УДК 681.3

Л.Я. Глинчук

ОЦІНКА ЕФЕКТИВНОСТІ АЛГОРИТМІВ КРИПТОГРАФІЧНОГО СТИСНЕННЯ ПОБУДОВАНИХ НА ОСНОВІ ДЕРЕВА ШТЕРНА – БРОКО

Розглядаються кількісні показники оцінки часової та ємнісної складності алгоритмів шифрування, розшифрування, які побудовані як алгоритми криптографічного стиснення на основі дерева (системи числення) Штерна – Броко.

Вступ

Задачу криптографічного стиснення можна розв'язувати за допомогою різних підходів, різних алгоритмів. Постає питання, який з алгоритмів кращий, ефективніший? Для відповіді на нього використовується спеціальний математичний апарат аналізу алгоритмів на предмет їх складності. Під складністю алгоритму розумітимемо часову оцінку роботи алгоритму або об'єм необхідної пам'яті для його виконання за допомогою деякої абстрактної обчислювальної машини. Визначення цих характеристик називатимемо апріорним аналізом алгоритму. При апріорному аналізі алгоритму ігнорують залежності від комп'ютера (обчислення здійснюють на РАМ-машині), мови запису алгоритму (мови програмування) і зосереджуються на визначенні або типу функції, яка характеризує час виконання алгоритму, або типу функції, що характеризує розмір потрібної пам'яті для виконання алгоритму.

Слід зазначити, що окрім фази апріорного аналізу алгоритму важливе місце займає фаза тестування, яка дає змогу за допомогою експериментально отриманих статистик про час виконання алгоритму або необхідну пам'ять на різних даних визначити найтиповіші, найкращі та найгірші оцінки для різних випадків застосування алгоритму.

Отже, час, необхідний алгоритму для вирішення проблеми, який є деякою функцією від розміру вхідних даних (розміру проблеми, задачі), називають часовою складністю алгоритму. Граничну поведінку росту складності, залежно від розміру, називають асимптотичною часовою складністю. Аналогічно можна визначити складність алгоритму щодо пам'яті (ємнісна складність) та асимптотичну складність (асимптотична ємнісна складність) алгоритму щодо пам'яті.

Зрозуміло, що навіть для сучасних комп'ютерів з величезною пам'яттю і швидкодією такі характеристики алгоритмів є і будуть актуальними [1].

У роботі пропонуються і досліджуються алгоритми шифрування та дешифрування (з одночасним стисненням) побудовані на основі системи числення Штерна – Броко, які використовувались при побудові спеціалізованих експертних діагностичних систем.

Алгоритм шифрування

Часом роботи алгоритму називатимемо число елементарних кроків, які він виконує. Будемо вважати, що крок алгоритму (алгоритм записаний покроково) включає фіксоване число операцій (якщо тільки це не опис деякої складної операції – типу "відсортувати всі точки за x координатою") [2].

Оцінимо часову і ємнісну складність алгоритму шифрування [3], який зводиться до наступної послідовності кроків:

1) вхідній послідовності символів (під символами розуміємо всі можливі символи, і цифри, і букви, і ін.) A поставити у відповідність числа за якимось законом, правилом. Обов'язкова умова: вхідна послідовність повинна бути рівна 2^n байт. Кількість рівнів перетворення буде n;

2) кількість кроків $k\_krok = n$; кількість елементів вхідної послідовності $k\_el = 2^n$;

3) $i := 1$, пару сусідніх чисел A_i, A_{i+1} розглядати як дріб $\frac{A_i}{A_{i+1}}$ (або підібрати випадковим чином чисельник і знаменник, отримаємо новий дріб $\frac{A_i^{'}}{A_{i+1}^{'}}$);

4) $j := 1$, від даних чисел відняти отримані, тобто $KL_{j,i} = A_i - A_i^{'}$, $KL_{j,i+1} = A_i - A_{i+1}^{'}$ (де $j = \overline{1,n}$, $i = \overline{1,2^n}$) і занести в матрицю *KL*, яка і буде матрицею-ключем (матриця *KL* має 2^n стовпців і n рядків, але вона не буде повністю заповнена: кожен її рядок буде містити у 2 рази менше елементів від попереднього, решту елементи – нулі);

5) парі отриманих з п. 4 чисел (дробу) поставити у відповідність двійкову послідовність *Dv* за деревом Штерна – Броко;

6) двійкову послідовність *Dv* перетворити у десяткове число *Des* за алгоритмом переведення з двійкової системи числення у десяткову;

7) десяткове число *Des* записати у проміжний масив, оскільки воно переходить у інший рівень;

8) $i := i + 2$;

9) виконати п.п. 3-8 $k\_el = \frac{k\_el}{2}$ разів, тому що кожен раз береться по 2 елементи;

10) десяткові числа, що були записані у проміжний масив у п. 7 переходять як вхідна послідовність до 3;

11) $j := j + 1$;

12) виконувати пп. 3–11 *k_krok* разів;

13) після виконання пп. 1–10, отримаємо *REZULTAT* = одне число і матрицю-ключ *KL*;

14) кінець алгоритму.

