чином, що стан РЗНЛЗЗ стає збалансованим. Отже, в алгоритмі Grain використовується позначення РЗНЛЗЗ хоча це насправді фільтр. Те, що відомо про циклічні структури регістрів зсуву з нелінійним зворотним зв'язком не може бути негайно застосовано для аналізу шифру. Обидва регістра зсуву мають розмір 80 біт. Розмір ключа складає 80 біт і розмір вектора ініціалізації IV складає 64 біта. Структурна схема криптоперетворення схематично зображена на рис. 6.

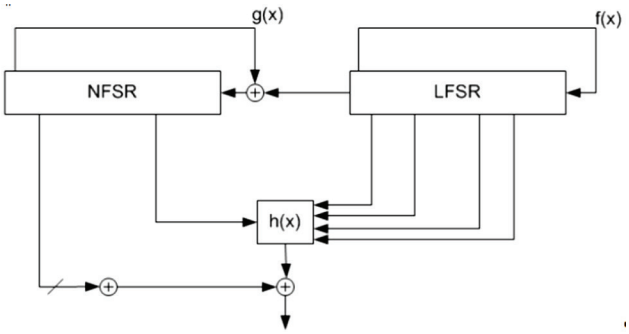

Рис. 6

Шифр складається з трьох основних блоків, а саме РЗЛЗЗ, в РЗНЛЗЗ і вихідної функції. Вміст РЗЛЗЗ позначається як

$$s_i, s_{i+1}, \ldots, s_{i+79},$$

вміст РЗНЛЗЗ позначається як

$$b_i, b_{i+1}, \ldots, b_{i+79}.$$

Поліномом зворотного зв'язку РЗЛЗЗ $f(x)$ є примітивний поліном степеня 80. Він має такий вигляд:

$$f(x) = 1 + x^{18} + x^{29} + x^{42} + x^{57} + x^{67} + x^{80}.$$

Для виключення будь-якої можливої двозначності в специфікації алгоритму шифрування визначено функцію оновлення в РЗЛЗЗ таким чином:

$$s_{i+80} = s_{i+62} + s_{i+51} + s_{i+38} + s_{i+23} + s_{i+13} + s_i.$$

Поліномом зворотного зв'язку РЗНЛЗЗ $g(x)$ визначається як

$$g(x) = 1 + x^{18} + x^{20} + x^{28} + x^{35} + x^{43} + x^{47} + x^{52} + x^{59} + x^{66} + x^{71} + x^{80} + x^{17}x^{20} + x^{43}x^{47} + x^{65}x^{71} + x^{20}x^{28}x^{35} + x^{47}x^{52}x^{59} + x^{17}x^{35}x^{52}x^{71} + x^{20}x^{28}x^{43}x^{47} + x^{17}x^{20}x^{59}x^{65} + x^{17}x^{20}x^{28}x^{35}x^{43} + x^{47}x^{52}x^{59}x^{65}x^{71} + x^{28}x^{35}x^{43}x^{47}x^{52}x^{59}.$$

Функція оновлення РЗНЛЗЗ має вигляд:

$$b_{i+80} = s_i + b_{i+62} + b_{i+60} + b_{i+52} + b_{i+45} + b_{i+37} + b_{i+33} + b_{i+28} + b_{i+21} + b_{i+14} + b_{i+9} + b_i + b_{i+63}b_{i+60} + b_{i+37}b_{i+33} + b_{i+15}b_{i+9} + b_{i+60}b_{i+52}b_{i+45} + b_{i+33}b_{i+28}b_{i+21} + b_{i+63}b_{i+45}b_{i+28}b_{i+9} + b_{i+60}b_{i+52}b_{i+37}b_{i+33} + b_{i+63}b_{i+60}b_{i+21}b_{i+15} + b_{i+63}b_{i+60}b_{i+52}b_{i+45}b_{i+37} + b_{i+33}b_{i+28}b_{i+21}b_{i+15}b_{i+9} + b_{i+52}b_{i+45}b_{i+37}b_{i+33}b_{i+28}b_{i+21}.$$

Слід звернути увагу на те, що біт s_i маскується з входом функції поновлення.

