# 改进加密算法实现的性能*

张 猛, 杨可新, 鞠九滨

(吉林大学 计算机科学与技术系,吉林 长春 130023)
E-mail: jjb@mail.jlu.edu.cn
http://www.jlu.edu.cn

**摘要**: 介绍了在实现加密算法时改进性能的方法,给出若干加密算法以及使用这些方法实现后的效果实测数据.

**关键词**: 加密算法;实现;性能
**中图法分类号**: TP393 **文献标识码**: A

加密算法的快速软件实现是加密学应用的重要问题.加密算法的两个重要指标是加密强度和速度.加密强度是算法的基本性质,是算法抵抗各种攻击的能力.在大多数应用中,在保证一定加密强度的前提下,加密/解密的速度是最重要的性质,尤其是对要求具有较高速度的应用.

加密强度在算法的设计阶段已经确定,算法的不同实现一般不会对加密强度产生影响.而算法的加/解密速度在不同的实现中则可能有很大不同.加/解密速度在根本上由算法本身的设计决定,但在实现中可以使用各种优化技术最大限度地发挥算法速度的潜力.本文研究在算法确定情况下提高算法速度的实现方法,不涉及新算法的设计问题.

在国际上的同类工作中,文献[1]偏重于算法设计阶段的性能优化的考虑,文献[2]从性能上对最新的AES算法进行了评价和比较.在AES算法描述[3,4]中也包含了一些优化算法性能的研究.本文在这些工作的基础上进行了一些改进,对一些加密算法的实现进行优化并给出一些有益的测量结果.

## 1 加密算法的实现环境

提高加密算法实现的性能,必须对算法本身及其实现环境的特性有较好的了解.

### 加密算法的一般计算特征

对称密钥加密算法包括两个部分:密钥扩展和加/解密.密钥扩展是将原始密钥扩展为加/解密函数使用的子密钥.在一般应用中,密钥扩展所占的执行时间比例很小,绝大部分是加/解密.目前大多数块加密算法是循环密码(iterated block cipher),其加/解密过程是一个按照轮数循环执行轮函数(round function)的加密循环.因此,在应用中轮函数是占执行时间比例最大的操作,其效率是影响速度的决定因素.一般算法的轮函数中的操作都是整数运算[5],包括加减乘除、移位、异或、查表等,很多加密算法还使用S-box和P-box等内部表.

* 收稿日期: 1999-09-13; 修改日期: 2000-03-15
基金项目: 国家自然科学基金资助项目(69873018);高等学校博士点基金资助项目(1999018319)
作者简介: 张猛(1974-),男,吉林长春人,博士生,主要研究领域为分布式系统,计算机网络;杨可新(1970-),男,吉林长春人,博士生,主要研究领域为分布式系统,计算机网络;鞠九滨(1935-),男,黑龙江哈尔滨人,教授,博士生导师,主要研究领域为分布式系统,计算机网络.

**当代CPU的特征**

在以Intel Pentium为代表的当代CPU中采用了许多巨型计算机的技术[6],包括指令流水线、超标量等.对于Pentium系列CPU,它拥有两条整数运算流水线,在一定条件下一个时钟周期内可同时执行两条整数运算指令.如果程序设计得当,使相邻指令满足匹配规则(paring ruls),就能并行执行,从而提高速度.在程序设计时,针对加密算法和CPU的特征进行优化会产生更好的效果.

**语言的选择**

与其他高级语言相比,C语言编译生成的可执行码常常具有较高的执行效率,而且在可移植性方面具有极大的优势,在几乎所有的平台上都有ANSI C编译器,并且技术成熟,相互间的兼容性也很好.对于不同的机器只需在很少的某些程序段内分别编写,大部分程序都不必重写.汇编语言具有适合编写高速度程序的特点,但对不同的机器需重新编写程序.Java语言的跨平台能力使得它非常适合某些应用场合,尤其是在瘦客户机的安全通信中.

## 2 提高算法速度的实现原则

以文献[1]的工作为基础,参考文献[2,6~8],结合我们对加密算法的特点和机器特性的分析以及实验,归纳总结出以下提高算法速度的实现原则.

