УДК 004.056
DOI: 10.31673/2409-7292.2025.011455

Щербина Ю. В., Казакова Н. Ф., Логінова Н. І., Фразе-Фразенко О. О., Фоміна Н. Б.

АНАЛІЗ СПОСОБІВ УДОСКОНАЛЕННЯ АЛГОРИТМУ ПОТОКОВОГО ШИФРУВАННЯ RC4

У статті розглядаються проблеми, пов'язані з підвищенням стійкості шифру RC4, що використовується в мережних протоколах сучасних розподілених комп'ютерних систем. Протягом останнього десятиріччя в алгоритмі шифрування RC4 було виявлено деякі слабкі місця, наявність яких пояснюється простотою закладеного в нього алгоритму. Головним таким слабким місцем є кореляція між секретним ключем і, сформованою на його основі, шифруючою гамою. В статті надано аналіз запропонованих різними авторами способів усунення цього недоліку за рахунок ускладнення алгоритмів планування ключів та формування псевдовипадкових чисел. Зроблено висновок про те, що рішення, основані на простому збільшенні складності алгоритму за рахунок втілення додаткових криптографічних перетворень, не є ефективним, оскільки це позбавляє шифр RC4 його швидкодії. Через це, перевагу пропонується надавати рішенням, основаним на 32/64 бітних мікропроцесорних системах, що дозволяють суттєво збільшити простір внутрішніх станів шифрувального алгоритму без помітного зниженні швидкості шифрувального процесу.

Ключові слова:потоковий шифр RC4, випадкова перестановка, ключовий потік, шифруючи гама, крипто аналіз, слабкий внутрішній стан, слабкі ключі.

Вступ

Для захисту інформації в комп'ютерних протоколах стандарт блочного шифрування AES надає достатньо ефективні засоби. Однак там, де обчислювальні ресурси і швидкість оброблення даних обмежені, перевага надається потоковому шифруванню. Потокові шифри забезпечують високу швидкість роботи і можуть бути адаптовані до конкретних умов їх реалізації. Розроблений у 1987 році Роном Рівестом потоковий шифр RC4, в минулі роки був де-факто стандартом шифрування. Wireless Equivalency Protection (WEP), а також ранні версії TLS/SSL – це ті відомі схеми шифрування, де він був втілений. Цей шифр вкрай невибагливий до витрат обчислювальних ресурсів і не має специфічних вимог до апаратного забезпечення. Все це робить його ідеальним варіантом в системах захисту даних, що працюють в реальному масштабі часу і призначені для шифрування відносно коротких повідомлень. Нажаль простота реалізації є причиною наявності в цьому шифрі деяких слабких місць, через що від нього поступово почали відмовлятись там, де, ще недавно, він був основним засобом шифрування. В той же час, його модифіковані версії залишаються криптографічно безпечними і на даний момент вони є складовою частиною таких сучасних мережних протоколів як Secure Sockets Layer (SSL), Transport Layer Security (TLS), Wired Equivalent Privacy (WEP) та Wi-Fi Protected Access (WPA).

Постановка проблеми

До захисту конфіденційності інформаційних потоків, що передаються в комп'ютерних мережах, де пріоритетом є швидке їх оброблення у реальному часі, зазвичай використовуються потокові шифри, оскільки вони краще вписуються в діючі протоколи обміну даними. З іншого боку, там, де пріоритетом є високий рівень безпеки, там доводиться користуватись блочними шифрами, не дивлячись на їх, відносно невисоку швидкість шифрування. Таким чином, вибір способу шифрування ґрунтується на правильному розумінні умов його використання.

Практика минулих років показує, що добре спроектовані потокові шифри можуть забезпечувати такий рівень безпеки, що в достатній мірі відповідає конкретним умовам роботи. Так, наприклад, один з фіналістів конкурсу eSTREAM HC-128, можна розглядати як наступне покоління програмного потокового шифру RC4. Він, хоча і забезпечує високу криптографічну стійкість, але сильно поступається RC4 у швидкодії та вимогах до пам'яті. Саме цим пояснюється поява великої кількості робіт, присвячених різним способам такого удосконалення RC4, які дозволяють зберегти його позитивні сторони.