Вміст двох зсувних регістрів РЗЛЗЗ та РЗНЛЗЗ представляє стан шифру. П'ять змінних з цього стану взяті як вхід булевої функції, $h(x)$. Ця функція є нелінійним фільтром, вона обирається так, щоб бути збалансованою, забезпечувати кореляційний імунітет першого порядку і мати алгебраїчну степінь 3. Максимально можливою нелінійністю такої функції може бути 12. Вхід приймається як з РЗЛЗЗ, так і з РЗНЛЗЗ. Булева функція визначається як

$$h(x) = x_1 + x_4 + x_0x_3 + x_2x_3 + x_3x_4 + x_0x_1x_2 + x_0x_2x_3 + x_0x_2x_4 + x_1x_2x_4 + x_2x_3x_4,$$

де змінні x_0, x_1, x_2, x_3 і x_4 відповідають позиціям s_{i+3}, s_{i+25}, s_{i+46}, s_{i+64} і b_{i+63}, відповідно.

Вихідна функція має вигляд:

$$z_i = \sum_{k \in A} b_i + k + h(s_{i+3}, s_{i+25}, s_{i+46}, s_{i+64}, b_{i+63}),$$

де $A = \{1, 2, 4, 10, 31, 43, 56\}$.

Ініціалізація ключа. Перед генерацією ключового потоку шифр має бути ініційований ключем і вектором ініціалізації. Нехай біти ключа k будуть позначені як k_i, причому $0 \le i \le 79$, а біти вектора ініціалізації будуть позначені як IV_i, $0 \le i \le 63$. Ініціалізація ключа робиться таким чином.

По-перше, РЗНЛЗЗ завантажується ключовими бітами:

$$b_i = k_i,\ 0 \le i \le 79,$$

а потім перші 64 біти РЗЛЗЗ завантажуються бітами вектора ініціалізації:

$$s_i = IV_i,\ 0 \le i \le 63.$$

Решта бітів РЗЛЗЗ заповнюються одиницями:

$$s_i = 1,\ 64 \le i \le 79.$$

Через це РЗЛЗЗ не може бути ініційованим нульовим станом (одними нульовими значеннями). Далі шифр функціонує 160 тактів без виводу ключового потоку. Замість цього функція виходу подається назад і виконується операція XOR з входом як для РЗЛЗЗ, так і для РЗНЛЗЗ. Схематично це наведено на рис. 7.

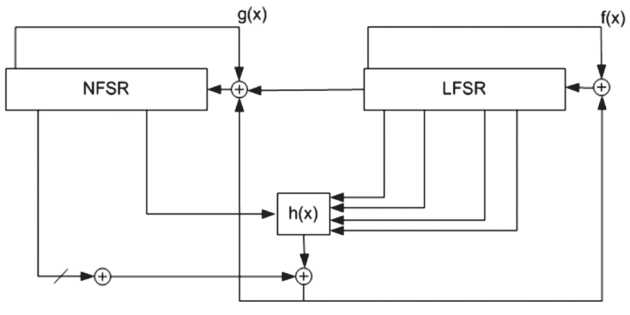

Рис. 7

Можна збільшити швидкість шифру за рахунок більш потужних апаратних засобів. Це може бути зроблено через збільшення кількості реалізацій функцій зворотного зв'язку $f(x)$ і $g(x)$. Внаслідок цього швидкість виведення також буде збільшена в декілька разів. Для того, щоб спростити цю реалізацію останні 15 біт регістрів зсуву s_i, $65 \le i \le 79$ і b_i, $65 \le i \le 79$ не використовуються у функціях зворотного зв'язку або на вході функції фільтра. Це дозволяє легко примножити швидкість до 16 разів, звісно, якщо доступна достатня кількість апаратних ресурсів. Приклад реалізації шифру, коли швидкість подвоюється, можна побачити на рис. 8.

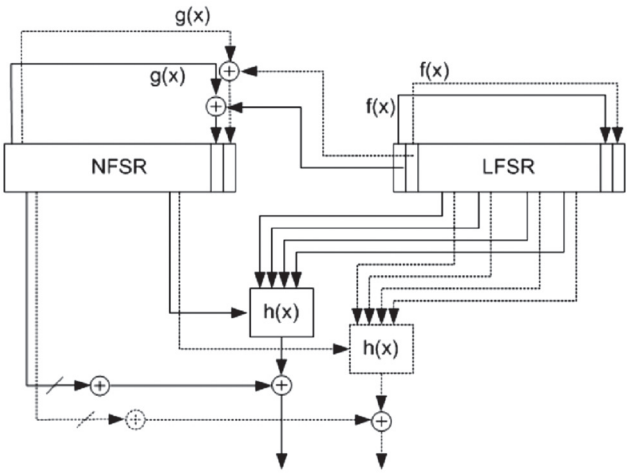

Рис. 8

Природно, що зсувні регістри також мають бути реалізовані так, щоб кожен біт зсувався t кроків замість одного, коли швидкість збільшено з коефіцієнтом t. Зі збільшенням швидкості на коефіцієнт 16, шифр виводить 16 біт за один зсув. Оскільки в ключовий ініціалізації шифру виконується 160 зсувів, можливості для збільшення швидкості обмежено показником $t \le 16$, причому t має бути дільником 160. Число зсувів, що використовуються при ініціалізації ключа дорівнюватиме $160 / t$. Оскільки функції фільтра і зворотного зв'язку досить малі, цілком можливо збільшити швидкість виведення таким чином [1].

