CryptoLib: Cryptography in Software

John B. Lacy [1]
Donald P. Mitchell [2]
William M. Schell [3]

AT&T Bell Laboratories

ABSTRACT

With the capacity of communications channels increasing at the current rates and with the kinds of electronic services becoming more varied and multidimensional, there is also coming a greater tendency to store and forward sensitive, semi-private or very private information. With this tendency there is an increasing interest in protecting the privacy and security of communications channels and sensitive information.

Many protocols have been devised which provide varying levels of security and which have overhead associated with them roughly proportional to the level of that security. These protocols make use of cryptosystems which may be divided roughly into two groups: private key and public key. It is generally assumed that efficient implementations of these systems are possible only in hardware or at least coded in very nonportable assembly language for particular processors. **CryptoLib** is a very portable and efficient library of routines necessary for the aforementioned cryptosystems. It written entirely in C and exists under UNIX™.

1. Introduction

The tools included in **CryptoLib** must be very efficient to facilitate efficient implementation of crypto protocols. Great effort has therefore been put into streamlining our functions and making them run fast (this is an ongoing effort). A list of the tools in **CryptoLib** follows:

Big integer creation and manipulation tools.
Fast Big Arithmetic functions.
Chinese Remainder Theorem speedup of exponentiation.
Quadratic Residues and Modular Square Roots
Random number generators -- pseudo and true.
Fast Prime number generator and Primality test.
Euclid's Extended Greatest Common Divisor algorithm.
DES encryption and decryption tools.
RSA key generation, encryption and decryption tools.
El Gamal key generation, encryption and decryption tools
  and signature generation and verification tools.
NIST key generation and signature generation and verification tools.
NIST Secure Hash Standard (SHS)
MD5 Message Digest tools (RSA, Inc.)

The most interesting aspects of our library are:

(1) It is written entirely in C.

(2) An efficient **big number package** which includes routines for addition, subtraction,

1. lacy@research.att.com
2. don@research.att.com
3. bill@research.att.com

multiplication, modular exponentiation, left and right shifting, division, the modulo operation, comparison and copying.

(3) An efficient implementation of large integer **multiplication** based on a recursive version of the Karatsuba $O(n sup size +2 log3)$ algorithm [Knuth 82]. Karatsuba multiplication is also used at the most elemental level, $32 bit by 32 bit$ multiplication. (Inline assembly versions of this function are also available for some processors).

(4) It includes a very fast implementation of **modular exponentiation** using addition chaining table lookup and Montgomery reduction [Mont 85].

(5) A large speedup in exponentiation is further achieved using the **Chinese Remainder Theorem** when the factorization of the modulus is known.

(6) A true random number generator based on the randomness inherent in operating system interrupts. A pseudo-random number generator based on a feedback shift register initialized by truly random numbers and updated by a DES variant.

(7) An implementation of Marsaglia's subtract with carry pseudorandom number generator [Mars 91] which is combined with DES.

(8) A very fast **strong prime** generator based on Gordon's strong prime method and Rabin's probabilistic primality test.

(9) A very fast implementation of **DES block encryption**. This has a throughput of roughly 0.9 Megabits/second.

The following sections go into the implementation of the unique aspects of **CryptoLib** in greater detail. Tables illustrating timing information of various functions will also be included. There are also sections describing the basic concepts of the crypto systems supported.

2. Implementation

2.1 Bigmath

2.1.1 Big Number Representation and Basic Arithmetic: Numbers of arbitrary length may be represented as

$B ˜=˜sumi(0, n-1, b sub size +2 i * r sup size +2 i)$

For this library r is assumed to be $2 sup 32$.

$Bignums$ have a sign, a length, a space allocation counter and a pointer to the 32 bit chunks which comprise the actual number. Manipulating big numbers reduces to the problem of manipulating large arrays. Efficiency is critical. We provide routines for creating and initializing bignums, for converting character arrays to bignums and for converting bignums to character arrays.

With the exception of multiplication and exponentiation, basic arithmetic functions are implemented in a straightforward manner. We provide addition, subtraction, division (which returns both the quotient and remainder), modulo, right and left shifting, comparison and copy functions.

2.1.2 Multiplication of Bignums: This section will describe recursive Karatsuba multiplication for bignums, efficient $32 bit X 32 bit$ multiplication and squaring speedups. It will also contain a table of timing results for $n sup size +2 2$ and Karatsuba multiplication. The 32 bit multiply instruction is also implemented as an inline assembly function (1 instruction for some processors). Since multiplication is the most critical function in public key cryptosystems (as a result of the importance of modular exponentiation) this latter speedup is crucial. Timing results of this implementation will also be included.

2.1.3 Modular Exponentiation: Exponentiation ($a sup size +2 b ˜mod˜ N$) is done most efficiently using addition-chaining [Knuth 82 and Brickell 89]. Each multiplication or squaring result must be reduced modulo N. We use Montgomery's technique for modular reduction [Mont 85]. This section will describe this implementation in detail and will include a table of times for various size arguments. The basic result is that without assembly language implementation of $32 bit by 32 bit$ multiplication, 512 bit modular exponentiation (a, b, and N all 512 bits) takes 1.3 seconds. With the 32 bit multiply done as inline assembly, the same calculation takes 0.4 seconds.

2.1.4 Extended Euclid Algorithm: Euclid’s extended algorithm for finding the greatest common divisor of two numbers is based on the following equation:

$$aa' - bb' = gcd(a, b)$$

If *a* and *b* are relatively prime, their *gcd* is 1. In this case, if one can satisfy the above equation, then the multiplicative inverse, a', of $a\ (\mathrm{mod}\ b)$ is known (and so is the negative multiplicative inverse, b' of $b\ (\mathrm{mod}\ a)$). Among other places, inverses modulo some number are useful for generating RSA keys. A brief description of the implementation of this function will be included.