Аналіз останніх публікацій

Шифр RC4 був створений Роном Рівестом у 1987 році і зберігався в таємниці до свого опублікування у 1994 році. Його стійкість визначається таємним внутрішнім станом, що описується перестановкою 2^n n-бітних слів в масиві S, двома індексами (i та j) у цьому масиві, які також змінюються від 0 до $2^n - 1$. Розмір слова $n = 8$ був обраний Роном Рівестом через те, що на момент створення шифру для комерційного використання були доступні лише 8/16-бітні процесори, а зростання n приводило до експоненційного збільшення об'єму пам'яті для зберігання масиву S. Таким чином, для $n = 8$, кількість варіантів внутрішнього стану складає $\log_2(2^8! \times (2^8)^2) \approx 2^{1700}$ біт і, через це, вгадати навіть малу частину цього стану вкрай важко. Сьогодні вже доступні 32/64-бітні процесори і вартість доступної пам'яті значно знизилась, але збільшувати n більше 16 не ватро, тому що тримати у пам'яті таблицю, наприклад, розміром 2^{32} через її величезний розмір, вже непрактично [1].

По суті, алгоритм RC4 включає дві складових частини. Перша – це алгоритм планування ключів (Key Scheduling Algorithm – KSA). Саме він ініціалізує внутрішній стан шифру. Друга його частина – це алгоритм формування псевдовипадкової шифруючої гами G (Pseudo-Random Number Generator – PRNG) [2].

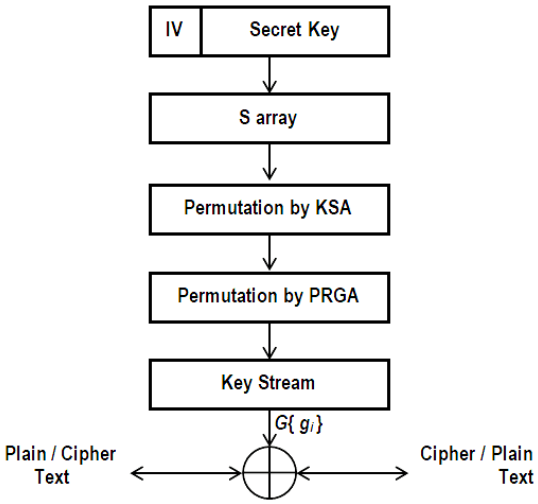

Рис. 1. Загальний алгоритм потокового шифрування RC4

Спочатку алгоритм KSA приймає секретний l-розрядний ключ K і початковий N-розрядний ($N = 2^n$) масив S, що містить числа $\{0, 1, 2, \ldots, 2^n - 1\}$. Потім відбувається скремблювання чисел в масиві, а саме, цей процес протікає за принципом:

```
for(int i = 0; i < 256; i++)                // Ініціалізація
    S[i] = i;

j = 0;
for(int i = 0; i < N – 1; i++) {            // Скремблювання
  j = (j + S[i] + K[i % l]) % N;
  t = S[i]; S[i] = S[j]; S[j] = t;          // Перестановка
}
```

Загальна кількість варіантів таких перестановок складає $N!$.

Формування елементів L-розрядної шифруючої гами G відбувається у відповідності з наступним алгоритмом:

```
i = 0; j = 0;
while (i < L)
{
   i = (i + 1) % N;
```

© Щербина Ю. В., Казакова Н. Ф., Логінова Н. І., Фразе-Фразенко О. О., Фоміна Н. Б. (2025). Аналіз способів удосконалення алгоритму потокового шифрування RC4. Сучасний захист інформації, 1(61), 23–30. https://doi.org/10.31673/2409-7292.2025.011455.

```
   j = (j + S[i]) % N;
   t = S[i]; S[i] = S[j]; S[j] = t;   // Перестановка
   t = (S[i] + S[j]) % N;
   g = S[t]
}
```