2.2. Потоковий шифр MICKEY

В роботі [3] наведено вдосконалену версію 2.0 потокового шифру MICKEY (розшифровується як Mutual Irregular Clocking KEYstream generator – генератор ключового потоку із взаємно нерівномірним рухом), який призначений для апаратних платформ з обмеженими ресурсами.

Вхідні і вихідні параметри. Потоковий шифр MICKEY приймає два вхідних параметри:

– ключ K довжиною 80 біт, його біти позначаються k_0, k_1 ..., k_{79};

– вектор ініціалізації IV довжиною від 0 до 80 біт, його біти позначаються iv_0, ..., $iv_{LENGTH\,IV-1}$.

Послідовність бітів ключового потоку, що формує шифр MICKEY, позначається як $z_0, z_1, \ldots$. Шифртекст виробляється з незашифрованого (відкритого) тексту за допомогою побітової операції XOR з ключовим бітовим потоком, як і в більшості потокових шифрів.

Обмеження для застосування. Максимальна довжина ключового потоку, отриманого за допомогою однієї пари (K, IV) становить 2^{40} біт. Це є прийнятним для генерації подібних послідовностей з одним ключем K, але з різними значеннями IV. Не допускається використання двох векторів ініціалізації різної довжини з одним і тим же ключем K. Звичайно не є прийнятним повторне використання одного і того ж значення IV з фіксованим K.

Складові частини генератора послідовності ключів. Генератор складається з двох регістрів R і S. Довжина кожного регістра дорівнює 100 розрядів, кожен розряд містить 1 біт.

Біти цих регістрів позначаються r_0, r_1, ..., r_{99} та s_0, s_1, ..., s_{99} відповідно. Взагалі, в специфікації алгоритму вказано, що R – це «лінійний регістр», а S – «нелінійний регістр».

Зсув регістра R. Набір входів зворотного зв'язку регістра R:

$RTAPS$ = {0, 1, 3, 4, 5, 6, 9, 12, 13, 16, 19, 20, 21, 22, 25, 28, 37, 38, 41, 42, 45, 46, 50, 52, 54, 56, 58, 60, 61, 63, 64, 65, 66, 67, 71, 72, 79, 80, 81, 82, 87, 88, 89, 90, 91, 92, 94, 95, 96, 97}

Операція зсуву

CLOCK_R (R, $INPUT\_BIT\_R$, $CONTROL\_BIT\_R$)

визначається такою послідовністю дій:

– нехай r_0, r_1, ..., r_{99} – стан регістра R до зсуву, а r'0, r'1, ..., r'99 – після зсуву;

– FEEDBACK_BIT = r99 ⊕ INPUT_BIT_R;

– для всіх $1 \le i \le 99$, r'i = ri-1; r'0 = 0;

– для всіх $0 \le i \le 99$, якщо $i \in$ RTAPS, r'i = r'i ⊕⊕ FEEDBACK_BIT;

– якщо CONTROL_BIT_R = 1, то

– для всіх $0 \le i \le 99$, r'i = r'i ⊕ ri.

Зсув регістра S. Визначимо чотири послідовності

COMP01 ... COMP098, COMP11 ... COMP198,
FB00 ... FB099, FB10 ... FB199

як це наведено у табл. 6.

Функція зсуву регістра

CLOCK_S (S, $INPUT\_BIT\_S$, $CONTROL\_BIT\_S$)

визначається такою послідовністю дій:

– нехай s_0, s_1, ..., s_{99} – стан регістра S до зсуву, а s'_0, s'_1, ..., s'_{99} – після зсуву. Для простоти позначатимемо проміжний стан регістра як $\hat{s}_0$, $\hat{s}_1$, ..., $\hat{s}_{99}$;

– $FEEDBACK\_BIT = s99 \oplus INPUT\_BIT\_S$;

– для всіх $1 \le i \le 98$, $\hat{s}_i = s_{i-1} \oplus ((s_i \oplus COMP0_i) \cdot (s_{i+1} \oplus COMP1_i))$; $\hat{s}_0 = 0$; $\hat{s}_{99} = s_{98}$;

– якщо $CONTROL\_BIT\_S = 0$, тоді для всіх $0 \le i \le 99$, $s'_i = \hat{s}_i \oplus (FB0_i \cdot FEEDBACK\_BIT)$;

– якщо $CONTROL\_BIT\_S = 1$, то для $0 \le i \le 99$, $s'_i = \hat{s}_i \oplus (FB1_i \cdot FEEDBACK\_BIT)$.