### 2.1 展开加密循环和函数

加密循环是影响算法实现效率的决定因素.展开加密循环就是将加密循环变为多个循环体程序模块,同时也将加密循环中调用的子函数展开.这样不仅减少了循环所生成的条件跳转指令和计算指令以及函数调用的开销,而且将许多变量变为常量.由于Pentium CPU采用了指令流水线、超标量等技术,循环和变量操作数可能会造成指令流水线的阻断和指令预取的作废[6],因此,展开加密循环和函数减少了这些降低速度的因素,从而提高了速度.这种方法的缺点是增大了程序的长度.

### 2.2 在内部循环中避免使用条件跳转指令

在Intel Pentium系列CPU中分支预测的成功率对程序的执行效率有重要的影响[6].条件跳转指令会产生不可预见的指令流,容易引起分支预测的失败,从而浪费大量的指令周期,所以,在加密算法内部循环中应避免使用条件跳转指令.

### 2.3 变量长度与CPU内部寄存器长度相同

对于Pentium等32位CPU上的算法C语言的实现,无论算法描述中的密钥和中间结果等变量是16位还是8位,在程序中都应定义为32位.因为若变量长度与CPU内部寄存器长度不同,将使得变量的存取都需要附加其他指令,因而增加了程序长度和指令周期数.根据实际测算,使用32位中间变量的IDEA算法的实现要比用16位中间变量的实现速度快一倍多.

### 2.4 限制变量的数量

在Pentium CPU中只有7个通用寄存器[6],如果内部加密循环中有太多的变量,它们就不会被存入寄存器中,而是被存入内存.因为内存操作与寄存器操作相比是很浪费时间的,所以应削减变量的数量以减少内存操作.另外,算法中可定义为常量的数据不要定义为变量.

### 2.5 避免使用耗时的指令

在Pentium CPU中,位移数非1的变量左/右移指令、乘法指令、除法指令占用很多指令周期,

而且左/右移等指令不能与其他指令配对形成流水[6],阻断了流水线,限制了并行性.但是,在加密算法中,为了获得安全性,使用了大量的乘法、移位操作,在实现中应尽量避免使用耗时指令实现这些操作.

以上这些原则可应用于不同的语言编程,对于ANSI C语言、汇编语言和Java语言都能起到提高性能的效果.

## 3 实现方法与结果

在加密算法实现中应用加速原则会产生显著的加速效果,下面,我们通过在具体应用中的一些比较来考察其效果.在比较中我们选取最具代表性并且广泛应用的算法,加密模式采用常用的CBC(cipher block chain)模式,测试平台为Intel Celeron 333 CPU/Window98/IE4.0,表中的速度单位是千字节/秒(kbps).测试数据都是10次测试结果的平均值.在每项比较中,相比较的程序只是在相应的优化方法上不同,其余部分是相同的,这样就保证了测试结果只反映相应的优化方法的加速效果.

### 3.1 展开加密循环

表1是几种典型算法使用加密循环和展开加密循环的速度比较,算法用ANSI C语言编写,编译器为Visual C++ 5.0.

**Table 1** Performance of encryption loop and unrolled encryption loop

**表1** 加密循环和展开加密循环的速度比较

| Program method② \ Algorithm① | RC2[5] | RC5[5] | IDEA[5] |
|---|---|---|---|
| Encryption loop③ | 3 260.59 | 16 644.06 | 5 295.84 |
| Unrolled encryption loop④ | 3 323.54 | 22 550.02 | 5 295.84 |
| Improvement⑤(%) | 1.9 | 35.5 | 0 |

①算法,②编程方式,③加密循环,④展开的加密循环,⑤改进.

这种方法不适于Java语言,因为展开加密循环会导致class的长度过大,失去了Java语言的短小、适于网上传输的优点.

### 3.2 字节顺序转换

各种算法都要使用多字节的数据类型.不同机器上对多字节数据的解释可能不同.例如,M68000系列和SUN sparc系列是高字节在先(big-endian),而Intel×86体系是低字节在先(little-endian).各种算法为了能在多种机器上正确运行而不出现错误,都指定了其内部字节顺序,将原始数据在不同机器上都转化为相同的值,以保证结果的一致性.

在算法的运行中,字节顺序转换是一种频繁的操作,每调用一次加/解密函数就需要进行两次字节顺序的转换,密钥扩展也要用到这种操作.它的运行效率对算法的运行速度影响很大.

在大多数加密算法的实现中,使用移位操作实现字节转换,如SSL库SSLeay[7].移位量非1的移位操作属于费时的操作,不能与其他指令配对形成流水线,应该避免使用.所以,在实现中我们根据字节顺序的转换语义,用其他指令重编此操作,完全替换了移位指令.