Оскільки, розроблений алгоритм містить 13 пунктів (14 – кінець алгоритму), то будемо іти по порядку, можливо деякі пункти будемо об'єднувати.

Складність алгоритмів

На першому кроці вхідній послідовності символів, за деяким правилом ставляться у відповідність числа. Оскільки, вхідна послідовність містить 2^n байт, тобто 2^n елементів-символів (вважаємо, що один елемент займає 1 байт), то кількість вхідних даних до алгоритму рівна $k = 2^n$. Нехай c_1 – кількість операцій, які необхідно виконати для того, щоб поставити у відповідність k елементам – числа. Тоді для виконання першого кроку необхідно виконати kc_1 операцій.

На другому кроці виконується звичайне переприсвоювання. На третьому кроці для кожної пари сусідніх чисел, якщо потрібно (але, якщо не потрібно – ці числа все одно порівнюються, тобто для цього теж виконуються операції), підбираються нові, нехай тоді для підбору k елементів необхідно виконати c_2 операцій. Тоді kc_2 операцій виконуються на третьому кроці. На четвертому кроці шукається різниця між вхідними числами і числами, що отримуються у п. 3. Ці числа записуються у матрицю-ключ. Нехай для цього необхідно c_3 операцій. Оскільки перстворення виконуються знову з усіма k елементами, то маємо kc_3 операцій. На наступному кроці – парі отриманих на кроці 4 чисел ставиться у відповідність двійкова послідовність. Нехай для цього необхідно виконати c_4 операцій. Тоді загальна кількість операцій на цьому кроці є kc_4. Отримані з попереднього пункту двійкові послідовності (їх є $\frac{k}{2}$, тому що парі чисел ставилась у відповідність одна двійкова послідовність) перетворюються у десяткове число за алгоритмом переведення з двійкової системи числення у десяткову. На цьому кроці розглядаються $\frac{k}{2}$ двійкові послідовності і вони дають у результаті $\frac{k}{2}$ десяткових числа. Нехай для такого перетворення необхідно виконати c_5 операцій. Тоді отримаємо для цього кроку $\frac{k}{2}c_5$ операцій. Пункти 3–11 будуть повторюватися стільки разів, скільки у нас є кроків, тобто n разів, але на кожному

кроці в кожному з пунктів буде розглядатися у 2 рази менше елементів.

Отже, для кроку 1 отримаємо:

$$kc_2 + kc_3 + kc_4 + \frac{k}{2}c_5$$

операцій, для кроку 2:

$$\frac{k}{2}c_2 + \frac{k}{2}c_3 + \frac{k}{2}c_4 + \frac{k}{4}c_5$$

операцій, для кроку 3:

$$\frac{k}{4}c_2 + \frac{k}{4}c_3 + \frac{k}{4}c_4 + \frac{k}{8}c_5$$

операцій і так далі.

Об'єднуючи операції для n кроків отримаємо:

$$T_1(k) = \sum_{i=1}^{n}\left(\frac{k}{2^{i-1}}c_2 + \frac{k}{2^{i-1}}c_3 + \frac{k}{2^{i-1}}c_4 + \frac{k}{2^i}c_5\right) =$$

$$= \sum_{i=1}^{n}\left((c_2 + c_3 + c_4)\frac{k}{2^{i-1}} + c_5\frac{k}{2^i}\right) =$$

$$= k * \sum_{i=1}^{n}\left(\frac{c_2 + c_3 + c_4}{2^{i-1}} + \frac{c_5}{2^i}\right) =$$

$$= k * \left((c_2 + c_3 + c_4)\sum_{i=1}^{n}\frac{1}{2^{i-1}} + c_5\sum_{i=1}^{n}\frac{1}{2^i}\right).$$

Розглянемо суму

$$\sum_{i=1}^{n}\frac{1}{2^i} = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + ... + \frac{1}{n}.$$

