**Алгоритм LOKI97**

**Історія створення алгоритму**

Алгоритм LOKI був розроблений у 1989 році професорами Лоуренсом Брауном (Lawrence Brown), Йозефом Пєпржиком (Josef Pieprzyk) і Дженніфер Себеррі (Jennifer Seberry) з Академії Збройних сил Австралії. Цікаве походження назви алгоритму - від імені скандинавського бога Локі, якого видання описує як “Злісний бог, батько чудовищ. Завдавав богам багато шкоди...”. Згодом початковий алгоритм LOKI отримав назву LOKI89 для відмінності від останніх версій алгоритму.

Алгоритм LOKI був помічений світовою криптологічною спільнотою, зокрема, завдяки його участі в австралійському конкурсі криптоалгоритмів Auscrypt90, що проходив у Сіднеї в січні 1990 р. У 1991 р. з’явився алгоритм LOKI91, який був підсиленням попереднього (розроблений переліченими вище авторами за участю Метью Квана). Однак в алгоритмі було знайдено декілька недоліків, які не сприяли широкому поширенню даного алгоритму, зокрема:

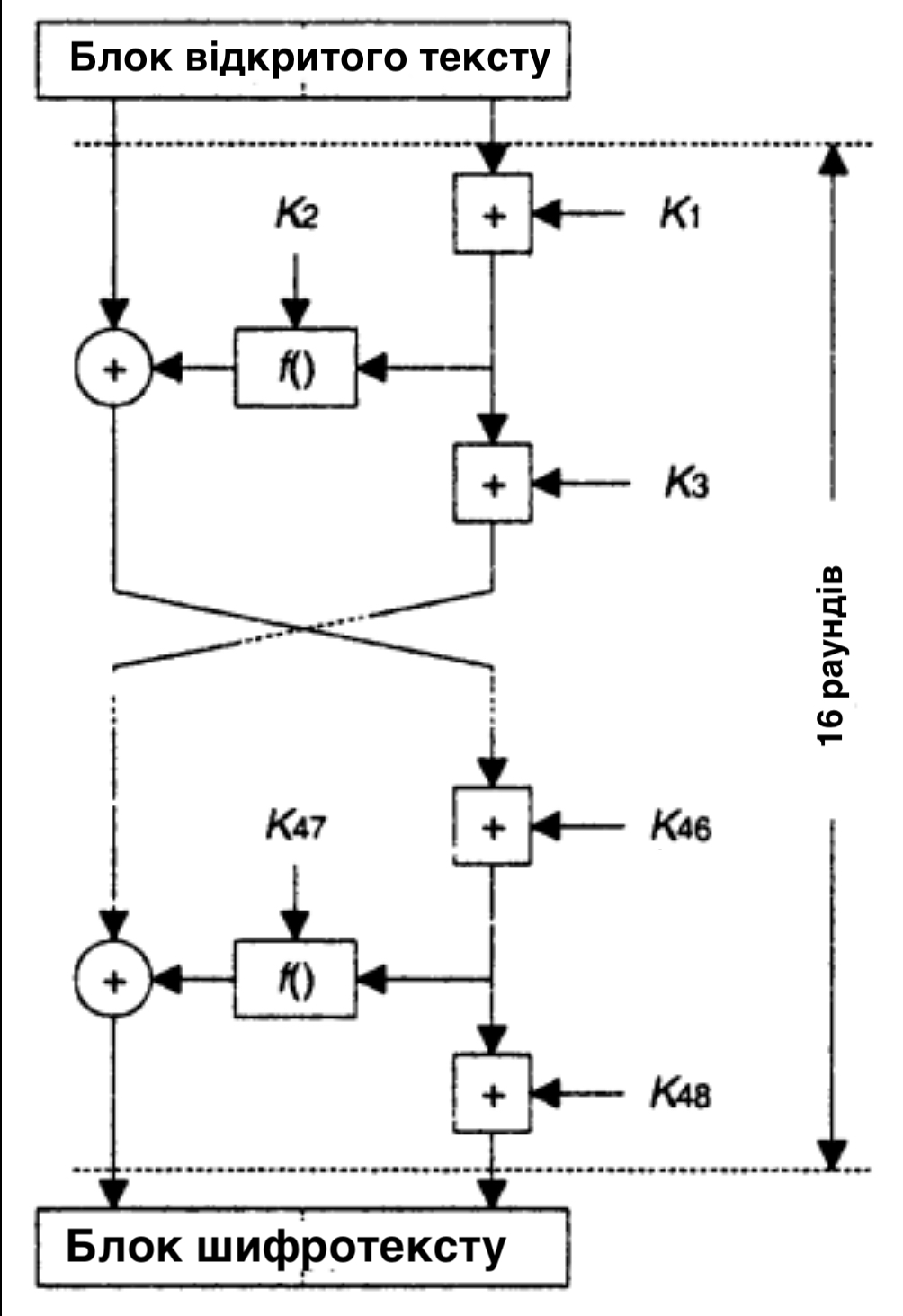
* 64-бітовий секретний ключ алгоритму не міг протистояти атакам, що виконуються методом «грубої сили»;
* можливість атаки, що базується на зв'язаних ключах і відбувається через слабкість схеми розширення ключа, яка встановлювала підвищені вимоги до генераторів ключів шифрування, які використовуються в даному алгоритмі;
* було виявлено ефективність лінійного криптоаналізу проти алгоритму LOKI91 з усіченою кількістю раундів (до 12 раундів замість 16 у основного варіанту алгоритму).