Тут *i* ітеративно збільшується на одиницю у межах від 0 до $N - 1$, а *j* визначається через *i*, з урахуванням *i*-го елементу масиву *S*. Далі відбувається перестановка *i*-го і *j*-го елементів масиву *S*, а вихідний потоковий символ шифруючої гами g_i обчислюється за правилом:

```
g = S[(S[i] + S[j]) % N].
```

Перетворення поточних байтів відкритого тексту у відповідні зашифровані байти відбувається шляхом складання їх по модулю два з відповідними байтами шифруючої гами *g* так, як це показано на рисунку 1. Причому, цей процес є двостороннім, а саме, розшифровування відбувається так само як і шифрування, оскільки операція складання по модулю два є зворотною.

Слабким місцем алгоритму RC4 є зміщення у перших кількох байтах шифруючої гами, що утворюється на виході PRNG, і дає можливість криптоаналітику відновити секретний ключ *K*. Термін "зміщення" був введений для кількісної оцінки нерівномірності розподілення нульових і одиничних символів у випадковій послідовності і визначається як:

$$b = \frac{|p_1 - p_0|}{2},$$

де p_1 і p_0, ймовірності одиничних і нульових символів, відповідно. Для послідовності, що не відрізняється від абсолютно випадкової, показник *b* має дорівнювати 0,5 [3].

Вектор ініціалізації (initialization vector – IV), що утворюється алгоритмом KSA і зберігається у масиві *S*, формується на основі відносно короткого секретного ключа *K*. Наявність зміщення говорить про те, що кожен біт ключа *K* впливає не на всі символи IV під час його формування. Саме це дає змогу супротивнику встановити ключ *K* на основі аналізу перехоплених повідомлень, зашифрованих одним ключем. Ця уразливість використовується для організації атаки тільки на зашифрований текст у деяких мережних протоколах, де одне і те ж повідомлення циркулярно передається різним суб'єктам з різними ключами *K*.

Переважна більшість слабких місць у потокових шифрах і у RC4, в тому числі, визначається математичними співвідношеннями між відкритим та закритим текстами і змістом ключа *K*. Таким чином, основна ціль захисту полягає у створенні алгоритму, що формує шифруючу гаму *G*, яка не відрізняється від абсолютно випадкового числового потоку і не дає змоги знайти ключ *K*, інакше, як методом грубої сили [4]. Відповідно, основною ціллю супротивника є розроблення алгоритму криптографічної атаки на шифр RC4 на основі результатів дослідження або невипадковості внутрішнього стану PRNG, або на основі виявлених аномалій вихідного потоку шифруючої гами *G*.

В роботі [5] показано, що слабкими місцями в алгоритмі RC4 є:

– наявність обмеженого набору слабких ключів, що залишають "сліди" у векторі ініціалізації після KSA, або у вихідній шифруючій гамі після PRNG;

– присутність декількох зміщень або кореляцій, пов'язаних із секретним ключем *K* або змінними частинами стану PRNG, або короткостроковими і довгостроковими зміщеннями у складі вихідної шифруючої гами *G*;

– наявність колізій між кількома ключами, що приводять до створення однакових потоків шифруючої гами;

– можливість з будь-якого стану PRNG перейти у будь-який інший стан і знаходження на його основі секретного ключа *K*;

– можливість знаходження секретного ключа *K* з вихідного потоку шифруючої гами *G*;

– можливість знаходження поточного стану PRNG не зважаючи на велику їх кількість.

Основна частина атак на шифр RC4 будуються на основі використання описаних слабких місць [6], розділяється на такі види як:

– атаки на основі тільки зашифрованого тексту (Ciphertext only) – знаходження секретного ключа *K* або відкритого тексту (Plain text) тільки на основі перехопленого зашифрованого повідомлення;

– атаки з відомим відкритим текстом (Known plaintext) – знаходження секретного ключа *K* або відкритого тексту (Plain text) на основі відомого відкритого тексту повідомлення та відповідного зашифрованого повідомлення;

– атаки з відомим відкритим текстом, який криптоаналітик може обирати (Chosen plaintext) – знаходження секретного ключа *K*, на основі обраного відкритого тексту та відповідного зашифрованого повідомлення;

– атаки з відомим вектором ініціалізації (known initialization vector) – отримання відкритого тексту з використанням шифруючої гами *G*, сформованої на основі підібраного або відомого вектора ініціалізації IV.

