УДК 004.422.833

І.І. Атаманенко, Д.П. Сівцов

ВИКОРИСТАННЯ МОДУЛІВ НА АСЕМБЛЕРІ ДЛЯ ЗБІЛЬШЕННЯ ПРОДУКТИВНОСТІ С / С + + ПРОГРАМ СИМЕТРИЧНОГО ШИФРУВАННЯ

Анотація. У статті обговорюється можливість збільшення продуктивності С/С++ програм для шифрування інформації на основі симетричних криптографічних алгоритмів. Було встановлено, що істотний ефект може бути досягнутий шляхом реалізації частин таких програм на асемблері.

Ключові слова: симетричні криптоалгоритми, асемблер, продуктивність.

Введення. На теперішній час симетричні криптоалгоритми використовуються в основному для забезпечення конфіденційності інформації на локальних комп'ютерах, робочих станціях і серверах мереж (шифрування), генерації кодів перевірки цілісності мережних повідомлень і електронних документів (технологія MAC), віддаленої мережної автентифікації (протокол Kerberos), забезпечення безпеки мережних застосувань і протоколів стеку TCP/IP (PGP, IPsec, SSL/TLS, SET і т.д.). Окрім цього, симетричні криптоалгоритми можуть використовуватися в якості генераторів псевдовипадкових послідовностей для потокового шифрування (DES в режимі OFB, ГОСТ 28147-89 в режимі гамування) для генерації сеансових ключів (стандарт ANSI X9.17 на основі 3-DES), а також для інших цілей [1,2].

У зв'язку з тим, що потоки інформації в сучасних комп'ютерних системах і мережах характеризується великим об'ємом інформації і швидкістю передачі, криптографічне програмне забезпечення та апаратні засоби повинні задовольняти зростаючі потреби не тільки щодо криптостійкості, але і щодо експлуатаційних характеристик.

Постановка задачі. Мета цієї роботи - визначення і експериментальне дослідження способу поліпшення експлуатаційних особливо-

стей реалізації програм симетричних шифрувальних алгоритмів в застосуванні до блокових шифрів.

Основна частина. Однією з важливих експлуатаційних характеристик програмних реалізацій симетричних криптоалгоритмів є продуктивність.

Дійсно, продуктивність є однією з основних причин апаратної реалізації симетричних криптоалгоритмів (криптопроцесори, криптокарти) [3].

З тих же міркувань програмні реалізації алгоритмів симетричного шифрування виконуються на мовах програмування С або С ++.

Наприклад, у [2,4] наведені численні вихідні коди програм блокових і потокових шифрів, створені на мовах С/С++.

Програмна реалізація симетричних криптографічних алгоритмів лише мовами високого рівня С або С++ на основі програмних платформ, наприклад, таких як MS Visual Studio, дозволяє отримати виконуваний код досить високої якості. Але це не гарантія того, що виконуваний код буде оптимальним в плані продуктивності.

Це обумовлено тим, що продуктивність залежить від ряду факторів, які викликані особливостями і характеристиками алгоритму, середовища програмування, використовуваної операційної системи та апаратних засобів.

Розглянемо деякі особливості симетричних криптоалгоритмів в контексті вказаних факторів.

Для визначеності обговоримо тільки алгоритми блокового шифрування / розшифрування (або скорочено - блокові шифри).

Довжина блоків (або внутрішніх субблоків) і ключів шифру (або внутрішніх субключів) зазвичай коливається від 16 до 512 біт, що порівнянне з розміром форматів даних мов програмування С або С++, а головне - з розрядністю регістрів загального призначення сучасних процесорів.

Алгоритми блокових шифрів будуються на використанні переважно таких бітових операцій, які можуть бути достатньо просто реалізовані на регістровому рівні процесора, а саме:

– побітова операція *XOR* над *n*-бітними послідовностями;
– додавання або множення за модулями 2^n або 2^n-1;
– операції простого і циклічного зсувів на 2^n біт;
– обмін вмістом між блоками або між субблоками;

 ISSN 1562-9945

– операції n-бітних підстановок, перестановок, конкатенації;

– деякі операції на кінцевих полях $GF(2n)$;

Примітка: суттєво, що в усіх випадках де n – ціле, яке порівнянне з розрядністю регістрів загального призначення сучасних процесорів.

Тому можна зробити висновок, що для програмування розділів алгоритму блочного шифрування, які потребують таких операцій, асемблер буде найкращим інструментальним засобом. У такому випадку ці розділи будуть виконуватися на рівні машинних команд, а значить, з максимальною швидкістю.

Окрім операцій, зазначених вище, в блокових шифрах також використовуються і такі операції, які проблематично реалізувати тільки на регістровому рівні. Наприклад, це читання даних з масиву цілочисельних констант, операції перетворення на основі S-блоків, і так далі.