Управління рухом всього генератора визначено функцією

Таблиця 6

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
$COMP0_i$		0	0	0	1	1	0	0	0	1	0	1	1	1	1	0	1	0	0	1	0	1	0	1	0
$COMP1_i$		1	0	1	1	0	0	1	0	1	1	1	1	0	0	1	0	1	0	0	0	1	1	0	1
$FB0_i$	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	0	0	1	0	1	1	1	1	1	1
$FB1_i$	1	1	1	0	1	1	1	0	0	0	0	1	1	1	0	1	0	0	1	1	0	0	0	1	0
i	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49
$COMP0_i$	1	0	1	0	1	1	0	1	0	0	1	0	0	0	0	0	0	0	1	0	1	0	1	0	1
$COMP1_i$	0	1	1	1	0	1	1	1	1	0	0	0	1	1	0	1	0	1	1	1	0	0	0	0	1
$FB0_i$	1	1	1	1	0	0	1	1	0	0	0	0	0	0	1	1	1	0	0	1	0	0	1	0	1
$FB1_i$	0	1	1	0	0	1	0	1	1	0	0	0	1	1	0	0	0	0	0	1	1	0	1	1	0
i	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74
$COMP0_i$	0	0	0	0	1	0	1	0	0	1	1	1	1	0	0	1	0	1	0	1	1	1	1	1	1
$COMP1_i$	0	0	0	1	0	1	1	1	0	0	0	1	1	1	1	1	1	0	1	0	1	1	1	0	1
$FB0_i$	0	1	0	0	1	0	1	1	1	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0
$FB1_i$	0	0	1	0	0	0	1	0	0	1	0	0	1	0	1	1	0	1	0	1	0	0	1	0	1
i	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99
$COMP0_i$	1	1	1	0	1	0	1	1	1	1	1	1	0	1	0	1	0	0	0	0	0	0	1	1	
$COMP1_i$	1	1	1	0	0	0	1	0	0	0	0	1	1	1	0	0	0	1	0	0	1	1	0	0	
$FB0_i$	1	1	0	1	0	0	0	1	1	0	1	1	1	0	0	1	1	1	0	0	1	1	0	0	0
$FB1_i$	0	0	0	1	1	1	1	0	1	1	1	1	1	0	0	0	0	0	0	1	0	0	0	0	1

CLOCK_KG (*R*, *S*, *MIXING*, *INPUT_BIT*) таким чином:

– $CONTROL\_BIT\_R = s_{34} \oplus r_{67}$;

– $CONTROL\_BIT\_S = s_{67} \oplus r_{33}$;

– якщо *MIXING* = *TRUE*, тоді $INPUT\_BIT\_R = INPUT\_BIT \oplus s_{50}$, а якщо *MIXING* = *FALSE*, тоді $INPUT\_BIT\_R = INPUT\_BIT$;

– $INPUT\_BIT\_S = INPUT\_BIT$;

– *CLOCK_R*(*R*, *INPUT_BIT_R*, *CONTROL_BIT_R*);

– *CLOCK_S*(*S*, *INPUT_BIT_S*, *CONTROL_BIT_S*).

Завантаження ключа та ініціалізація. Ініціалізація регістрів з вхідних параметрів відбувається таким чином:

– заповнення регістрів *R* і *S* нулями;

– (завантаження в *IV*). Для всіх $0 \le i \le IV_{LENGTH} - 1$,

o CLOCK_KG (*R*, *S*, *MIXING* = *TRUE*, $INPUT\_BIT = iv_i$);

– (завантаження в *K*). Для всіх $0 \le i \le 79$,

o CLOCK_KG (*R*, *S*, *MIXING* = *TRUE*, $INPUT\_BIT = k_i$);

– (перед зсувом). Для всіх $0 \le i \le 99$,

o CLOCK_KG (*R*, *S*, *MIXING* =*TRUE*, $INPUT\_BIT = 0$).