У результаті подальших досліджень авторів алгоритму LOKI з’явився алгоритм LOKI97.

**Основні характеристики та структура алгоритму**

Алгоритм LOKI97 шифрує дані блоками по 128 бітів і використовує ключі розміром 128, 192 або 256 бітів, що відповідає основним вимогам конкурса AES.

Структура алгоритму подана на малюнку:

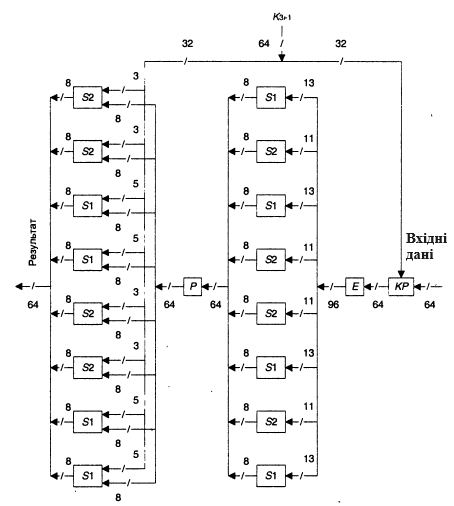


Як видно з малюнку, алгоритм базується на мережі Фейстеля: дані блоку, що шифрується розбиваються на два субблоки по 64 біти, над одним з яких виконуються наступні операції:

1. Додавання з першим фрагментом ключа раунду за модулем 264. Процедура розширення ключа шифрування відносно складна і детально буде описана далі.
2. Накладання функції f(), яка виконує нелінійне перетворення даних і використовує як аргумент другий фрагмент ключа раунду.
3. Знову додавання за модулем 264 з третім фрагментом ключа раунду.

Після виконання перерахованих дій оброблений субблок накладається на інший застосуванням операції XOR, після чого субблоки міняються місцями (за винятком останнього раунду).

Функція f() є досить складною, вона передбачає такі перетворення даних:



1. KP (keyed permutation) - проста перестановка вхідних даних на основі молодших 32 бітів використовуваного функцією f() фрагмента ключа раунду K3i-1, де і - номер раунду. Перестановка виконується так:

* вхідні дані розбиваються на 2 субблоки по 32 біти;
* для кожного одиничного біта фрагмента ключа виконується перестановка відповідних йому бітів субблоків між собою (наприклад, нульовий біт ключа має значення 1; у цьому випадку нульовий біт першого субблока вхідних даних набуває значення нульового біта другого субблоку, і навпаки).

1. E (expansion) - генерація 96-бітної послідовності на основі 64-бітного результату операції КР. Послідовність формується на основі наступних бітів входу (тут і далі 0-й біт - наймолодший біт вхідної послідовності):

[4-0,63-56,58-48,52-40,42-32,34-24,28-16,18-8,12-0].

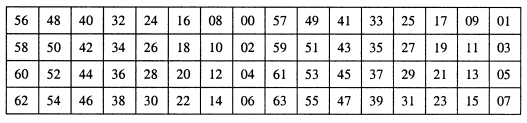
1. S1, S2 (substitution boxes) - таблиці замін; S1 обробляє 13-бітові фрагменти 96-бітної послідовності, отриманої операцією Е, S2 - 11-бітові фрагменти. Заміна фактично виконується шляхом зведення в куб інвертованого вхідного значення *x* в полі GF(213)(S1) або GF(211)(S2) згідно з наступними формулами (у формулах зазначені шістнадцяткові константи):

S1(*x*) = ((*x*⊕1FFF)3 mod 2911)&FF;

S2(*x*) = ((*x*⊕7FF)3 mod AA7)&FF.