Популярність шифру RC4 пояснюється його невибагливістю до обчислювальних ресурсів та високою швидкодією. Саме через це він широко використовується для захисту internet-трафіку. Але простота алгоритму RC4 і є причиною, що обумовлює наявність у ньому слабких місць. Головною з цих причин є зміщення у послідовності псевдовипадкових значень індексів *j*, що генеруються KSA. Щоб уникнути цієї уразливості необхідно покращити рівномірність розподілення символів у складі вектора ініціалізації IV, для чого варто було б обрати більш складний спосіб генерації індексу *j*.

Для усунення нерівномірності розподілення послідовності індексів *j* в алгоритмі KSA, в роботі [7] пропонується змінити звичайний порядок індексів *i* на більш складний: спочатку міняти його з середини масиву *S* вліво до його початку, а потім із середини цього масиву – вправо до його кінця. Далі, на додатковому етапі запропонованого алгоритму KSA+, виконується ще одна перестановка детермінованих індексів *i* типу “зіг-заг” у порядку:

0, 255, 1, 254, 2, 253, ..., 125, 130, 126, 129, 127, 128.

Автори роботи стверджують, що запропонований алгоритм усуває вразливості шифру, але час скремблювання збільшується у тричі.

В роботі [8] автори пропонують на етапі KSA розділити вектор ініціалізації IV на дві однакових частини, для чого, так само поділити навпіл масив *S*, і заповнювати кожну його частину окремо на основі загального секретного ключа *K*. Далі, на етапі PRNG, для кожного індексу *i*, передбачається формування двох індексів j_1 і j_2 на основі двох окремих частин масиву *S* (S_1 і S_2). Перестановки робляться між відповідними елементами різних масивів $S_1[j_1]$ і $S_2[j_2]$. Автори стверджують, що така організація алгоритмів KSA і PRNG, збільшує число можливих внутрішніх станів в алгоритмі шифрування і усуває загрозу від реалізацій великої кількості атак.

В роботі [9] автори запропонували свій удосконалений алгоритм під назвою RC4A. Вони залишили алгоритм планування ключів KSA без змін і зосередили свої зусилля на процесі генерації шифруючої гами *G*. Щоб забезпечити зменшення зміщення у перших двох вихідних байтах гами і відсутність кореляції між ними та внутрішніми станами PRNG. Для цього передбачається використовувати два масиви S_1 і S_2 розміром 2^n, кожний з яких заповнюється на етапі KSA зі своїм особистим ключем K_1 і K_2, відповідно. Далі, на основі цих двох масивів формується пара вихідних байтів гами *G*. Індекс $i = 0$ для масивів S_1 і S_2 є загальним, а індекс *j* для кожного масиву свій (j_1 і j_2, відповідно). Формування вихідних символів відбувається за правилом:

```
Initialization:
i = 0; j1 = j2 = 0

while (i < L)
```

```
{
  i = i + 1
  j1 = j1 + S1[i]
  t = S1[i]; S1[i] = S1[j1]; S1[j1] = t       // Перестановка
  g1 = S2[S1[i] + S1[j1]]
  j2 = j2 + S2[i]
  t = S2[i]; S2[i] = S2[j2]; S2[j2] = t       // Перестановка
  g2 = S1[S2[i] + S2[j2]
}
```

З наведеного алгоритму видно, що символи гами g_1 і g_2 вибираються із суміжного масиву, на основі двох чисел із власного масиву. Автори наводять обґрунтування того факту, що такий спосіб формування шифруючої гами G зменшує зміщення у перших двох її вихідних байтах g_1 і g_2, та робить шифр більш стійким до більшості відомих типів атак.