Збереження подібних масивів або блоків часто не може бути виконано тільки за підтримки масиву регістрів загального призначення процесора, а тому потребує розміщення їх в пам'яті.

Тим не менше, і в цьому випадку, принаймні, частина програмного коду відповідного розділу алгоритму може бути реалізовано на асемблері.

Таким чином ми можемо припустити, що вірне рішення в напрямку покращення швидкості програмного продукту, яка здійснює алгоритм блокового шифру, полягає в тому, щоб використовувати модулі асемблера разом з вихідним кодом на C/C++. Це твердження буде справедливим не тільки для блочних шифрів, а й для інших застосувань симетричних криптоалгоритмів. Подібне рішення підтримується також тим, що сучасні компілятори мов високого рівня мають, як правило, вбудований асемблер (у тому числі компілятори мов C/C++).

Для практичної реалізації запропонованого рішення, можна застосувати наступні варіанти сумісного використання модулів на асемблері і програмного коду на мові програмування високого рівня [4,5,6].

1. Компіляція об'єктного модулю, який містить одну або декілька процедур обробки даних, виконується окремо від основної програми. Зовнішній об'єктний модуль на асемблері приєднується до ос-

новної програми на етапі компоновки. Асемблерна процедура, оголошена відповідним чином, викликається з будь якого місця основної програми. Переваги:

– можливість використання асемблерних процедур в програмах, написаних на різних мовах і навіть в різних операційних системах;

– незалежність процесу розробки кінцевого продукту програмного забезпечення і процедур на асемблері;

Недолік - необхідність враховувати механізм виклику зовнішніх процедур і передачі параметрів у процедуру, яка викликається.

2. Використання вбудованого асемблера. Переваги:

– для процедури, яка була розроблена в основній програмі не потрібно спеціальних дій для зв'язування цієї процедури з викликаючою програмою.

– порядок передачі параметрів у процедуру, яка викликається, та відновлення стеку забезпечується автоматично.

– більш простіша і швидка відладка програмного продукту в цілому;

– Недоліки:

– певні обмеження, які накладає середовище програмування на виконання асемблерних модулів;

– процедури, розроблені за допомогою вбудованого асемблера, не можна перетворити у зовнішні модулі, які можуть використовуватися окремо.

З метою практичної перевірки запропонованого підходу було обрано шифр Віженера, який був реалізований як C++ програма, яка включала асемблерні модулі. Платформа програмування – MS Visual Studio 2010. Для створення асемблерних процедур використовувався вбудований асемблер (тобто варіант 2, описаний вище). Програма шифрує або розшифровує ASCII-текст з ключем, довжина L якого менша за довжину відкритого тексту. Таким чином, обробка тексту чи криптограми виконується по блокам довжиною L. Для проведення порівняння були розроблені дві версії програмної реалізації шифру Віженера. Перша версія виконана виключно на C++. Друга версія виконана в контексті запропонованого піходу, тобто на основі сумісного використання C++ і асемблера.

 ISSN 1562-9945

На рис.1 наведено вихідний код процедури шифрування за першою версією програми (тобто тільки на С++). Ця процедура разом з процедурою розшифрування визначає функціональність програми в цілому. Ділянка процедури, в якій виконується основна робота - розрахунок символів криптограми, відзначена курсивом:

```
. . . . . .
class crypt
{
public:
    string content;
    string crypt_content;
    string uncrypt_content;
    string key;
    bool doCrypt(string content="", string key=""){
        if(content.empty() && this->content.empty() ||
    key.empty() && this->content.empty()) return false;
        if(!content.empty()) this->content = content; else
content = this->content;
        if(!key.empty()) this->key = key; else key = this->key;
        int a=0;
        string crypt_content;
        crypt_content.clear( );
        for(int i=0; i<content.length( ); i++)
        {
            crypt_content+=content[i]+key[a];
          if(a<key.length( )-1) a++; else a=0;
        }
        this->crypt_content = crypt_content;
        return true;
    }
. . . . . .
```

Рисунок 1 - Лістинг програмного коду процедури шифрування, виконаної виключно на С++

