# Comparative Analysis of Software Libraries for Public Key Cryptography

Ashraf Abusharekh[1] and Kris Kaj[2]

[1] George Mason University, Fairfax VA 22030, USA,
aabushar@gmu.edu,
[2] George Mason University, Fairfax VA 22030, USA,
kgaj@gmu.edu

**Abstract.** Software implementations of public key cryptosystems require efficient realization of operations on large integers and elements of the Galois Field. Multiple libraries implementing such operations exist both commercially and in the public domain, in this paper, we perform comparison of eight libraries: CLN, CryptoPP, GNU MP, LiDIA, MIRACL, NTL, OpenSSL and PIOLOGIE, using performance and support of public key primitive operations. The performance of all libraries is ranked based on the measurements performed according to a methodology that takes into account the performance and relative use of primitive cryptographic operations. The performance results shows that GNU MP has the best performance for operations on large integers, OpenSSL has the best performance for operations on elliptic curves over prime fields and LiDIA and MIRACL have the best performance for operations on elliptic curves over binary fields. CryptoPP leads in terms of support for cryptographic primitives and schemes, but is the slowest of all investigated libraries.

## 1 Introduction

In order to assure the required level of cryptographic strength, mathematical functions used in public key schemes require operations on large integers of size varying between 768 to 2048 bits, as well as elliptic curve operations over fields with element size in the range of 140 to 240 bits. Software implementation of such arithmetic operations is difficult since currently available processors have a limited word-size up to 64 bit.

Multiple algorithms have been developed to perform these multi-precision arithmetic operations efficiently, and several libraries implementing such algorithms exist both commercially and in public domain. Nevertheless to our best knowledge, no systematic study has been done to compare and contrast these libraries against each other.

In our study, described in this paper eight libraries have been chosen from the public domain to perform the comparison: CLN[3], CryptoPP[5], GNU MP[6], LiDIA[14], MIRACL[20], NTL[17], OpenSSL[18] and PIOLOGIE[7]. The aim of this study is to evaluate the suitability of using the aforementioned software

libraries for implementation on a wide range of public key cryptosystems by using the performance of primitive operations as the main evaluation criterion, then further introducing other secondary criteria such as support for public key primitives and schemes, documentation, ease of use and portability.

Evaluation of software performance is not only considered difficult but also complex due to the increasing number of variables such as, operating system, processor, available memory and the choice of compiler and its optimization options. In order to achieve the performance evaluation, a methodology for ranking the entire libraries is developed based on the performance of their primitive cryptography related operations.

This study is intended to provide the developers of public key software implementations with knowledge needed to make better choices regarding the use of available libraries in their products based on the analysis of existing trade offs.

## 2 Libraries and Test Platforms

The libraries used in the comparison are listed in Table 1. Majority of these libraries can be described as multi-precision libraries or number theoretical libraries. The only exceptions are CryptoPP, MIRACL and OpenSSL which are specifically targeting cryptographic schemes.

**Table 1.** Libraries

| Library | Category | License | Version used |
|---|---|---|---|
| CLN | Number theoretic | GNU GPL | 1.1.5 |
| CryptoPP | Cryptographic | Copyrighted as a compilation | 5.1 |
| GMP | Multi-precision, Number theoretic | GNU GPL | 4.1.2 |
| LiDIA | Number theoretic | LiDIA group | 2.1pre7 |
| MIRACL | Cryptographic | Shamus Software Ltd. | 4.82 |
| NTL | Number theoretic | GNU GPL | 5.3.1 |
| OpenSSL | Cryptographic | Apache-style license | 0.9.7c |
| PIOLOGIE | Multi-precision, Number theoretic | www.hipilib.de | 1.3.2 |

CLN, LiDIA and NTL were compiled using GMP as an underlying multi-precision library as recommended by the library developers to achieve maximum speed. The structure of GMP has six function categories; two of them are used by the aforementioned libraries. These two are: mpz, high-level functions for signed/unsigned integer arithmetic, and mpn, low-level functions that operate on natural numbers. Most mpn functions contain machine-dependent code and are used by other function categories including mpz. CLN and NTL use GMP

mpn functions to build a different user interface, while LiDIA uses mpz functions. There are two different types of editions of PIOLOGIE; the normal editions, dependent on specific processors, compilers and operating systems; and the special editions, independent of these factors. A special scientific edition v1.3.2 was used in this paper. This edition is distributed under the terms and conditions of the GNU General Public License.

