可搬性の高い高速暗号演算ライブラリ

6P－4

桝本 裕幸　宮内 宏
NEC C&C研究所

1　はじめに

公開鍵暗号の登場は暗号鍵配送の問題を一部解決するとともに、デジタル署名という新しい概念を提供して暗号技術の応用の幅を広げたが、その有用性にも関わらず公開鍵暗号の実用化は遅れている。その要因の一つに公開鍵暗号演算の実装の難しさがある。提案された公開鍵暗号 / デジタル署名方式の多くは 10^{150} を越える大きな整数上のべき乗剰余演算 ($y = x^e \bmod n$) を利用しているが、この演算の計算量は大きく、実用に耐える処理速度を低コストで実現することは困難であった。

これまでにもべき乗剰余演算の高速化については、アルゴリズムの改良や専用ハードウェアの開発など、多くの研究がなされてきた。しかしながら、専用ハードウェアの利用はコストの上昇を招き、アルゴリズムの改良も大きな効果はなかった。

それでも他の用途と兼用できるマイクロプロセッサ上などで、公開鍵暗号演算をソフトウェアで実現したいという要望は多く、限られたCPU資源を最大限に発揮するような演算プログラムが求められる。安価なマイクロプロセッサでも、プログラムの実装次第で十分実用的な処理速度を達成できる。しかし、多くのプログラム言語が整数演算を効率良く扱えないために、可搬性の低いアセンブリ言語を利用しなければならず、CPU毎にプログラムを一から作成せねばならなかった。

このような状況を鑑みて本稿では、高速性を追求しつつ、可搬性(移植性)を最大限に考慮した、暗号用整数演算ライブラリのアーキテクチャを提案する。

2　実装上の問題点

本ライブラリの開発にあたって問題となったのは、大きく分けて次の2点である。

2.1　高速性と可搬性

可搬性を高めるには、広く普及しているC言語でプログラムを記述することが好ましい。しかし高速性を達成するにはどうしても一部アセンブラ記述が必要となる。

本ライブラリは、四則演算などの算術演算アルゴリズムとして、筆算に極めて良く似た方法[1]を採用している。多くのマイクロプロセッサが持っている1word×1word→2wordの乗算命令を利用すれば、筆算による乗算アルゴリズムを非常に効率良く処理することができる。

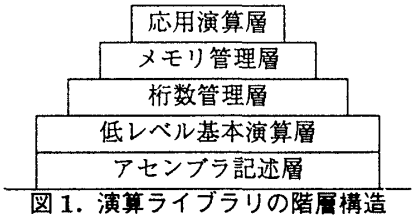

図1. 演算ライブラリの階層構造

ところが、C言語ではこの乗算結果の上位wordを扱うことができないため、同様の結果を得るためには1回で済む乗算命令を4回実行しなければならず、演算効率を大幅に低下させてしまう。したがって、高速性を確保するにはどうしても一部のアセンブラ記述が必要となる。

2.2　用途による機能レベルの違い

桁数やメモリをライブラリ側で管理すればライブラリを利用したプログラムの作成は容易になるが、低コストなCPUでは動的なメモリ管理が困難であったり、コードサイズや処理のオーバヘッドが無視できない場合がある。

ワークステーションやPCなどのメモリに余裕のあるシステムでは、動的メモリ管理により桁数の増減に応じて必要なサイズのメモリを確保すれば、ユーザは整数値の増減を気にせずにプログラムを書くことができ、高度な演算処理も容易に記述することができる。

しかしながら、携帯端末やICカードなどで、小規模で性能の低いプロセッサを利用する場合、メモリ確保 / 開放に伴って発生する不連続領域は乏しいメモリを更に圧迫するため、プログラマが計画的に割り当てた固定領域を利用する方が効率が良い。例えばべき乗剰余演算しか実行しない場合には桁数や符号の管理も必要ない。一般に小規模なシステムでは高度な演算処理は必要としないことが多く、それらの管理を省略すればコードサイズや処理のオーバヘッドを削減できる。

3　アーキテクチャ

以上のようなことから、本ライブラリは関数のインタフェースを複数の層に分割し、アセンブラ利用による高速化を実現するとともに、システムの規模や演算の複雑さに応じて最適な階層を選択できるようにした。この階

High Speed Cryptographic Library with High Portability.
Hiroyuki MASUMOTO, Hiroshi MIYAUCHI.
C&C Research Laboratories, NEC Corporation,
4-1-1 Miyazaki, Miyamae, Kawasaki 216, Japan.

