Implementation of Cryptographic Primitives

Shreya Rajkumar[1], Dr. Chester Rebeiro[2]

Student ECE National Institute of Technology,Tiruchirappalli
Tanjore Main Road, National Highway 67,India

Asst.Prof,Dept. of Computer Science and Engineering, Indian Institute of Technology, Madras

Abstract *— The efficiency of cryptographic library depends on the implementation of multi-precision algorithms. In this paper, implementation of algorithms for modular addition, subtraction, comparison, Extended GCD Algorithm, Montgomery multiplication, Montgomery Exponentiation is discussed and the developed library is tested for correctness and analyzed on various platforms. The developed library works for very large numbers .It is also scalable from 8 bit to 64 bit for a wide range of platforms that includes embedded controllers and DSP processors.*

Keywords — *Chinese Remainder Theorem (CRT), Cryptography, Functionality testing, Montgomery Multiplication, Montgomery Exponentiation, Multi-precision Library, RSA Algorithm..*

I. INTRODUCTION

In today's computer-centric world, cryptography is most often associated with scrambling plaintext into cipher text by the process of encryption. Modern cryptography concerns itself with the following objectives:

- Confidentiality
- Integrity
- Non-repudiation
- Authentication.

In order to realize the above mentioned goals, implementation of cryptographic primitives requires integers of extremely large magnitude. This can be realized by using a multi precision library. There are already a large number of multi-precision libraries available. However, one of this proposed library's developed feature is that it is compatible with many systems. It does not use any predefined functions and there is no dynamic memory allocation. The advantages of these are the following :

- Memory is limited in embedded systems.
- Embedded systems can run for years which can cause wastage of memory due to fragmentation.
- Dynamic memory allocation is slow.
- Dynamic memory allocation makes it difficult to debug especially with limited debugging tools that are present on the embedded system.

II. PROCEDURE FOR IMPLEMENTATION OF THE LIBRARY

The multi-precision number is represented as structure *bignumber* which contain words of size specified by word_size. The library is scalable to any magnitude by just giving the word size (word_size) and input size (No_of_bits) as parameters.
The inputs and structure are defined in the following way.
The example here indicates input of 1024 bit and word size of 64 bits

- define No_of_bits 1024
- define word_size 64
- define max_digits (No_of_bits/word_size)
- typedef unsigned long *word;*
- *typedef struct*
  { *word digits*[*max_digits*];
  int sign;
  } *bignumber t;*

III. ALGORITHMS

Multi-Precision Addition

procedure ADD(*word ∗r,word ∗a,word ∗b*) ⊲Inputs : array of words, performs *c= a +b*
2:*carry* ← 0
3. *for i from 0 to n do*
4: *word t* ← *a*[*i*]
5: *t* ← *t +carry*
6: *carry* ← (*t* < *carry*)
7: *word l* ← *t +b*[*i*]
8: *carry* ← *carry +*(*l* < *t*)
9: *r*[*i*] ← *l*
10:*i ++*

Multi-Precision Subtraction

```
1: procedure Sub(word *r,word *a,word  *b)
2: borrow ← 0
3: word temp1 ← a[i]
4: word temp2 ← b[i]
5: r[i] ← (temp1 − temp2 − borrow)
6: if (temp1! = temp2) borrow = (temp1 < temp2)
7: i + +
```

Modular Addition

```
1: procedure ModularAdd(bignumber *r, bignumber
* a, bignumber *b,bignumber *m)
2: ADD(r, a, b)
3: if (COMPARE(r,m) >= 0)
4: SUB(r, r,m)
```

Bignumber Comparison

```
1: procedure COMPARE(bignumber*a,bignumber*b)
2: word  *ap ← (a → digits) ⊲ digits is an array of
                              words in the structure a
3: word  *bp ← (b → digits)
4: for i from n to 0 do
5: if (ap[i]! = bp[i])
6: return ((ap[i] > bp[i])?1 : −1)
7: return 0
```

Modular Subtraction

```
1:    procedure ModularSUB(bignumber * r,
    bignumber * a, bignumber * b, bignumber * m)
2: i f (COMPARE(a, b) < 0)
3: SUB(r, b, a)
4: SUB(r,m, r)
5: else
6: SUB(r, a, b)
```

Montgomery Multiplication:

```
1: procedure MontMul(bignumber *m, bignumber
   *a, bignumber * b) ⊲ Performs (a*b)R−1mod m
2: R ← 0
3: for i from 0 to (n − 1) do
4: ti ← (r0 + aib0)m1 mod b
5: R ← (R + aib + tim)/b
6: If R >= m then R ⊲- R – m
7: return R
```