Генерація ключового потоку. Після завантаження та ініціалізації можна генерувати ключовий потік $z_0, \ldots, z_{L-1}$ таким чином:

– Для всіх $0 \le i \le L - 1$,

$$z_i = r_0 \oplus s_0;$$

CLOCK_KG (*R*, *S*, *MIXING* = *FALSE*, $INPUT\_BIT = 0$).

Шифр MICKEY використовує дуже просту функцію виходу ($r_0 \oplus s_0$) для обчислення бітів ключового потоку від станів регістру. Алгоритм шифрування MICKEY має просту апаратну реалізацію і при цьому забезпечуючи високий рівень безпеки. У ньому використовується нерегулярний рух регістрів зсуву, а також нові методи, що забезпечують досить великий період ПВП і стійкість до певних криптоаналітичних атак.

2.3. Потоковий шифр Trivium

Алгоритм Trivium – це апаратно-орієнтований потоковий шифр [7]. Він був сконструйований як приклад спрощення потокового шифру без зменшення рівня безпеки, швидкості і гнучкості. У [7] наведено повний опис потокового шифру Trivium, досліджено особливості реалізації шифру та безпеки алгоритму.

За специфікацією [7] алгоритм Trivium – це потоковий шифр, призначений для генерації 2^{64} біт ключового потоку з 80 біт секретного ключа і 80 біт вектора ініціалізації (табл. 7).

Як і у більшості інших потокових шифрів, процес шифрування складається з двох фаз: спочатку внутрішній стан шифру зініціюється за допомогою ключа та вектора ініціалізації, далі стан кілька разів оновлюється і використовується для генерації бітів ключового потоку. Розглянемо спочатку другу фазу.

Таблиця 7

Параметри Trivium	
Розмір ключа	80 біт
Розмір вектора ініціалізації	80 біт
Внутрішній стан	288 біт

Генерація ключового потоку. Схема шифрування містить 288 біт внутрішнього стану і позначається (s_1, ..., s_{288}). Процес генерації ключового потоку є ітеративним, застосовує значення 15-ти спеціальних станів бітів і використовує їх всі для оновлення трьох бітів стану та підрахунку одного біта ключового потоку z_i.

Далі біти стану, перевертаються і процес повторюється знову, доки не буде згенеровано $N \le 2^{64}$ бітів ключового потоку. Повний опис подається у вигляді простого псевдокоду:

for i=1 to N do

$t_1 \leftarrow S_{66} + S_{93}$

$t_2 \leftarrow S_{162} + S_{177}$

$t_3 \leftarrow S_{243} + S_{288}$

$z_i \leftarrow t_1 + t_2 + t_3$

$t_1 \leftarrow t_1 + S_{91}S_{92} + S_{171}$

$t_2 \leftarrow t_2 + S_{175}S_{176} + S_{264}$

$t_3 \leftarrow t_3 + S_{286}S_{287} + S_{69}$

$(S_1, S_2, ..., S_3) \leftarrow (t_3, S_1, ...S_{92})$

$(S_{94}, S_{95}, ..., S_{177}) \leftarrow (t1, S_{94}, ...S_{176})$

$(S_{178}, S_{279}, ..., S_{288}) \leftarrow (t_3, S_{178}, ...S_{287})$

end for

Зазначимо, що операції «+» і «·» означають додавання і множення в полі GF(2), тобто це, відповідно, операції XOR і AND. Графічне подання процесу генерації ключового потоку зображено на рис. 9.

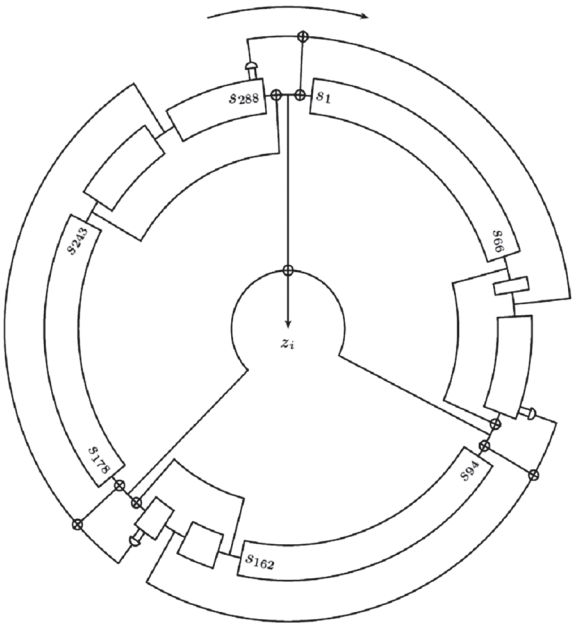

Рис. 9

Установка ключа і вектора ініціалізації. Алгоритм починається з установки 80-бітового ключа і 80-бітового вектора ініціалізації в 288-бітовий внутрішній стан, всі інші біти встановлюються в 0, крім s_{286}, s_{287}, s_{288}. Далі, значення стану перемішується протягом повних чотирьох циклів, таким же чином як було описано вище, однак без генерації потоку ключових біт. Все вищесказане описує псевдокод:

$(S_1, S_2, ..., S_3) \leftarrow (K_1, ..., K_{80}, 0, ...0)$

$(S_{94}, S_{95}, ..., S_{177}) \leftarrow (IV_1, ..., IV_{80}, 0, ...0)$

$(S_{178}, S_{279}, ..., S_{288}) \leftarrow (0, ..., 0, 1, 1, 1)$

for i =1 to 4*288 do

$t_1 \leftarrow S_{66} + S_{91}S_{92} + S_{93} + S_{171}$

$t_2 \leftarrow S_{162} + S_{175}S_{176} + S_{177} + S_{264}$

$t_3 \leftarrow S_{243} + S_{286}S_{287} + S_{288} + S_{69}$

$(S_1, S_2, ..., S_3) \leftarrow (t_3, S_1, ...S_{92})$

$(S_{94}, S_{95}, ..., S_{177}) \leftarrow (t1, S_{94}, ...S_{176})$

$(S_{178}, S_{279}, ..., S_{288}) \leftarrow (t_3, S_{178}, ...S_{287})$

Таким чином, шифр Trivium – це апаратно-орієнтований шифр, конструкція якого сфокусована на гнучкості. Його метою є компактність в середовищі з обмеженими вхідними параметрами, енергоефективність на платформах з малими джерелами енергії, а також швидкість в додатках, які вимагають високошвидкісного шифрування.

3. СТАТИСТИЧНІ ДОСЛІДЖЕННЯ ПОТОКОВИХ ШИФРІВ

Для дослідження статистичної безпеки розглянутих алгоритмів потокового шифрування було застосовано методику [12], згідно з якою оцінювалося математичне сподівання числа пройдених статистичних тестів NIST Special Publication 800-22 [13]. Отримані результат зведено у табл. 8.

В таблиці 8 наведено такі дані:

– «M096» та «M099» – оцінки математичного сподівання (вибіркові середні) числа пройдених статистичних тестів за критерієм $P_j \ge 0{,}96$ та за критерієм $P_j \ge 0{,}99$, відповідно;

– «D096» та «D099» («S096» та «S099») – оцінки дисперсій (середньоквадратичних відхилень) результатів тестування числа пройдених статистичних тестів за критеріями $P_j \ge 0{,}96$ та $P_j \ge 0{,}99$, відповідно;

– «P099» – значення довірчої ймовірності для числа пройдених статистичних тестів за критерієм $P_j \ge 0{,}99$ та при точності $\varepsilon = 2$;

– «P096» – значення довірчої ймовірності для числа пройдених статистичних тестів за критерієм $P_j \ge 0{,}96$ та при точності $\varepsilon = 1$.

В останній колонці «Min096» табл. 8 наведено мінімальні значення числа пройдених статистичних тестів за критерієм $P_j \ge 0{,}96$.

Під час проведення експериментальних досліджень розглядалися різні випадки: змінювався вектор ініціалізації без зміни секретного ключа (позначено у таблиці як «iv»), або змінювався секретний ключ без зміни вектора ініціалізації (позначено у таблиці як «key»). Отримані результати свідчать про високі характеристики досліджуваних криптографічних примітивів. У кожному з розглянутих випадків формовані ПВП мають