Two machines were used for the performance analysis, 2.0GHz Pentium IV with 512 MB RAM and 2x 400MHz UltraSPARC-Solaris-II with 4-MB E-cache and 2048 MB RAM. The Pentium IV machine is hosting two operating systems, Windows XP (Cygwin) and RedHat Linux 9.0. The libraries were compiled using GNU C/C++ compiler on all three operating systems using instructions provided by the libraries writers. Measurements were analyzed to obtain a general overall ranking of the libraries with respect to one another on each platform based on the overall rank of each operation.

## 3 Cryptographic Operations

The primitive cryptographic operations and sizes of operands were chosen based on their use in practical cryptographic algorithms related to the three well-known mathematical problems, integer factorization, discrete logarithm and elliptic curve discrete logarithm, such as RSA, DSA and ECDSA. The operations are divided into two main sets according to the operand sizes and types. The first set contains operations on large integers: multiplication, modular exponentiation, greatest common divisor and multiplicative inverse (extended greatest common divisor), with operands sizes 768, 1024 and 2048 bits.

The second set, contains operations on elliptic curve points; point addition and scalar multiplication with base point order lengths 163, 233 and 409 bits (equivalent to 1024, 2240 and 7680 bit RSA/DSA keys [21]) for elliptic curves over binary fields (EC2) and 162, 226 and 386 bits (equivalent to 1024, 2048 and 7680 bit RSA/DSA keys[21]) for randomly generated elliptic curves over prime fields (ECP).

### 3.1 Large Integer Operations

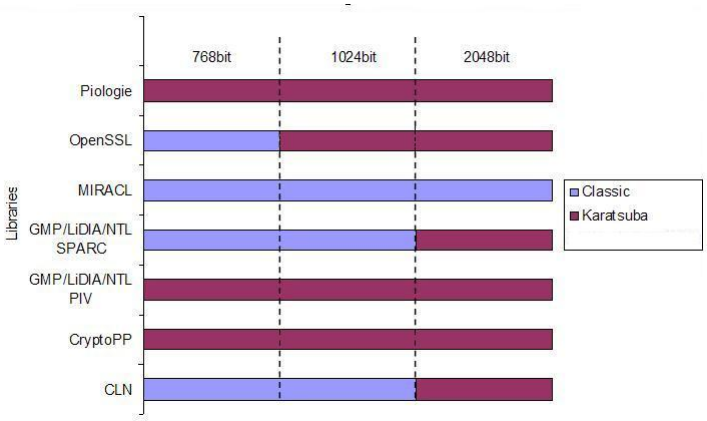
**Multiplication:** Multiplication Algorithms implemented in the libraries are summarized in Table 2. The Karatsuba[1][11] algorithm has a running complexity of $O(n^{log\,3})$ which is an improvement over the classical multiplication [11] algorithm at $O(n^2)$. Classical, Comba[4] and Karatsuba multiplication algorithms are of practical importance for the operand sizes used in the performance testing. The Toom-Cook (T-C) algorithm[11], with a running complexity of $O(n\ 2^{\sqrt{2\ log\ n}}\ log\ n)$ and Fast Fourier Transform (FFT) algorithm[11], with running complexity of $O(n\ log\ n\ log\ log\ n)$ are asymptotically superior to Karatsuba algorithm. However these algorithms do not offer any speed improvements for the operand sizes currently used in public key cryptography. Figure 1 shows the different operand sizes in bits and the algorithms used for