На рис.2 наведено вихідний код процедури шифрування другої версії програми, у якому та ж ділянка реалізована у вигляді асемблерного модулю. Для наочності цю ділянку виділено також курсивом:

```
. . . . . .
int a=0, b=0;
        int keyL;
        char buf[256], bufk[256];

        keyL=key.length();
        b=content.length();
        this->crypt_content.clear();
        // addition of ASCII-plain text with cipher key
        while(content.length()-a>0)
        {
            (content.length()-
a>256)?(b=256):(b=content.length()-a);
            for(int i=0;i<b;i++)
                buf[i]=content[i+a];
            for(int i=0;i<keyL;i++)
                bufk[i]=key[i];
            _asm{
                xor EBX, EBX
                mov BL, 1
                mov ECX, b
                    lea ESI, buf
                lea EDX, bufk
    next_ch:    mov AL, [EDX]
                                    add [ESI], AL
                inc ESI
                inc EDX
                inc EBX
                cmp EBX, keyL
                jg decK
                 decc:dec ECX
                jnz next_ch
                jmp ex
            decK:   sub EDX, keyL
```

```
                    xor EBX, EBX
                    mov BL,1
                    jmp decc
        ex:
            };

            for(int i=0;i<b;i++)
                    this->crypt_content+=buf[i];
                (content.length()-
a>256)?(a+=256):(a+=content.length()-a);
            }
            return true;
        }
```

. . . . . .

Рисунок 2 – Лістинг програмного коду процедури шифрування на мові C++ і на асемблері

Для оцінки і порівняння продуктивності у тому і другому випадку були обрані наступні числові показники:

– значення середнього часу виконання програмного коду, тобто обробки текстового файлу в режимі шифрування;

– кількість тактів процесора, витрачених на виконання даного програмного коду.

Перша версія програми підраховує значення зазначених числових показників використовуючи виключно C++ програмний код, друга – використовуючи блоки асемблера всередині C++ програмного коду.

Відповідний фрагмент вихідного коду програми наведений на рис. 3. Для наочності асемблерні фрагменти виділені курсивом так, як і вище:

```
. . . . . .
start=clock();
long long t1,t2;
long long delta_t;
_asm{
    CPUID
    RDTSC
```

```
        mov DWORD PTR[t1], eax
        mov DWORD PTR[t1 + 4], edx
    }
    crypt.doCrypt(crypt.content, key);
    _asm{
    CPUID
        RDTSC
        mov DWORD PTR[t2], eax
        mov DWORD PTR[t2 + 4], edx
    }
    stop=clock();
    . . . . . .
```

Рисунок 3 – Вихідний код для обчислення значень показників продуктивності з використанням асемблерних команд

Внаслідок симетричності криптоалгоритму Віженера, величини середнього часу і кількості тактів процесора практично однакові як для шифрування так і для розшифрування (і це було підтверджено в ході тестувань). З тих же міркувань не наведено програмного коду процедури розшифрування. Тому ми далі обговорюємо результати експериментів тільки шифруванням.

Наведемо конкретний приклад. Зашифровувався ASCII-текст з загальною кількістю символів 533 498. Довжина ключа шифрування - 8 символів.

Отримані такі результати характеристик продуктивності. Для першої версії програми (що написана виключно на C++) середній час її виконання дорівнює 0,3691 с. Для другої версії програми (написана на C++ з використанням асемблерних модулів) цей час становить 0,2072 с.

Загальна кількість тактів процесора на виконання першої версії програми дорівнює $\approx 1265 \cdot 10^6$, а для другої версії - відповідно $\approx 641 \cdot 10^6$.

Таким чином за допомогою модулів асемблера ми досягли зменшення середнього часу виконання шифрування приблизно на 43%. Крім того загальна кількість циклів процесора зменшилася приблизно на 51%.

ЛІТЕРАТУРА

1. Stallings W., Cryptography and Network Security: Principles and Practice (5th Edition) [Електронний ресурс] / W. Stallings – NY.: Prentice Hall, 2011. – 721 pp. // WORDPRESS.COM – 2012 – Режим доступа: http://mrajacse.files.wordpress.com/2012/01/cryptography-network-security-5th-edition.pdf
2. Шнайер, Б. Прикладна криптографія. Протоколи, алгоритми, вихідні тексти на мові Сі [Текст] / Б. Шнайер – М.: Видавництво Тріумф, 2002. – 816 с.
3. Коркішко Т., Алгоритми та процедури симетричного блокового шифрування [Текст] / Т. Коркішко, А. Мельник, В. Мельник – Львів: Бак, 2003. –168 с.
4. Аграновский А.В., Практическая криптография: алгоритмы и программирование Сі [Текст] / А.В. Аграновский, Р.А. Хади – М.: СОЛОН-Пресс, 2009. – 256 с.
5. Магда Ю.С., Використання асемблера для оптимізації програм на С++ [Текст] / Ю.С. Магда – СПб.: БХВ-Петербург, 2004. – 496 с.
6. Кулаков, В. Програмування на апаратному рівні: спеціальний довідник [Текст] / В. Кулаков – СПб.: Пітер, 2003. – 847 с.
7. Уоррен, Г. Алгоритмічні трюки для програмістів [Текст] / Г. Уоррен – М.: Видавничий дім "Вільямс", 2004. – 288 с.