以SSLeay中的一个字节转换宏c2l为例:

```
#define c2l(c,l)(l=((unsigned long)(*((c)++))),\
                 l|=((unsigned long)(*((c)++)))<<8L,\
                 l|=((unsigned long)(*((c)++)))<<16L,\
```

```
l|=((unsigned long)(*((c)++)))<<24L)
```

它的语义是将32位数据块c按照Little-Endian的方式转换为长整数.

优化后的字节转换宏为

```
/* Big-Endian 类型的机器 */
#define c2l_HF(c,l,tmp) (tmp=(unsigned char *)(&l)+4,\
                *--tmp=*(c)++,\
                *--tmp=*(c)++,\
                *--tmp=*(c)++,\
                *--tmp=*(c)++)
/* Little-Endian 类型的机器 */
#define c2l_LF(c,l) (l=*((unsigned long *)(c))++)
```

我们用新编的字节顺序转换程序替换SSLeay的原有程序,使用Visual C++5.0编译器在相同的编译选项下重新编译,其中3个典型算法的速度比较见表2.

**Table 2** Performance of two byte order conversion methods

**表2** 不同字节转换方式的速度比较

| Algorithm① / Program method② | RC2 | RC5 | IDEA |
|---|---|---|---|
| Byte order conversion mehod of SSLeay③ | 3 030.57 | 16 644.68 | 4957.81 |
| Optimized byte order conversion method④ | 3 236.35 | 22 550.62 | 4 969.55 |
| Improvement⑤(%) | 6.8 | 35.5 | 0.2 |

①算法,②编程方式,③SSLeay原有字节转换,④优化后的字节转换,⑤改进.

由于Java语言运行于Java虚拟机,不具备指针等直接访问内存的手段,因此以上方法不适用于Java语言.

### 3.3 限制变量长度

在IDEA算法和RC2算法的定义中,密钥和中间变量以及操作的长度都是16位.在RC4算法中所有的数据和表都是8位,但若在实现中也按照算法的描述定义变量的长度,则在Intel Pentium等32位机器上将会产生很大的速度损失.按照前述原则,改用32位中间变量编程将会使速度显著地得以提高.表3列出了C语言实现的各种算法由于变量长度的不同而产生的速度差异.

**Table 3** Performance of C implementations with different variable size

**表3** 使用不同变量长度的C语言实现速度比较

| Algorithm① / Program method② | RC2 | RC5 | IDEA |
|---|---|---|---|
| 32 bits variable③ | 3 323.54 | 33 500.83 | 5 295.84 |
| Variable size of algorithm specification④ | 1 784.81 | 11 331.16 | 2 563.76 |
| Improvement⑤(%) | 86.2 | 195.7 | 106.6 |

①算法,②编程方式,③变量长度限制为32位,④变量长度按照算法定义,⑤改进.

表4是Java语言实现的各种算法由于变量长度的不同而产生的速度差异,浏览器为IE4.0.

**Table 4** Performance of different variable size in Java

**表4** 使用不同变量长度的Java语言实现速度比较

| Program method③ \ Algorithm① | | RC4 | (Improvement②(%)) | IDEA | (Improvement(%)) |
|---|---|---|---|---|---|
| 32 bits variable④ | JDK | 150 | (14.5) | 43 | (4.9) |
| | JIT | 6 828 | (8.6) | 2 184 | (15.2) |
| Variable size of algorithm specification⑤ | JDK | 131 | (—) | 41 | (—) |
| | JIT | 6 286 | (—) | 1 896 | (—) |

①算法,②改进,③编程方式,④变量长度限制为32位,⑤变量长度按照算法定义.

## 4 减少条件跳转指令

在IDEA算法中,基本加密运算是异或、模65537乘法和模65536加法,其中模65537乘法是最耗时的操作.以下是一个未优化的C语言程序:

```
unsigned long idea_mul (unsigned long x, unsigned long y)
{
    unsigned long s,b1,low8,high8;
    if (x=0) return 0x10001-y;
    if (y=0) return 0x10001-x;
    s=x * y;
    low8=(s&0xFFFF);
    high8=(s>>16);
    b1=low8-high8;
    if (low8<high8)
        return (0x10001+b1);
    else
        return b1;
}
```

算法中有3个条件跳转指令,而且没有按照各条件分支的执行概率排列次序.在下面的程序中,根据优化原则减少条件跳转指令并且将函数转换为宏:

```
#define idea_mul(c,a,b,m) \
m=(unsigned long)a * b; \
if (m!= 0) \
    { \
    c=(m&0xffff)-(m>>16); \
    c-=((c)>>16); \
    } \
else \
    c=(0x10001-a-b)
```

表5为使用两种乘法程序的IDEA算法的C语言和Java语言实现的速度比较.