2.2 Random Number Generation

We include three random number generators: one which generates truly random numbers and two which generate pseudo-random numbers.

The truly random numbers are generated by an algorithm which makes use of the randomness in system interrupts. A detailed description of this algorithm will be included.

One pseudo-random number generator uses a feedback shift register and the randomness properties of DES. The other uses Marsaglia’s subtract with carry generator combined with DES. These generators will be described in great detail. Tables will be included illustrating the results of applying Knuth’s statistical randomness tests to the output of all of these generators.

2.3 Primes

This section will include a description of two methods of finding strong primes. Both use Rabin’s probabilistic primality test. The average time to generate a 512 bit strong prime (averaged over 100 generated primes) is 24 seconds (with inline assembly 32 bit multiplication). Primes of 1024 bits take 5.1 minutes. More timing results for finding strong primes of various sizes will be included.

2.4 Message Digests

For many cryptosystems a one-way hashing function is needed which will take an arbitrary amount of input and yield a message digest which is impossible to generate with any message other than the one used to create the digest. For this purpose we use the MD5 message digest functions from RSA Data Security, Inc. [MD5 91] and the NIST secure hash standard (SHS) [NIST 92]. We provide a $Bignum$ interface to these functions.

2.5 RSA Tools

We provide the user with a RSA key set generation tool which takes the modulus size and the public exponent size as arguments. The key set contains the public key and the private key. The private key is set up for the Chinese Remainder Theorem speedup. A brief description of RSA will be included as well as a table of times for key generation, encryption and decryption for various sized keys. With the inline assembly 32 bit multiply, RSA decryption can be done at 3.0 Kbits/sec. With Karatsuba 32 bit multiplication, decryption runs at 1.3 Kbits/sec.

2.6 El Gamal Tools

2.6.1 Basic El Gamal: We provide the following tools for El Gamal systems: a key set generation tool which returns a public and private key, message encryption and decryption tools, a signature generation tool and a signature verification tool. This section briefly describes the El Gamal system and includes a table of timing information for various sized moduli.

2.6.2 NIST Proposal for Signature Generation: The National Institute of Standards and Technology (NIST) has proposed a digital signature standard which is a variant of El Gamal and a system proposed by Schnorr [NIST 91].

We provide tools to generate NIST keys and NIST signatures and to verify NIST signatures. A timing table will be included.

2.7 DES Tools

We provide tools for doing basic DES encryption and decryption in electronic codebook, cipher block chaining and output feedback modes using keys generated from 8 character strings. This is an extremely fast version of DES block encryption (˜ 1 Mbit/sec) which makes use of many of the well known speedups discussed in the literature. This implementation is a factor of 2 faster than the Finland implementation. Details will be included.

3. Conclusion

CryptoLib is a very complete, efficient and portable library of tools necessary for implementing cryptosystems and crypto-protocols. Some speed has been sacrificed for portablity. However, it appears that the only critical piece of code that need be programmed in assembly language for large speed benefits is the $32 bit by 32 bit$ multiplication primitive. On many machines this is a single instruction. It is clear that these function can indeed be implemented in portable software.

4. References

[Rivest 78]
R. L. Rivest, A. Shamir, L. Adleman. A Method for Obtaining Digital Signatures and Public Key Cryptosystems. *Communications of the ACM* 21,2 (Feb, 1978), 120-126.

[ElGamal 85]
Taher ElGamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory* IT-31,4 (July, 1985), 469-472.

[Knuth 80]
Donald E. Knuth. *The Art of Computer Programming, Vol 2: Seminumerical Algorithms.* (Addison-Wesley, 1982, 2nd edn.)

[Brickell 89]
Ernest F. Brickell. A Survey of Hardware Implementations of RSA. Viewgraphs from a Technical talk by Brickell at AT&T Bell Laboratories, Murray Hill, NJ, (Sept 12, 1989).

[Mont 85]
Peter L. Montgomery. Modular Multiplication Without Trial Division. *Mathematics of Computation* 44,170 (April, 1985), 519-521.

[Dussé 85]
Stephen R. Dussé *et. al.* A Cryptographic Library for the Motorola DSP56000. *Advances in Cryptology - EUROCRYPT '90* I.B. Damgard ed., Springer Lecture Notes in CS #473, 1991, pp. 230-244.

[Gordon 84]
J. A. Gordon. Strong Primes are Easy to Find. *Proc. Eurocrypt-84*, Paris, April, 1984.

[MD5 91]
RSA Data Security, Inc. MD5 Message-Digest Algorithm. Copyright (C) 1990, RSA Data Security, Inc. All rights reserved. Received via the net.

[Diffie 76]
W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory* IT-22,6 (Nov, 1976), 644-654.

[NIST 91]
National Institute of Standards and Technology (NIST). Digital Signature Standard. *Federal Information Processing Standards Publication XX* National Technical Information Service, U.S. Dept. of Commerce, 1991.

[NIST 99]
National Institute of Standards and Technology (NIST). Secure Hash Standard *Federal Information*

Processing Standards Publication YY National Technical Information Service, U.S. Dept. of Commerce, 1992.

[DES 77]
Data Encryption Standard. *Federal Information Processing Standards Publication 46-1* National Technical Information Service, U.S. Dept. of Commerce, 1977.

[Bong 89]
Dieter Bong and Christoph Ruland. Optimized Software Implementations of the Modular Exponentiation on General Purpose Microprocessors. *Computers and Security* 8,7 (1989), 621-630.

[Mars 91]
George Marsaglia and Arif Zaman. A New Class of Random Number Generators. *The Annals of Applied Probability*, 1,3 (1991), 462-480