З наведених джерел видно, що, через свою простоту, шифр RC4 має декілька слабких місць, через що розробники сучасного програмного забезпечення поступово міняють його на більш стійкі сучасні шифри. Однак, він все ще викликає серйозний інтерес дослідників. З'являються роботи, де пропонуються нові способи усунення нерівномірності у розподілення перших байтів шифруючої гами та їх кореляції з секретним ключем і внутрішніми станами шифру. Одночасно ставиться задача зберегти його високу швидкодію, простоту і базову структуру.

Мета і задачі дослідження

Зважаючи на те, що основною ціллю мережного криптографічного захисту є забезпечення високого рівню захисту конфіденційності даних, що передаються і обробляються в розподілених комп'ютерних мережах, ціллю роботи є аналіз існуючих варіантів підвищення криптографічної стійкості алгоритму потокового шифрування RC4.

Адаптація шифру RC4 до сучасних 32/64 бітних процесорів

Генератор шифру RC4 і більшість розглянутих вище варіантів його модифікації, передбачають отримання кожного наступного символу шифруючої гами G на основі перестановок у складі вектора ініціалізації IV, що уявляє собою масив S, заповнений 8-бітними числами від 0 до 255. Намагання підвищення його стійкості у більшості випадків пропонується виконувати за рахунок ускладнення алгоритму, закладеного у PRNG. Але таке ускладнення позбавляє цей шифр його основних переваг і, тому, варто було б підвищувати його стійкість за рахунок збільшення розміру слова з 8 до 16 або 32 біт. Це значно збільшило б число внутрішніх станів шифру. У разі 16-бітних слів ця проблема для сучасних 32/64-бітних процесорів не уявляє проблеми, а от у разі 32-бітних слів, розмір масиву стає вже не прийнятним.

В роботі [10] автори запропонували 32-бітний варіант генератора шифруючої гами для шифру RC4 з прийнятними вимогами до пам'яті. Щоб обійти проблему із занадто великим масивом для зберігання вектора ініціалізації IV, який містить $M = 2^{32}$ 32-бітних слів a_i, автори залишили його розмір рівним $N = 2^8$, а розмір слова зробили 32-бітним.

На відміну від базового алгоритму RC4, на етапі планування ключів KSA, вхідними параметрами, крім ключа K розміром від 40 до 256 байт, є набір випадкових чисел $\{a_i\}$ розміром N для початкового заповнення масиву S. В такому варіанті, в цьому масиві не буде всіх можливих варіантів 32-бітних комбінацій тому, що для них просто не вистачить місця. Це означає, що поточний стан в масиві S, не є перестановкою всіх можливих 32-бітний чисел.

Щоб у процесі формування вихідних слів шифруючої гами приймали участь усі рівномірно розподілені 32-бітні комбінації, запропоновано наступний варіант алгоритм планування ключів KSA.

```
Input:  Secret  Key K[l],       0 ≤ j < l    // Вхід
        Initial Values  {ai},   0 ≤ i < N
Output: Internal State S{ai}                 // Вихід

j = 0;
```

```
for(int i = 0; i < N - 1; i++) {              // Скремблювання
  j = (j + S[i] + K[i % l]) % N;
  t = S[i]; S[i] = S[j]; S[j] = t;            // Перестановка
  S[i] = S[i] + S[j] % M;                     // Оновлення елементу S[i] у масиві S
}
```

На відміну від базового алгоритму KSA, після виконання кожної перестановки відбувається заміна елементу $S[i]$ у масиві S на результат складання елементів $S[i]$ і $S[j]$ по модулю 2^m. Саме цей спосіб оновлення стану і є основною відміною між алгоритмами RC4 і RC4(n, m).