**Table 2.** Multiplication Algorithm Ranges

| | CLN | CryptoPP | GMP LiDIA NTL/PIV | GMP LiDIA NTL/SPARC | OpenSSL | PIOLOGIE |
|---|---|---|---|---|---|---|
| Classical | [0,1120) | - | [0,576) | [0,1280) | [0,512) | [0,256] |
| Comba | - | [0,256] | - | - | - | - |
| Karatsuba | [1120,80000) | $> 256$ | [576,4448) | [1280,7104) | $\geq 512$ | (256,160000) |
| T-C | - | - | [4448,188416) | [7104,122800) | - | - |
| FFT | $\geq 80000$ | - | $\geq 188416$ | $\geq 122800$ | - | $\geq 160000$ |

their multiplication. All libraries were compiled using default ranges and thresholds, however these thresholds can be changed to best fit the underlying platform/microprocessor. GMP is the only library that adjusts the threshold not only depending on operand sizes, but also on the underlying microprocessor architecture, a set of tune up programs are supplied with GMP that can be invoked on the targeted machine to measure the timing of GMP routines and propose thresholds that produce better results. The library must be recompiled in order for the change to be effective. This directly affects LiDIA and NTL which use GMP implementations of multiplication algorithms. On the other hand CLN uses GMPs mpn functions to build its own multiplication algorithms as a result is not directly affected. CryptoPPs implementation of the Karatsuba algorithms

**Fig. 1.** Multiplication Algorithms for Different Key




requires the input sizes to be powers of 2. In case they are not, they have to

be extended to the next power of 2 before applying the algorithm e.g. an input of size 768 bit is extended to 1024 bits. Although, OpenSSLs implementation of Karatsuba-Comba algorithm also requires the input sizes to be powers of 2, classical multiplication is used when this condition does not hold. MIRACL implements classical multiplication for all sizes.

**Modular Exponentiation:** Modular exponentiation algorithms implemented in the libraries and their corresponding thresholds are summarized in Table 3. Left-to-right[15] (denoted as LR) and right-to-left[15] (denoted as RL) algorithms require $L(E) - 1$ squarings, where $L(E)$ is the bitlength of the exponent, and $W(E) - 1$ multiplications, where $W(E)$ is the Hamming weight of the exponent. Both algorithms do not require precomputations. Left-to-right k-ary[15](denoted as LR k-ary), simultaneous multiple exponentiation[15] (denoted as SME) and sliding window[12][15] (denoted as k-ary SW) algorithms need precomputations. Figure 2 shows the algorithms used by the respective li-

**Table 3.** Modular Exponentiation Algorithm Ranges

| Library | CLN | CryptoPP | GMP | LiDIA | MIRACL | NTL | OpenSSL | PIOLOGIE |
|---|---|---|---|---|---|---|---|---|
| LR | [2,8] | - | [2,32] | - | - | [2,512) | - | - |
| RL | - | - | - | $\geq 2$ | - | - | - | $\geq 2$ |
| LR k-ary | (8,) | - | - | - | - | - | - | - |
| SME | - | $\geq 2$ | - | - | - | - | - | - |
| k-ary SW | - | - | $> 32$ | - | $\geq 2$ | $\geq 512$ | $\geq 2$ | - |

braries for three different exponents, $E = 3$, $E = 65537$, and a random exponent the same size as the size of the modulus $N$.

**GCD and xGCD:** Table 4 summarizes the libraries implementations for the GCD and xGCD algorithms. Complete description and analysis of these algorithms can be found in [11].

### 3.2 Elliptic Curve points

Operations on elliptic curves are limited to four libraries: CryptoPP, LiDIA, MIRACL and OpenSSL. The used version of OpenSSL supports only ECP, and does not suppoet EC2.