Montgomery Exponentiation:

```
1:  procedure MontExp(bignumber *m, bignumber
    *a, bignumber *e) ⊲ Performs ae mod m
2: x1 ←MontMul(a, R2 mod m),A←R mod m
3: for i f rom n to 0 do
4: A ←MontMul(A,A)
5: If ei = 1 then A←MontMul(A, x1)
6: A←MontMul(A, 1)
7: return A
```

Extended GCD Algorithm

Given two positive integers x and y, the algorithm returns a,b and v such that $ax + by = v$, where $v = gcd(x, y)$.

```
1: g = 1
2: While x and y are both even : x = x/2, y = y/2, g =
2g
3: u = x, v = y,A = 1, B = 0, C = 0,D = 1
4: for u is even do the following do
5: u = u/2
6:  I f A and B are even, then A = A/2, B = B/2;
otherwise,A = (A + y)/2,
   B =(B − x)/2.
7: for v is even do the f ollowing do
8: v = v/2
9:  I f C and D are even, then C = C/2,D = D/2;
otherwise, C = (C + y)/2,
   D =(D − x)/2.
10: I f u >= v, then u = u − v,A = A − C, B = B − D
11: otherwise, v = v − u, C = C − A, D = D − B.
12: I f u = 0, then a = C, b = D, and return(a, b, g.v);
otherwise go to Step4
```

RSA was implemented along with CRT in order to speed up the calculations during decryption.

IV. IMPLEMENTATION AND RESULTS

Functionality testing :

Python scripts were written in order to test the functionality of the library developed. Python's inbuilt bignum library was used to check the correctness of the developed C library.

Fig. 1. Testing the developed addition algorithm with Python's in-built Multiprecision library.

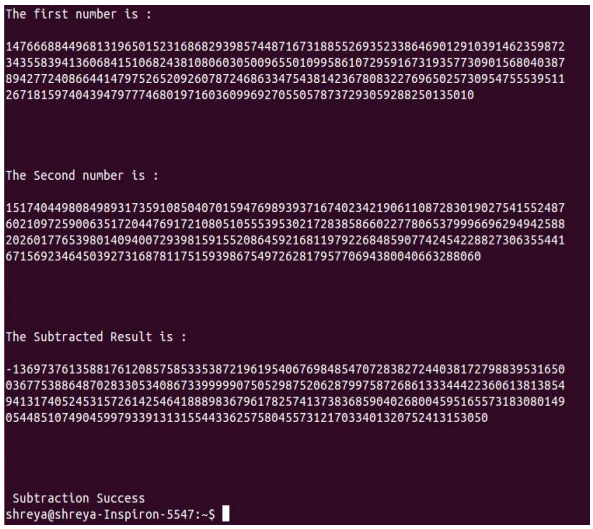

Fig. 2. Testing the developed subtraction algorithm with Python's in built Multiprecision library.

The RSA algorithm implemented using the developed multi-precision library is evaluated on different platforms and the total time taken by the RSA algorithm which includes encryption and decryption is measured in seconds.

Evaluation of RSA on Intel 64-bit System

Input Size	Without CRT	With CRT
128	0.000081	0.000019
256	0.000502	0.000304
512	0.001680	0.010923
1024	0.091220	0.077648

Evaluation of RSA on LPC Xpresso 1347

Input Size	Without CRT	With CRT
128	2.58000	1.224000
256	19.674000	12.958000
512	169.754000	118.123000

We can see that in the above cases, RSA with CRT outperforms RSA without CRT.

V. CONCLUSION

This paper explains implementation details of multi-precision arithmetic library which is key for performing cryptographic operations. The RSA algorithm is implemented and CRT technique is used for faster implementation. The library of multi-precision arithmetic operations can be used in implementing not only RSA but also various cryptographic primitives like ECC, Elgamal etc. The main advantage of the developed library when compared to other available libraries is that it is supported by almost all the systems as neither dynamic memory allocation nor predefined functions were used.

ACKNOWLEDGMENT

I would like to thank IIT Madras for the facilities provided.

I would also like to thank my parents for their support and encouragement.

References

[1] P. C. v. O. Alfred J. Menezes. Handbook of Applied Cryptography. CRC Press, August

[2] Koltuksuz A., Hışıl H. (2005) Crympix: Cryptographic Multiprecision Library. In: Yolum ., Güngör T., Gürgen F., Özturan C. (eds) Computer and Information Sciences - ISCIS 2005. ISCIS 2005. Lecture Notes in Computer Science, vol 3733. Springer, Berlin, Heidelberg.

[3] Koc, C.K.: High Speed RSA Implementation. RSA Laboratories. TR201 (1994)

[4] Bosselaers, A., Govaerts, R., Vandewalle, J.: A Fast and Flexible Software Library for Large Integer Arithmetic. In: Proceedings 15th Symposium on Information Theory in the Benelux, Louvain-la-Neuve (B), pp. 82–89 (1994)