Таблиця 8

ПСШ	M099	D099	S099	P099	M096	D096	S096	P096	MIN
Grain_iv	131.63	33.24	5.77	1.00	186.78	0.51	0.71	1.00	182
Grain_key	133.05	24.56	4.96	1.00	186.91	0.39	0.62	1.00	182
hc-128_iv	132.49	37.51	6.12	1.00	186.52	2.30	1.52	1.00	176
hc-128_key	132.84	40.05	6.33	1.00	186.97	0.31	0.55	1.00	183
MICKEY_iv	133.17	19.82	4.45	1.00	186.78	0.47	0.68	1.00	181
MICKEY_key	131.62	35.89	5.99	1.00	186.80	0.28	0.53	1.00	183
rabbit_iv	133.01	30.32	5.51	1.00	186.65	1.03	1.01	1.00	180
rabbit_key	131.67	26.47	5.14	1.00	186.72	1.66	1.29	1.00	179
salsa20_iv	132.61	29.14	5.40	1.00	186.83	0.99	1.00	1.00	180
salsa20_key	133.64	26.58	5.16	1.00	186.97	0.31	0.55	1.00	183
sosemanuk_iv	132.42	21.44	4.63	1.00	186.77	0.53	0.73	1.00	181
sosemanuk_key	132.98	37.55	6.13	1.00	186.76	0.73	0.86	1.00	180
Trivium_iv	133.71	25.29	5.03	1.00	187.05	0.32	0.57	1.00	183
Trivium_key	132.27	31.22	5.59	1.00	186.83	0.86	0.93	1.00	181

вигляд псевдовипадкових і в статистичному сенсі не відрізняються від деякої гіпотетичної «випадкової послідовності». Експериментальні дані отримані з високою точністю і достовірністю, тобто можна стверджувати, що досліджувані алгоритми потокового симетричного криптоперетворення є статистично безпечними у будь-яких практичних застосуваннях.

ВИСНОВКИ

Проведений аналіз потокових криптоперетворень, які відібрані як переможці міжнародного проекту eSTREAM, показав, що за своєю специфікацією відповідні алгоритми поділяються на дві функціональні групи. Перша група алгоритмів це програмно-орієнтовані потокові шифри, до яких відносяться: HC-128; Rabbit; Salsa 20/12; SOSEMANUK. До другої групи апаратно-орієнтованих шифрів відносяться алгоритми Grain; MICKEY та Trivium.

Аналіз структури та основних застосованих функцій перетворення програмно-орієнтованих потокових шифрів (HC-128, Rabbit, Salsa 20/12, SOSEMANUK) показав, що у відповідних алгоритмах використовуються переважно прості та обчислювально ефективні операції побітового додавання за модулем 2, AND, циклічні зсуви, інші логічні функції, з можливістю застосовувати всі переваги розпаралелювання процесів формування ключового потоку. Стан генератора подається, як правило, або у вигляді слів, або табличним способом, що значно спрощує реалізацію обчислень при програмній реалізації. Виключенням можна вважати алгоритм SOSEMANUK, в якому для подання стану генератору використовується регістр зсуву довжини 10, але невелика довжина цього регістру не впливає суттєво на зменшення швидкодії навіть при програмній реалізації. Слід також відмітити, що потокові криптоперетворення застосовують елементи конструкції БСШ, зокрема, алгоритми SOSEMANUK та HC-128 застосовують нелінійні вузли заміни шифру AES.

Аналіз апаратно-орієнтованих шифрів, до яких відносяться алгоритми Grain, MICKEY та Trivium, показав, що в їхній структурі як правило використовуються регістри зсуву великої довжини. Це надає змогу при апаратній реалізації за один часовий такт роботи пристрою змінювати стан генератора ключового потоку, що дозволяє досягти надзвичайно високих показників швидкості формування ПВП. Для забезпечення криптографічної стійкості в структуру генератора додається також нелінійна функція, яка реалізується або шляхом нелінійного зворотного зв'язку у регістрах зсуву, або, наприклад, через застосування криптографічної булевої функції з високими показниками нелінійності. Складність криптоаналізу підвищується також через застосування нерівномірного руху регістрів, яке ускладнює вихідну ПВП.

Отримані результати свідчать про високі показники статистичної безпеки ПСШ, досліджувані алгоритми потокового криптоперетворення є статистично безпечними у будь-яких практичних застосуваннях.

Література

[1] The eSTREAM Project - eSTREAM Phase 3. Grain (Portfolio Profile 2). [Електронний ресурс]. – Режим доступу: http://www.ecrypt.eu.org/stream/grainpf.html.

[2] The eSTREAM Project - eSTREAM Phase 3. HC (Portfolio Profile 1). [Електронний ресурс]. – Режим доступу: http://www.ecrypt.eu.org/stream/hcpf.html.

[3] The eSTREAM Project - eSTREAM Phase 3. MICKEY (Portfolio Profile 2). [Електронний ресурс]. – Режим доступу: http://www.ecrypt.eu.org/stream/mickeypf.html.

[4] The eSTREAM Project - eSTREAM Phase 3. Rabbit (Portfolio Profile 1). [Електронний ресурс]. – Режим доступу: http://www.ecrypt.eu.org/stream/rabbitpf.html.