При $n \to \infty$, ця сума ніколи не буде більше 1, оскільки утворює спадний ряд. Отже, маємо $\sum_{i=1}^{n}\frac{1}{2^i} \le 1$. Аналогічно для суми

$$\sum_{i=1}^{n}\frac{1}{2^{i-1}} = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + ... + \frac{1}{n} \le 2.$$

Після оцінок сум отримаємо

$$T_1(k) \le k(2(c_2 + c_3 + c_4) + c_5).$$

Залишається додати до отриманого результату кількість операцій, що виконуються на першому кроці, отже, остаточно отримаємо:

$$T(k) = kc_1 + T_1(k) =$$
$$= kc_1 + k(2(c_2 + c_3 + c_4) + c_5) =$$
$$= k(c_1 + c_5 + 2(c_2 + c_3 + c_4)).$$

Таким чином, функція $T(k)$ є лінійною функцією, яка має вигляд $T(k) = ck$, де c – деяка константа.

Оскільки нас цікавить гранична поведінка часової складності алгоритму і те, що часова відмінність виконання базових операцій алгоритму (додавання, віднімання, множення, ділення, присвоєння) незначна, ми можемо взяти деяке одне середнє константне часове значення для характеристики виконання всіх базових операцій алгоритму.

Твердження. Якщо

$$A(k) = a_m k^m + ... + a_1 k^1 + a_0$$

є поліномом степеня m, то

$$A(k) = O(k^m).$$

Це, як відомо [1], випливає з наступних співвідношень:

$$|A(k)| \le |a_m|k^m + ... + |a_1|k + |a_0| \le$$

$$\le \left(|a_m| + \frac{|a_{m-1}|}{k} + ... + \frac{|a_0|}{k^m}\right)k^m \le (|a_m| + ... + |a_0|)k^m$$

$k \ge 1$.

Поклавши $c = (|a_m| + ... + |a_0|)$ і $k_0 = 1$, отримаємо $A(k) = O(k^m)$.

Тоді за твердженням складність алгоритму криптографічного стиснення є $O(k)$, тобто $T(k) = O(k)$.

Існують випадки, коли має місце $T(k) = o(g(k))$, $T(k) = \Theta(g(k))$, $T(k) \sim g(k)$.

Означення 1. Говорять, що $T(k) = o(g(k))$, якщо $\lim\limits_{k\to\infty}\frac{T(k)}{g(k)} = 0$ [4].

Тому можна записати, що $T(k) = o(k^2)$, оскільки $\lim\limits_{k\to\infty}\frac{ck}{k^2} = c\lim\limits_{k\to\infty}\frac{1}{k} = 0$.

З двох символів "o" та "O" більш точну інформацію дає символ "o", тому що він зазначає, що $T(k)$ росте точно повільніше, ніж $g(k)$, на відміну від символу "O", який зазначає, що $T(k)$ росте може повільніше, а може так само.

Означення 2. $T(k) = \Theta(g(k))$, якщо існують константи c_1, $c_2 > 0$ і k_0: $k > k_0$, $c_1 g(k) < T(k) < c_2 g(k)$. Іншими словами, мають однаковий порядок росту з точністю до константи [4].

Згідно означення 2 можна записати, що $T(k) = \Theta(c^2 k)$, (тобто $ck = \Theta(c^2 k)$). З трьох символів "o", "O" і "Θ" найбільш точно описує функцію "Θ".

Означення 3. Функції T і g у відношенні $T(k) \sim g(k)$, якщо $\lim\limits_{k\to\infty} \dfrac{T(k)}{g(k)} = 1$. Іншими словами, функції $T(k)$ і $g(k)$ мають той самий порядок росту [4].

Згідно означення 3 отримаємо, що $k \sim k+1$, тобто $T(k) \sim g(k+1)$, оскільки $\lim\limits_{k\to\infty} \dfrac{k}{k+1} = \lim\limits_{k\to\infty} \dfrac{1}{1+\dfrac{1}{k}} = 1$. Символ "~" з усіх найбільш точний.

Для досить великих n справедливе відношення

$$O(1) < O(\log k) < O(k) < O(k \log k) < \\ < O(k^2) < O(k^3) < O(2^k).$$

Часові оцінки $O(k)$ і $O(\log k)$ значно менші від інших [1]. Тому можна сказати, що за часовою оцінкою складності розроблений алгоритм дає досить небогатий результат.

Часова складність оцінена так, що не залежить від реалізації. Не потрібно знати ні точного часу виконання різних інструкцій, ні число бітів, що використовуються для представлення різних змінних, ні навіть швидкість процесора. Такий підхід дозволяє оцінити вимоги до часу в залежності від об'єму вхідних даних. В нашому випадку $T(k) = O(k)$ – це означає, що подвоєння вхідних даних подвоїть і час виконання алгоритму.