層構造を図1に示し、各階層の概要を説明する。

多倍長整数の値はプロセッサが扱う $k$bit 幅の word 要素の配列として表現される。配列 $x_0, x_1, \ldots, x_{l-1}$ は、整数値 $\sum_{i=0}^{l-1} x_i 2^{k \cdot i}$ を保持する。桁数管理層以上では、この配列へのポインタとともに、配列長 (桁数), 符号, メモリ確保サイズなどを構造体にまとめて処理している。

3.1 アセンブラ記述層

アセンブラ記述層で用意する関数は、1word×1word →2word の乗算、およびその繰り返し処理など、前述の C 言語が効率良く扱えない部分である。

アセンブラ記述部分のソースはおよそ 200 行程度となっており、この部分を CPU に応じて書き換えることにより、その CPU の演算能力を最大限に引き出しながら、本ライブラリを移植することができる。この層の機能を C 言語で記述したバージョンも用意しているが、演算速度は約 1/3 程度に低下する。

一般にユーザが直接この層の関数を用いることはない。

3.2 低レベル基本演算層

この層では配列型の整数値に対する四則演算やべき乗剰余演算、文字列変換などを提供する。配列長や符号、メモリ確保などはユーザが管理する。例えばべき乗剰余演算しか必要としない用途であれば、処理のオーバヘッドが小さく、コードサイズも抑えられるため、携帯機器など廉価なプロセッサで限られた演算処理を行う用途に有用である。この層での関数呼出は配列長を毎回指定する形になり、例えば乗算では以下のようになる。

```
unsigned int mx[64], my[64], mz[128];
  :
zb_mul(mz, mx, my, 64, 64); /* z = x y */
```

3.3 桁数管理層

整数は構造体で扱われ、値の変動に応じて桁数や符号が管理される。ただしメモリ管理は行われないので、必要なサイズのメモリをユーザが予め構造体に確保し、関数の入力領域と出力領域が重なる時などは配慮が必要である。メモリ管理機構は不要であり、ユーザの管理の下で必要最小限のメモリ領域を割り当てることができるため、メモリの乏しいシステムで、ユークリッドの互除法など桁数の変動する演算を行う際に有用である。この層での関数呼出は、例えば乗算では以下のようになる。

```
unsigned int mx[64], my[64], mz[128];
struct Z_INT x, y, z;
x.val=mx;  y.val=my;  z.val=mz;
  :
za_mul(&z, &x, &y); /* z = x y */
```

3.4 メモリ管理層

整数は構造体で扱われ、桁数の変動や入出力領域の重なりに応じて、必要なサイズのメモリを動的に確保 / 開放する。動的なメモリ管理機構が必要だが桁数や符号メモリ確保などをほとんど考慮せずにプログラムを作成できる。この層での関数呼出は、例えば乗算では以下のようになる。

```
struct Z_INT x, y, z;
z_init_va(&x, &y, &z, NULL);
  :
z_mul(&z, &x, &y); /* z = x y */
```

3.5 応用演算層

応用演算層では、下位層が提供する基本演算を利用して、暗号に関連した有用な関数を多種用意するライブラリ上のライブラリである。応用演算層が現在提供する関数の一部を以下に示す。

素数判定, 素数生成, 強い素数の生成
最大公約数, 剰余逆数演算, Jacobi 関数
GF(p) 上の平方根計算

4 性能

本ライブラリを利用した full-512bit べき乗剰余演算の演算時間を表に示す。

R3000	(25MHz)	0.33 秒	
SPARC	(25MHz)	0.87 秒	
80386	(20MHz)	3.8 秒	(32bit 演算命令未使用)
Z80	(8MHz)	180 秒	(e=3 の RSA 暗号化 0.6 秒)

5 おわりに

高速性を追求しながら可搬性を考慮した暗号演算ライブラリのアーキテクチャを提案した。本ライブラリは、全てアセンブラで記述した場合と同等の演算速度を持ちながら、全て C 言語で記述したプログラムに匹敵する可搬性を実現し、廉価なマイクロプロセッサから高性能ワークステーションまで、多様なシステムの上で利用可能なものとなっている。今後、C++ 言語への対応や秘密鍵暗号の組み込みなど、更なる充実をはかりたい。

参考文献

[1] D.E.Knuth, *"The Art of Computer Programming, 2nd Edition, Seminumerical Algorithms Vol.2"*, Addison-Wesley Publishing Company, 1981.

[2] S.R.Dussé, B.S.Kaliski Jr., *"A Cryptographic Library for the Motorola DSP56000"*, EUROCRYPT'90, pp.230-244.