**Scalar Multiplication:** Table 5 summarizes the libraries implementations of EC point scalar multiplication.

**Fig. 2.** Modular Exponentiation Algorithms for Different Exponent Sizes

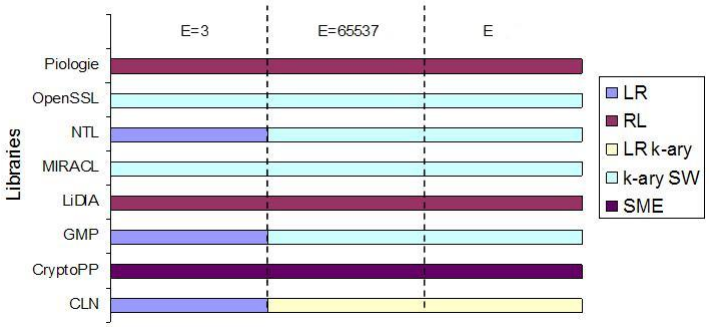



**Table 4.** GCD and xGCD Algorithms

| Library | GCD | xGCD |
|---|---|---|
| CLN | Lehmer[13][11][15] | Lehmer |
| CryptoPP | Euclid[11][15] | Binary[11][15] |
| GMP/LiDIA/NTL | Generalized Binary[10][23] | Lehmer |
| MIRACL | Lehmer | Lehmer |
| OpenSSL | Binary | Binary |
| PIOLOGIE | Generalized binary | Euclid |

**Table 5.** Elliptic Curve Scalar Multiplication Algorithms

| Library | Scalar Multiplication |
|---|---|
| CryptoPP | Simultaneous Sliding Window |
| LiDIA | Left-to-Right |
| MIRACL | wNAF-based interleaving [16] |
| OpenSSL | wNAF-based interleaving |

## 4 Methodology

Measurements were conducted in two different ways depending on the platform. The first method of testing referred to as RDTSC method was used on Pentium IV platforms. The RDTSC method uses the RDTSC[9][2] (read time-stamp counter) instruction to access the time-stamp counter, a 64 bit model specific register that is incremented every clock cycle, present on Intel processors beginning with the Pentium processor. The CPUID instruction is used as a serializing instruction to prevent out-of-order execution. The RDTSC method was used to determine the number of clock cycles required to perform the given operation.

The overhead associated with the call of the instructions was calculated and subtracted from the final result. Both instructions were called several times before testing the given operation to flush the instruction cache.

The second method referred to as Timing method was used on the UltraSPARC platform. The Timing method uses the function gettimeofday()[1] to determine the amount of time in milliseconds consumed in the execution of the given operation due to the lack of CPU cycle counter in the UltraSPARC platform. The function gettimeofday() gives resolution in the range of microseconds.

### 4.1 Operands

For large Integer Operations, two groups of operands were used: Group A, a group of randomly generated integers containing three sets of numbers with sizes 768, 1024 and 2048 bits respectively. Each set contains three large integers denoted as $I_i$, $J_i$ and $K_i$, where $i$ is the size of the integer in bits, e.g. the first set contains $I_{768}$, $J_{768}$, $K_{768}$ . The values of $I_i$, $J_i$ and $K_i$ are listed in Appendix A to [24]. Group B, a group of randomly generated large prime numbers contains three sets of numbers with sizes 768, 1024 and 2048 bits respectively. Each set contains ten large prime integers denoted as $P_i^j$, where $i$ is the size of an integer in bits and $j$ is the index of a given prime in the set. For example, the first set, contains $P_{768}^0$, $P_{768}^1$, ... $P_{768}^9$. Table 6 summarizes the operations tested and the corresponding groups of operands. With respect to

**Table 6.** Large Integer Operations

| Operation | OP | Group | Comments |
|---|---|---|---|
| Multiplication | MUL | A | $I_i \times J_i$ |
| MOD Exp E = 3 | $E_3$ | A | $I_i^3$ MOD $K_i$ |
| MOD Exp E = 65537 | $E_{65537}$ | A | $I_i^{65537}$ MOD $K_i$ |
| MOD Exp E, size of modulus | $E$ | A | $I_i^{J_i}$ MOD $K_i$ |
| Greatest Common Divisor | GCD | B, A | GCD($P_i^j$, $K_i$) |
| Extended GCD | xGCD | B, A | xGCD($P_i^j$, $K_i$) |

a particular library under a particular operating system, each operation using group A of operands is tested using either RDTSC or Timing method. All operations are tested using three operand sizes. Thus, each experiment on a given operation produces three sets of 100 execution times. Each value represents one iteration while each set represents one operand size. The three sets of 100 execution times are sorted and the minimum value for each set is recorded and denoted as $LIB^{OP}_{AMIN_{768}}$, $LIB^{OP}_{AMIN_{1024}}$, $LIB^{OP}_{AMIN_{2048}}$. The final set of raw results for each operation tested on a particular library under a certain operating system OS is denoted by $LIB^{OP}_{OS} = \{ LIB^{OP}_{AMIN_{768}}, LIB^{OP}_{AMIN_{1024}}, LIB^{OP}_{AMIN_{2048}} \}$. The same approach was used with operations using group B, except that each