Зазвичай алгоритми класифікують відповідно до часової або ємнісної складності. Оскільки розглянутий алгоритм має часову оцінку $O(k)$, то він є лінійний [5–7].

Визначимо ємнісну складність алгоритму криптографічного стиснення.

Отже, пам'ять, необхідну алгоритму для вирішення проблеми, яка є деякою функцією від розміру вхідних даних (розміру проблеми, задачі), будемо називати ємнісною складністю алгоритму. Оскільки, вхідна послідовність складається з k елементів, то для її зберігання необхідно k елементів пам'яті. Далі, k елементам вхідної послідовності ставляться у відповідність k чисел, які теж зберігаються, тому потрібно ще k елементів пам'яті. Після підбору нової послідовності, яка матиме теж k чисел, необхідно також k елементів пам'яті. На наступному кроці шукаються різниці між підібраними числами і вхідними. Ці різниці записуються у матрицю розмірністю $k \times n$, де k – кількість стовпців, n – кількість рядків (n – кількість кроків виконання пп. 3–11, а $k = 2^n$). Тому для зберігання цієї матриці необхідно $k \times n$ одиниць пам'яті. Далі, за алгоритмом, беремо по 2 числа і ставимо їм у відповідність двійковий номер за деревом Штерна – Броко. Для зберігання двійкових послідовностей потрібно $\dfrac{k}{2}$ елементів пам'яті. Після цього йде перетворення двійкових послідовностей у десяткові числа – цих чисел буде стільки скільки було двійкових послідовностей, тобто $\dfrac{k}{2}$, тому і для їх зберігання потрібно $\dfrac{k}{2}$ елементів пам'яті. Протягом виконання всього алгоритму дані зменшуються, наприклад, матриця буде більш як на половину порожня, кожен з зарезервованих масивів буде на кожному кроці у 2 рази зменшуватися, але на початку резервується пам'ять для найбільш можливого числа елементів. Тому підсумовуючи вищеописане, додамо ще c одиниць пам'яті, які будуть виділені для проміжних змінних. Отримаємо:

$$S(k) = k + k + k + kn + \frac{k}{2} + \frac{k}{2} + c =$$

$$= k\left(1 + 1 + 1 + n + \frac{1}{2} + \frac{1}{2}\right) + c = k(4 + n) + c.$$

Нехай $a = 4 + n$. Тоді $S(k) = ak + c$ і ми одержали знову лінійну функцію, аналогічну функції $T(k)$. Тому за твердженням, $S(k) = O(k)$. Оцінки згідно означень 1–3 будуть аналогічними до оцінок, які отримали розглядаючи функцію $T(k)$.

Оцінимо часову і ємнісну складність алгоритму розшифрування-розпаковування. Визначимо спочатку часову складність.

На вхід алгоритму розшифрування-розпаковування подається матриця-ключ і

один елемент – результат (результатом є одне число). Матриця має розмірність $2^n \times n$, але вона майже на половину порожня, наприклад, якщо її розмірність $2^3 \times 3$, то матриця має такий вигляд:

$$KL = \begin{pmatrix} el & el & el & el & el & el & el & el \\ el & el & el & el & & & & \\ el & el & & & & & & \end{pmatrix}.$$

Після підрахунку елементів матриці в загальному випадку отримаємо, що вона містить

$$\left(2^n + \frac{2^n}{2} + \frac{2^n}{4} + ... + \frac{2^n}{2^{n-1}}\right) =$$

$$= 2^n\left(1 + \frac{1}{2} + \frac{1}{4} + ... + \frac{1}{2^{n-1}}\right) = 2^n \sum_{i=1}^{n} \frac{1}{2^{i-1}}$$

елементів. Тоді на вхід до алгоритму подаємо $2^n \sum_{i=1}^{n} \frac{1}{2^{i-1}} + 1$ або $2 * 2^n - 1$ елементів, і позначивши $2^n = k$ остаточно отримаємо $2k-1$. Отже, функція, що відображає залежність вхідних даних від часу запишеться так

$$T_R(2k-1) = T_R(x).$$

У алгоритмі розшифрування-розпаковування слід виконати такі операції:

1) для $k-1$ елемента – перетворити з десяткової у двійкову систему числення, нехай для цього необхідно виконати c_1 операцій, тоді результат $(k-1)c_1$;

2) для $k-1$ елемента – поставити у відповідність дріб у дереві Штерна–Броко, нехай для цього необхідно виконати c_2 операцій, результат $(k-1)c_2$;

3) для $2k-2$ елементів – зробити додавання самого елемента і елемента з матриці-ключа, нехай для цього потрібно виконати c_3 операцій, результат $(2k-2)c_3$;

4) для k елементів зробити перетворення, згідно якого одержимо вхідну послідовність у алгоритмі шифрування, нехай для цього потрібно виконати c_4 операцій, результат kc_4.