Залежно від наявності ресурсів у конкретних застосуваннях алгоритму, S1 і S2 можуть бути реалізовані як у вигляді наведених вище обчислень, так і у вигляді таблиць замін.

1. P (permutation) – побітова перестановка вхідної послідовності згідно таблиці:



Тобто 63-й біт вхідної послідовності стає 56-м бітом результату, 62-й біт стає 48-м і т.д.

1. 64-бітний результат перестановки P і 32-бітна старша частина фрагменту ключа раунду повторно обробляється таблицею замін. При цьому вхідна 64-бітна послідовність порівно (по 8 бітів) розподіляється між таблицями, а з фрагменту ключа раунду беруться відсутні біти до необхідних 13 бітів входу для таблиці S1 або 11 бітів входу для таблиця S2. Тобто, вхідна послідовність для таблиця формується так:

* [біти 63-61 K3i-1, біти 63-56 виходу операції P] - 11 бітів для S2;
* [біти 60-58 K3i-1, біти 55-48 виходу операції P] - 11 бітів для S2;
* [біти 57-53 K3i-1, біти 47-40 виходу операції P] - 13 бітів для S1 і т.д.

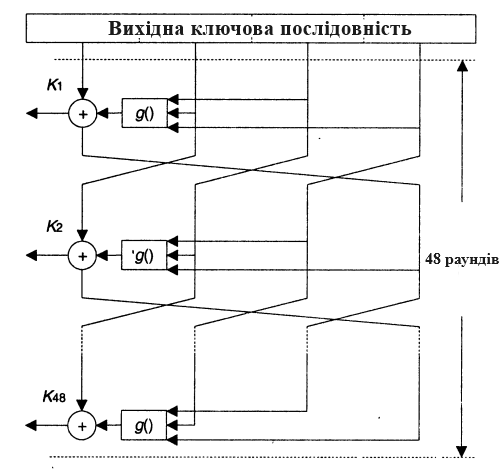
**Процедура розширення ключа**

Процедура розширення ключа на основі ключа шифрування забезпечує вироблення 48 фрагментів ключів раундів К1...К48, по три з яких використовуються в кожному раунді алгоритму, як описано вище.

На початку цієї процедури виконується ініціалізація вихідної 256-бітної ключової послідовності, яка, залежно від довжини ключа шифрування, виконується одним із наступних способів:

* 256-бітний ключ використовується у незмінному вигляді;
* 192-бітний ключ доповнюється результатом застосування описаної вище функції f(), яка як дані, що обробляються (перший аргумент) використовує перші 64 біта ключа шифрування, а як підключ (другий аргумент) - наступні 64 біти ключа шифрування;
* 128-бітовий ключ доповнюється результатом застосування функції f() зі зворотним відносно 192-бітного ключа порядком аргументів, а потім доповнюється аналогічним 192-бітного ключа чином.

Наступна обробка вихідної ключової послідовності заснована на мережі Фейстеля, після кожного раунду якої з'являється фрагмент ключа алгоритму раунду. Таким чином, виконується 48 раундів перетворень.

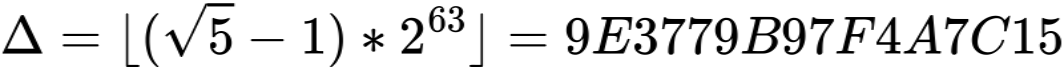


У кожному раунді три 64-бітові фрагменти поточного стану ключової послідовності обробляються функцією g() і накладаються операцією XOR на 64-бітовий фрагмент, що залишився. Функція g() описується так:

g(x,y,z) = f(x + y + ∆\*i,z),

де:

* i - номер поточного раунду процедури розширення ключа;
* ∆ - константа, яка визначається так:



**Чому LOKI97 не вийшов у фінал конкурсу AES**

При детальному розгляді алгоритму в рамках конкурсу AEЅ експерти зробили висновки про те, що даний алгоритм не має жодних переваг у криптостійкості порівняно з іншими алгоритмами, що брали участь у конкурсі. При цьому в алгоритмі було знайдено такі недоліки:

* дуже низька (в порівняні з іншими учасниками конкурсу) швидкість шифрування;
* дуже високі вимоги до ресурсів, особливо до енергонезалежної пам'яті, що унеможливлює реалізацію алгоритму в смарт-картах або аналогічних пристроях, що мають обмежені ресурси (а можливість застосування алгоритму в смарт-картах була однією з важливих вимог конкурсу);
* можливість криптоатаки у випадку наявності у зловмисника близько 256 пар «відкритий текст - зашифрований текст».

Будь-якого з цих недоліків було достатньо для відсутності алгоритму LOK197 у другому раунді конкурсу AEЅ.