operation is tested using 10 different operands $P_i^j$ of the same size e.g. there are 10 different operands for 768 bits $P_{768}^j$, $0 \leq j \leq 9$ so for each $j$ there will be 100 different values. For each $P_i^j$ the 100 values are sorted and the minimum recorded and denoted as $LIB^{OP}_{BMIN_i^j}$. The 10 minimum values for each operand size ($LIB^{OP}_{BMIN_{768}^j}$, $LIB^{OP}_{BMIN_{1024}^j}$, $LIB^{OP}_{BMIN_{2048}^j}$),$0 \leq j \leq 9$ are then averaged, the result is denoted as $LIB^{OP}_{BAVG_i}$.

$$LIB^{OP}_{BAVG_i} = \frac{1}{10} \sum_{j=0}^{9} LIB^{OP}_{BMIN_i^j} \tag{1}$$

The final set of raw results for each operation in a particular library under a certain operating system is denoted by $LIB^{OP}_{OS} = \{\ LIB^{OP}_{BAVG_{768}}, LIB^{OP}_{BAVG_{1024}}, LIB^{OP}_{BAVG_{2048}}\ \}$.

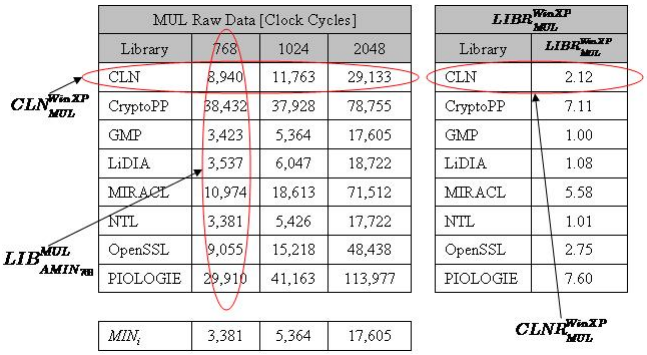
Elliptic curve point operations tested are Point Addition and Scalar Multiplication with input sizes of 163, 233 and 409 bits for EC2 and 162, 226 and 386 bits for ECP. For each elliptic curve, two randomly generated points $T_i$ and $S_i$ ($i$ = 163, 233, 409 for EC2, $i$ = 162, 226, 386 for ECP) were used as operands. Addition: $T_i + S_i$. Scalar Multiplication: ($r$ - 2) $T_i$ were $r$ is the order of the base point. The final raw results are collected as described for large integer operations using group A of operands. The elliptic curves and points are listed in Appendix A to [24].

### 4.2 Operation Ranking

After obtaining all values of execution times for all operations of the eight libraries under the three operating systems, rankings of the operations were calculated as follows: With respect to an operation OP tested on eight libraries under a particular operating system OS, execution times of OP are rearranged into three sets of eight values such that each set contains the results for a particular operand size under the eight libraries (one value for each execution time under a particular library i.e. $LIB^{OP}_{AMIN_i}$ or $LIB^{OP}_{BAVG_i}$ according to the operand group). The minimum value in each set, denoted by $MIN_i$, where i = 768, 1024, 2048, is determined and all values in a given set are divided by that value. The resulting values, denoted by $LIB_i r^{OS}_{OP}$ represent operation OP ranks with operands of size i on library LIB. For an operation OP, $LIB_i r^{OS}_{OP} = 1.00$ corresponds to the fastest library. A rank equal to $r$ means that an operation under a given library is $r$ times slower than the same operation under the fastest library for a given operand size $i$. Operation OP overall rank under a library $LIB$ denoted by $LIBR^{OS}_{OP}$ is the geometric mean of its ranks for the three operand sizes 768, 1024 and 2048. Figure 3 shows the raw results and ranks for multiplication under Pentium IV-Windows XP for all libraries. The final rank of CLN multiplication is calculated as follows:

$$CLN_{768} r^{WinXP}_{MUL} = \frac{CLN^{MUL}_{AMIN_{768}}}{MIN_{768}} = \frac{8,940}{3,381} = 2.64 \tag{2}$$