[5] The eSTREAM Project - eSTREAM Phase 3. Salsa20 (Portfolio Profile 1). [Електронний ресурс]. – Режим доступу: http://www.ecrypt.eu.org/stream/salsa20pf.html.

[6] The eSTREAM Project - eSTREAM Phase 3. SOSEMANUK (Portfolio Profile 1). [Електронний ресурс]. – Режим доступу: http://www.ecrypt.eu.org/stream/ sosemanukpf.html.

[7] The eSTREAM Project - eSTREAM Phase 3. Trivium (Portfolio Profile 2). [Електронний ресурс]. – Режим доступу: http://www.ecrypt.eu.org/stream/triviumpf.html.

[8] NESSIE Call for Cryptographic Primitives, Version 2.2 [Електронний ресурс]. – March, 2000. – Режим доступу: http://cryptonessie.org.

[9] NESSIE security report, deliverable D21, version 1.0 [Електронний ресурс]. – October 30, 2002ю – Режим доступу: http://www.cryptonessie.org.

[10] *Marcus Schafheutle.* A First Report on the Stream Cipher SNOW. [Електронний ресурс]. – Режим доступу: http://www.cryptonessie.org.

[11] *Ross J. Anderson.* Serpent: A Candidate Block Cipher for the Advanced Encryption Standard. University of Cambridge Computer Laboratory. Retrieved 2013-01-14. [Електронний ресурс]. – Режим доступу: http://www.cl.cam.ac.uk/~rja14/serpent.html.

[12] *Кузнецов А.А., Мордвинов Р.И., Колованова Е.П., Самойлова А.В.* Методика статистического тестирования криптографических алгоритмов // Спеціальні телекомунікаційні системи та захист інформації. – Київ – 2014. – №1(25). – С.54-61

[13] NIST Special Publication 800-22. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. [Электронный ресурс]. Режим доступа: http://csrc.nist.gov/publications/nistpubs/ 800-22-rev1a/SP800-22rev1a.pdf

Надійшла до редколегії 15.10.2015

Кузнецов Олександр Олександрович, доктор технічних наук, професор, професор кафедри БІСТ ХНУ ім. В.Н. Каразіна. Наукові інтереси: криптографія та стеганографія, методи обробки та передачі інформації.

Іваненко Дмитро Вікторович, кандидат технічних наук, доцент кафедри БІТ ХНУРЕ. Наукові інтереси: криптографія, криптоаналіз, атаки спеціального виду.

Мордвінов Руслан Ігорович, кандидат технічних наук, м.н.с. кафедри БІСТ ХНУ ім. В.Н. Каразіна. Наукові інтереси: криптографія, криптоаналіз, БСШ, генератори випадкових чисел.

УДК 004.056.55

Исследование поточных алгоритмов шифрования – победителей международного проекта eSTREAM / А.А. Кузнецов, Д.В. Иваненко, Р.И. Мордвинов // Прикладная радиоэлектроника: научн.-техн. журнал. – 2015. – Том 14. – № 4. – С. 321–334.

Анализируются поточные алгоритмы шифрования, которые являются победителями международного проекта eSTREAM. Исследуется базовая структура алгоритмов-победителей, примененные функциональные связи, слои линейных и нелинейных преобразований и другое. Проводятся экспериментальные исследования статистической безопасности по методике тестирования NIST Special Publication 800-22. Установлено, что выходные последовательности исследуемых шифров соответствуют современным требованиям, рассмотреные криптопреобразования могут быть использованы при разработке разных вариантов построения национального стандарта потокового шифрования Украины.

Ключевые слова: криптографические преобразования, поточный шифр, статистические тесты.

Табл.: 8. Іл.: 9. Библиогр.: 13 назв.

UDK 004.056.55

Researching stream encryption algorithms – the winners of the international project eSTREAM / A.A. Kuznetsov, D.V. Ivanenko, R.I. Mordvinov // Applied Radio Electronics: Sci. Journ. — 2015. — Vol. 14. — № 4. — P. 321–334.

Stream encryption algorithms are analyzed that determine the winners of the international project eSTREAM. The paper studies the basic structure of the winning algorithms, the applied functional ties, the layers of linear and nonlinear transformations etc. Pilot studies of statistical security using the method of testing NIST Special Publication 800-22 are conducted. It is found that the output sequence of investigated ciphers meet modern requirements, cryptotransformations are considered that can be used for the development of different variants of the Ukraine's national standard of streaming encryption.

Keywords: cryptographic transformation, stream cipher, static tests.

Tab.: 8. Fig.: 9. Ref.: 13 items.