**Table 5** Performance of IDEA implementations with different multiplication methods

**表5** 使用不同乘法程序的IDEA算法实现的速度比较

| | | Multiplication function without optimizing① | Optimized multiplication macro② | Improvement③(%) |
|---|---|---|---|---|
| C Language④ | | 2 573 | 5 295 | 98.1 |
| Java Language | JDK | 43 | 53 | 23.3 |
| | JIT | 2 184 | 2 669 | 22.2 |

①未优化的乘法函数,②优化的乘法宏,③改进,④语言.

## 5 结 论

我们针对加密算法以及当代CPU的特征提出的加密算法所实现的优化原则在实践中取得了很好的加速效果.对于各种语言的算法实现在速度上都得以提高.使用这些优化原则的加密算法C语言实现与未优化程序及SSLeay的比较见表6.

**Table 6** Performance of optimized implementation and others

**表6** 优化的实现与其他实现的比较

| Program method② \ Algorithm① | RC4 | (Improvement③(%)) | RC5 | (Improvement(%)) | IDEA | (Improvement(%)) |
|---|---|---|---|---|---|---|
| Optimized④ | 3 323.54 | (—) | 22 550.62 | (—) | 5 295.84 | (—) |
| SSLeay | 3 030.57 | (9.7) | 16 644.68 | (35.5) | 4 957.81 | (6.8) |
| Without optimizing⑤ | 1 736.05 | (91.4) | 14 768.68 | (52.7) | 2 868.88 | (84.6) |

①算法,②编程方式,③改进,④优化,⑤未优化.

上述这些原则是应用于所有算法的一般性原则,而算法和机器类型是多种多样的,所以在算法的实现中应用这些原则时应该具体问题具体分析,才能最大限度地提高算法的速度.

## References:


[1] Schneier, B., Whiting, D. Fast software encryption: designing encryption algorithms for optimal software speed on the Intel Pentium processor. In: Biham, E., ed. Proceedings of the 4th International Workshop, Fast Software Encryption. Haifa: Springer-Verlag, 1997. 242~259.

[2] Schneier, B., Kelsey, J., Whiting, D., *et al*. Performance comparison of the AES submissions. In: Schneier, B., ed. Proceedings of the 2nd AES Candidate Conference. New York: NIST, 1999. http://csrc.nist.gov/encryption/aes/.

[3] Daemen, J., Rijmen, V. AES proposal: Rijndael. http://csrc.nist.gov/encryption/aes/.

[4] Rivest, R., Robshaw, R. J. B., Sindy, R., *et al*. The Rc6 block cipher. http://csrc.nist.gov/encryption/aes/.

[5] Menezes, A., Vanstone, S. Handbook of Applied Cryptography. New York: CRC Press, 1996. 233~282.

[6] Intel. Intel Architecture Optimization Manual. http://www.intel.com.

[7] Young, E. A. SSLeay and SSLApp FAQ. http://www.cryptsoft.com/ssleay.

[8] Cryptix Development Team. http://www.systemics.com/doc/cryptix/.

[9] Rivest, R. L. The RC5 encryption algorithm. http://theory.lcs.mit.edu/~rivest/.


## Improving Performance in Implementing Encryption Algorithms*


ZHANG Meng, YANG Ke-xin, JU Jiu-bin

(*Department of Computer Science, Jilin University, Changchun* 130023, *China*)

E-mail: jjb@mail.jlu.edu.cn

http://www.jlu.edu.cn



**Abstract**: Methods for improving performance in implementing encryption algorithms are described in this paper. Some results of using these methods to implement encryption algorithms are given.



**Key words**: encryption algorithm; implementation; performance

* Received September 13, 1999; accepted March 15, 2000

Supported by the National Natural Science Foundation of China under Grant No. 69873018; the National Research Foundation for the Doctoral Program of Higher Education of China under Grant No. 1999018319