**Fig. 3.** CLN Multiplication Rank




$$CLN_{1024}r_{MUL}^{WinXP} = \frac{CLN_{AMIN_{1024}}^{MUL}}{MIN_{1024}} = \frac{11,763}{5,364} = 2.19 \tag{3}$$

$$CLN_{2048}r_{MUL}^{WinXP} = \frac{CLN_{AMIN_{2048}}^{MUL}}{MIN_{2048}} = \frac{29,133}{17,605} = 1.65 \tag{4}$$

$$CLNR_{MUL}^{WinXP} = \sqrt[3]{\prod_i CLN_i r_{MUL}^{WinXP}} = \sqrt[3]{2.64 \times 2.19 \times 1.65} = 2.1208 \tag{5}$$

### 4.3 Library Ranking

As a result we will have a set of operation rankings for each library on each operating system; the overall rank of the library denoted by $LIBR^{OS}$ on a particular operating system OS is determined by calculating the geometric mean of its individual operation ranks.

$$LIBR^{OS} = \sqrt[N]{\prod_{OP} LIBR_{OP}^{OS}} \tag{6}$$

N is the number of operations considered. N = 6 for large integer operations and N=2 for EC point operations. The two sets of rankings are considered separately because EC point operations are not supported by all libraries.

## 5 Performance Results

As discussed in the previous sections, the following tables show the individual operations, their ranks and the overall rankings of the libraries on the three platforms used for performance testing. Each table lists the individual operation ranking of each library and the overall ranking of the library (Geometric Mean of individual operation rankings).

### 5.1 Operations On Large Integers

The tables presented in this section show the overall ranking of the libraries for operations on large integers. The operations rankings are, MUL: Multiplication ranking, $E_3$: Modular Exponentiation Ranking with exponent = 3, $E_{65537}$: Modular Exponentiation Ranking with exponent = 65537, $E$: Modular Exponentiation Ranking with exponent of the same size as the modulus, GCD: Greatest Common Divisor ranking, and xGCD: Extended Greatest Common Divisor ranking.

Table 7 lists the performance results under Pentium IV, Windows XP. In terms of the overall rank $LIBR^{WinXP}$, GMP has the best rank and PIOLOGIE the worst. MIRACL and OpenSSL are very close. OpenSSL Multiplication and Modular Exponentiation are higher in rank than MIRACL; while MIRACL GCD and xGCD rank higher than OpenSSL. CryptoPP GCDs rank is higher than xGCDs rank unlike all other libraries while it is completely the opposite for PIOLOGIE; in both cases the reason behind that is the choice of algorithms (see Table 4). Table 8 lists the performance results under Pentium IV, RedHat 9.0. In terms of the overall rank $LIBR^{RH}$, GMP has the best rank and CryptoPP the worst. CryptoPP and PIOLOGIE are slower under RedHat than under Windows XP; the order of GMP, NTL, LiDIA and CLN is the same as for Pentium IV-Windows XP. MIRACLs rank is now slightly better than OpenSSLs due to rankings of GCD and xGCD. PIOLOGIE rank is slightly better than CryptoPP due to Modular Exponentiations rank with $E = 3$ and GCDs rank. Table 9 lists the performance results under Ultra-SPARC, Solaris. In terms of the overall rank (LIB-R)SPARC, GMP has the best rank and CryptoPP the worst. GMP, NTL, LiDIA and CLN have the same order. PIOLOGIE has a better ranking than Pentium IV, while OpenSSLs rank remains the same.