Так само, новому варіанті алгоритму PRGA, після виконання кожної поточної перестановки, відбувається оновлення поточного стану в масиві S після формування поточного слова шифруючої гами:

```
Input:  S{ai}                 0 ≤ i < N  // Вхід
i = 0; j = 0;
while (i < L)
{
  i = (i + 1) % N;
  j = (j + S[i]) % N;
  t = S[i]; S[i] = S[j]; S[j] = t;        // Перестановка
  t = (S[i] + S[j] % m) % n;
  g = S[t];                               // Вихідний елемент шифруючої гами G
  S[t] = S[i] + S[j] % m;                 // Оновлення елементу S[i] у масиві S
}
```

Тут в процесі генерації шифруючої гами G довжиною L, значення індексу слова i, обраного для оновлення в масиві S, співпадає з індексом слова, з якого отримано вихідне слово g. Це необхідно для того, щоб створене вихідне слово гами g було виведене з масиву S і на його місце було записано нове випадкове число, що, знову ж таки, пояснюється тим, що масив S не є повною перестановкою всіх можливих слів розміром m, а складає лише малу її долю. Така структура алгоритму PRGA забезпечує участь усіх рівномірно розподілених 32-бітних комбінацій у формуванні вихідних слів шифруючої гами G.

Розмір внутрішнього стану при такому способі шифрування дорівнює $(2^n m) + 2n + m$ біт. Ця величина складається з масиву S, де зберігаються 2^n m-бітних слів, m-бітного слова з ключа K і двох n-бітних індексів i та j.

Загальна схема алгоритму RC4(n, m) наведена на рисунку 2.

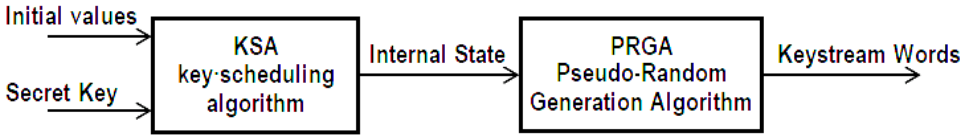

Рис. 2. Схема алгоритму RC4(n, m)

Оскільки параметри n і m є змінними, автори назвали свій шифр RC4(n, m). Пізніше йому дали назву NGG за ініціалам розробників (Y. Nawaz, K.C. Gupta і G. Gong) [11].

Проведені детальні дослідження шифру NGG [12] показали, що статистичне зміщення на етапі планування ключів, дозволяє крптоаналітику супротивника відновити секретний ключ на основі перших кількох кілобайт шифруючої гами і, тому, автори цього дослідження запропонували свій варіант удосконаленого RC4(n, m) і назвали його GGHN (G. Gong, K.C. Gupta, M. Hell, і Y. Nawaz). В новому варіанті KSA було введено додаткову m-бітну змінну k, яка залежить від ключа і маскує величини, що зберігаються в масиві S.

```
Input:  Secret  Key K[l],       0 ≤ j < l    // Вхід
        Initial Values  {ai},   0 ≤ i < N
Output: Internal State S{ai}                 // Вихід
        Variable k;
```

© Щербина Ю. В., Казакова Н. Ф., Логінова Н. І., Фразе-Фразенко О. О., Фоміна Н. Б. (2025). Аналіз способів удосконалення алгоритму потокового шифрування RC4. Сучасний захист інформації, 1(61), 23–30. https://doi.org/10.31673/2409-7292.2025.011455.

```
for (i = 0; i < N – 1; i++)
S[i] = ai;
j = k = 0;
for (i = 0; i < N – 1; i++) {                  // Скремблювання
  j = (j + S[i] + K[i % l]) % M;
  t = S[i]; S[i] = S[j]; S[j] = t;             // Перестановка
  S[i] = S[i] + S[j] % M;
  k = k + S[i] % M;
}
```

В окремих модифікаціях цикл скремблювання **for** пропонується повторювати до 20 разів.

На відміну від алгоритму NGG, в алгоритмі GGHN генератор PRGA використовує псевдовипадковий лічильник *k* як під час оновлення масиву *S*, так і при виведенні *g*:

```
Initialization:
i = 0; j = 0;

while (i < L)
{
  i = (i + 1) % N;
  j = (j + S[i]) % N;
  k = (k + S[j]) % M;
  S[(S[i] + S[j]) % N] = k + S[i] % M;
  g = (S[(S[i] + S[j]) % N] + k) % M;
}
```

Тут замість того, щоб міняти елементи *S* місцями, оновлення виконується заміною поточного елемента елементом, індексованим псевдовипадковим числом.