Отже сумарна кількість операцій необхідних для розшифрування є:

$$T_R(2k-1) =$$

$$= (k-1)c_1 + (k-1)c_2 + (2k-2)c_3 + kc_4 =$$

$$= (k-1)(c_1 + c_2) + (2k-1-1)c_3 + kc_4 =$$

$$= (2k-1)c_3 - c_3 + (k-1)(c_1 + c_2) + kc_4.$$

Покладемо,

$2k-1 = x$, $kc_4 + (k-1)(c_1 + c_2) - c_3 = b$ і $c_3 = a$.

Тоді

$$T_R(x) = ax + b.$$

Отже, ми отримали лінійну функцію $T_R(x) = O(x)$. Інші оцінки аналогічні до тих, які ми отримали розглядаючи функцію $T(k)$.

Визначимо ємнісну складність алгоритму розшифрування.

Оскільки, на вхід до алгоритму подається $2k-1$ елемент (з вище описаного), то нехай для зберігання вхідних даних необхідно $2k-1$ елемент пам'яті. Кожен раз в алгоритмі використовується проміжний масив (для проміжного результату), спочатку його розмірність рівна 1, але з кожним кроком вона росте у 2 рази, і остаточний розмір масиву – k елементів, нехай для його зберігання потрібно k одиниць пам'яті. Також використовується масив для зберігання вихідних даних, спочатку його розмірність рівна двом елементам, а далі розмірність росте у 2 рази з кожним кроком, поки не стане рівна k, нехай для цього масиву необхідно також k одиниць пам'яті. Підсумовуючи вищеописане, ще додамо c одиниць пам'яті, які будуть виділені для проміжних змінних. Отримаємо

$$S_R(2k-1) = 2k-1+k+k+c =$$

$$= 2k-1+2k-1+1+c = 2(2k-1)+1+c =$$

$$= 2x+1+c = S_R(x).$$

Покладемо, $1+c=b$. Тоді остаточно отримаємо

$$S_R(x) = 2x + b.$$

За твердженням $S_R(x) = O(x)$. Оцінки згідно означень 1–3 будуть аналогічними до оцінок, які отримали розглядаючи функцію $T(k)$.

Висновки

При визначені часової та ємнісної складностей розглядалися загальні алгоритми: криптографічного стиснення та розшифрування-розпаковування, без мож-

ливих допоміжних алгоритмів. В результаті, отримали лінійні функції у всіх випадках з оцінкою $O(x)$ Цей результат говорить про те, що такі алгоритми доцільно використовувати на практиці разом з уже відомими алгоритмами шифрування.

1. *Інтернет-освіта*. Основи комп'ютерних алгоритмів. [Електронний ресурс] – Режим доступу: http:// moodle. ukma.kiev.ua/mod/ resource/view.php?id=239
2. *Кормен Т., Лейзер Ч., Ривест Р.* Алгоритмы: построение и анализ. – М.: МЦНМО, 2004. – 863 с.
3. *Глинчук Л.Я.* Алгоритм криптографічного стиснення інформації за допомогою дерева Штерна – Брокко // Проблеми програмування. – 2008. – № 2-3. – С. 575–578.
4. *Китаев А., Шень А., Вялый М.* Классические и квантовые вычисления. – М.: МЦНМО, ЧеРо, 1999. – 192 с.
5. *Шнайер Б.* Прикладная криптография, 2-е издание. Протоколы, алгоритмы и исходные тексты на языке С. – М.: "Триумф", 2001. – 312 с.
6. *Ахо А., Хопкрофт Дж., Ульман Дж.* Построение и анализ вычислительных алгоритмов. – М.: Мир, 1979. – 536 с.
7. *Манин Ю. И.* Вычислимое и невычислимое. – М.: Советское радио, 1980. – 125 с.

Отримано 08.09.2009

Про автора:

Глинчук Людмила Ярославівна,
аспірантка кафедри ІС
факультету кібернетики
Київського національного університету
ім. Т. Шевченка
Lyda5@bigmir.net.