Кількість способів розміщення 2^m елементів в масиві розміром *N*, складає $(2^m)^N$. Враховуючи, що в алгоритмі GGHN використовується *m*-бітна змінна *k*, яку можна вважати додатковим операндом, кількість можливих внутрішніх станів слід збільшити на одиницю. Тоді загальна кількість внутрішніх станів буде складати $N^2 \times (2^m)^{N+1}$. Наприклад, для шифру GGHN(8, 32), ця величина дорівнює 8240 біт.

Загальна схема алгоритму GGHN наведена на рисунку 3.

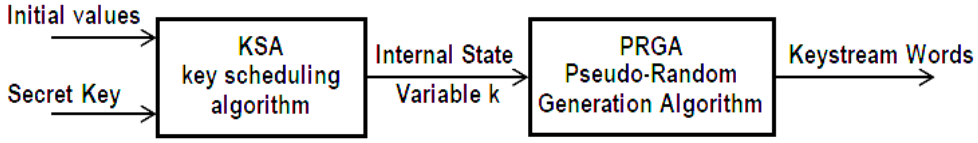

Рис. 3. Схема алгоритму GGHN

Проведені авторами алгоритму шифрування GGHN(8, 32) випробування показали, що за рахунок збільшення довжини слова до 32 біт запропонований генератор ключового потоку PRGA має значно більшу кількість внутрішніх станів і працює у 3,1 рази швидше ніж 8-бітний шифр RC4 на 32-бітній машині. Крім того, при такій конструкції PRGA, на основі поточного внутрішнього стану, не можливо відновити його попередній стан при відсутності ключового потоку *G*. Ні одна з атак, орієнтованих на оригінальний шифр RC4, не може бути використана для нападу на RC4(*n*, *m*).

У запропонованому варіанті RC4(*n*, *m*), проблемою залишається низька швидкість алгоритму планування ключів KSA. Перш ніж використовувати масив *S*, кількість його оновлень має бути достатньою для забезпечення високої випадковості чисел, що в ньому зберігаються.

Висновки

Через наявність слабких місць в алгоритмі шифрування RC4, одним з яких є кореляція між секретним ключем і, сформованою на його основі, шифруючою гамою, розробники сучасного програмного забезпечення, дедалі більше, відмовляються від його використання.

© Щербина Ю. В., Казакова Н. Ф., Логінова Н. І., Фразе-Фразенко О. О., Фоміна Н. Б. (2025). Аналіз способів удосконалення алгоритму потокового шифрування RC4. Сучасний захист інформації, 1(61), 23–30. https://doi.org/10.31673/2409-7292.2025.011455.

Основною причиною є велика кількість робіт, присвячених організації вдалих комп'ютерних атак на протоколи, що використовують цей шифр. Однак, намагання його удосконалити, приводить лише до створення нових потокових шифрів, що за своєю складністю та вибагливістю до обчислювальних ресурсів, мало відрізняються від блокових шифрів. Саме через це, шифр RC4 до цих пір викликає великий інтерес серед фахівців. На основі проведеного аналізу опублікованих різними авторами удосконалень базового алгоритму RC4, можна зробити висновок, що основною причиною наявності в ньому слабких місць є те, що за задумом автора, в ньому виконується тільки операція перестановки, і це не забезпечує необхідного розсіювання статистичних залежностей між символами секретного ключа і початковою частиною вихідної шифруючої гами. Можливим рішення проблеми могло б бути використання економічних нелінійних перетворень.

Перелік посилань

1. S. Fluhrer, I. Mantin, and A. Shamir. Weaknesses in the Key Scheduling Algorithm of RC4. SAC 2001 , vol. 2259 of LNCS, pp. 1-24, Springer-Verlag, 2001. URL: https://link.springer.com/content/pdf/10.1007/3-540-45537-X_1.pdf.

2. Bruce Schneier. Applied Cryptography (Second edition). Wiley, 1995. 662 p. URL: https://github.com/mhpanchal/Cyber-Security-Books/blob/master/Applied%20Cryptography%20(Bruce%20Schneier).pdf.

3. Vladimir Rozic, Bohan Yang, Wim Dehaene, Ingrid Verbauwhede. Iterating Von Neumann's post-processing under hardware constraints. Conference: 2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). DOI:10.1109/HST.2016.7495553. May 2016. URL : https://www.researchgate.net/publication/304456979_Iterating_Von_Neumann's_post-processing_under_hardware_constraints/

4. S. Mister and S. Tavares. Cryptanalysis of RC4-like Ciphers. SAC '98, vol. 1556 of LNCS, pp. 131-143, Springer-Verlag, 1999. URL : https://www.researchgate.net/publication/221274797_Cryptanalysis_of_RC4-like_Ciphers

5. Poonam Jindal, Brahmjit Singh. RC4 Encryption-A Literature Survey. Electronics and Communication Engineering Department, National Institute of Technology, Kurukshetra 136119, India. Procedia Computer Science 46 (2015) 697 – 705. URL : https://core.ac.uk/download/pdf/82455735.pdf.

6. Goutam Paul, Subhamoy Maitra, Anupam Chattopadhyay. Quad-RC4: Merging Four RC4 States towards a 32-bit Stream Cipher. IACR Cryptology ePrint Archive, January 2013, 572. URL : https://eprint.iacr.org/2013/572.pdf.

7. Subhamoy Maitra, Goutam Paul. Analysis of RC4 and Proposal of Additional Layers for Better Security Margin. Procedia Computer Science. Volume 46, 2015, Pages 697-705. DOI:10.1016/J.PROCS.2015.02.129. URL : https://eprint.iacr.org/2008/396.pdf.

8. Maytham Hammood, K. Yoshigoe, Ali M Sagheer. RC4-2S: RC4 stream cipher with two state tables. DOI:10.1007/978-94-007-6996-0-2. 11 April 2016.URL : https://www.researchgate.net/publication/283429259_RC4-2S_RC4_stream_cipher_with_two_state_tables.

9. Souradyuti Paul, Bart Preneel. A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher. Fast Software Encryption. Conference paper. pp 245–259. 2004. DOI:10.1007/978-3-540-25937-4_16. URL : https://iacr.org/archive/fse2004/30170244/30170244.pdf.

10. Yassir Nawaz, K. Gupta, G. Gong. A 32-bit RC4-like Keystream Generator. Information Security and Cryptology. First SKLOIS Conference, CISC 2005, Beijing, China, December 15-17, 2005, Proceedings. URL : https://eprint.iacr.org/2005/175.pdf.

11. Aleksandar Kircanski,· Rabeah Al-Zaidy ,·Amr M. Youssef. A new distinguishing and key recovery attack on NGG stream cipher. Cryptogr. Commun. (2009) 1:269–282. DOI 10.1007/s12095-009-0012-4. URL : https://users.encs.concordia.ca/~youssef/Publications/Papers/A%20New%20Distinguishing%20and%20Key%20Recovery%20Attack%20on%20NGG%20stream%20cipher.pdf.

12. G. Gong, K.C. Gupta, M. Hell, Y. Nawaz, Towards a General RC4-like Keystream Generator. Information Security and Cryptology. Conference paper. pp 162–174. SpringerVerlag, 2005, pp. 162–174. URL : https://the-eye.eu/public/Site-Dumps/campdivision.com/camp/Text%20Files/PDF/Computers%20General/Privacy/Cryptography/RC4%20Stream%20Cipher/Towards%20a%20General%20RC4-like%20Keystream%20Generator.pdf.

13. Aleksandar Kircanski, Amr M. Youssef. On the Weak State in GGHN-like Ciphers. Conference: Availability, Reliability and Security (ARES), 2012. DOI:10.1109/ARES.2012.32. URL : https://users. encs. concordia.ca/~youssef/Publications/Papers/A%20New%20Distinguishing%20and%20Key%20Recovery%20Attack%20on%20NGG%20stream%20cipher.pdf.

Надійшла 15.01.2025