Under all operating systems, LiDIA and NTL use GMP functions for Multiplication, GCD and xGCD. This makes their rankings according to these operations very much close to GMP. For Modular Exponentiation, NTL and LiDIA have their own different implementations, with NTL choice of algorithms similar to GMP. CLN has its own implementation for all operations. As an overall result

**Table 7.** Large Integer Operation Rankings Pentium IV, Windows XP

| Library | MUL | $E_3$ | $E_{65537}$ | $E$ | GCD | xGCD | $LIBR^{WinXP}$ |
|---|---|---|---|---|---|---|---|
| CLN | 2.12 | 2.23 | 2.25 | 2.79 | 1.34 | 1.37 | 1.95 |
| CryptoPP | 7.11 | 15.17 | 4.71 | 4.04 | 464.90 | 9.99 | 14.56 |
| GMP | 1.00 | 1.00 | 1.00 | 1.00 | 1.01 | 1.08 | 1.01 |
| LiDIA | 1.08 | 1.45 | 1.08 | 1.65 | 1.03 | 1.10 | 1.21 |
| MIRACL | 3.58 | 22.40 | 4.56 | 2.62 | 5.15 | 3.15 | 5.00 |
| NTL | 1.01 | 1.42 | 1.17 | 1.18 | 1.00 | 1.00 | 1.12 |
| OpenSSL | 2.75 | 8.07 | 2.65 | 2.33 | 8.31 | 12.17 | 4.90 |
| PIOLOGIE | 7.60 | 7.40 | 6.63 | 10.65 | 16.41 | 213.30 | 15.51 |

**Table 8.** Large Integer Operation Rankings Pentium IV, RedHat 9.0

| Library | MUL | $E_3$ | $E_{65537}$ | $E$ | GCD | xGCD | $LIBR^{RH}$ |
|---|---|---|---|---|---|---|---|
| CLN | 1.50 | 1.27 | 1.39 | 1.87 | 1.37 | 1.27 | 1.43 |
| CryptoPP | 4.49 | 9.19 | 3.79 | 5.04 | 65.96 | 16.82 | 9.78 |
| GMP | 1.00 | 1.00 | 1.01 | 1.00 | 1.00 | 1.08 | 1.01 |
| LiDIA | 1.00 | 1.10 | 1.06 | 1.84 | 1.00 | 1.09 | 1.15 |
| MIRACL | 3.60 | 21.06 | 4.30 | 2.77 | 3.99 | 2.36 | 4.52 |
| NTL | 1.01 | 1.20 | 1.10 | 1.29 | 1.01 | 1.00 | 1.10 |
| OpenSSL | 2.80 | 7.12 | 2.43 | 2.43 | 8.93 | 12.49 | 4.86 |
| PIOLOGIE | 5.35 | 5.01 | 5.07 | 8.79 | 21.95 | 24.22 | 9.27 |

**Table 9.** Large Integer Operations Rankings UltraSPARC, Solaris

| Library | MUL | $E_3$ | $E_{65537}$ | $E$ | GCD | xGCD | $LIBR^{SPARC}$ |
|---|---|---|---|---|---|---|---|
| CLN | 1.21 | 1.60 | 1.70 | 1.98 | 1.67 | 1.40 | 1.58 |
| CryptoPP | 16.43 | 38.52 | 18.10 | 17.68 | 184.68 | 49.08 | 34.99 |
| GMP | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.12 | 1.02 |
| LiDIA | 1.00 | 1.20 | 1.10 | 1.55 | 1.02 | 1.14 | 1.16 |
| MIRACL | 9.08 | 23.85 | 2.98 | 7.41 | 8.77 | 3.71 | 7.33 |
| NTL | 1.00 | 1.25 | 1.15 | 1.13 | 1.05 | 1.00 | 1.09 |
| OpenSSL | 2.16 | 7.80 | 2.81 | 2.45 | 7.67 | 7.74 | 4.36 |
| PIOLOGIE | 3.51 | 4.13 | 4.06 | 5.95 | 9.13 | 37.22 | 7.01 |

for large integer operations, GMP has the best ranking under all three platforms followed by NTL, LiDIA and CLN.

### 5.2 Operations On EC2 and ECP Points

Figures 4 and 5 show the performance results for operations on EC2 and ECP points respectively, on all platforms.

For EC2, LiDIA has the best rank under Pentium IV,RedHat 9.0 and UltraSPARC,Solaris while CryptoPP has the worst rank under all platforms. MIRACL has the best rank under Pentium IV, Windows XP.

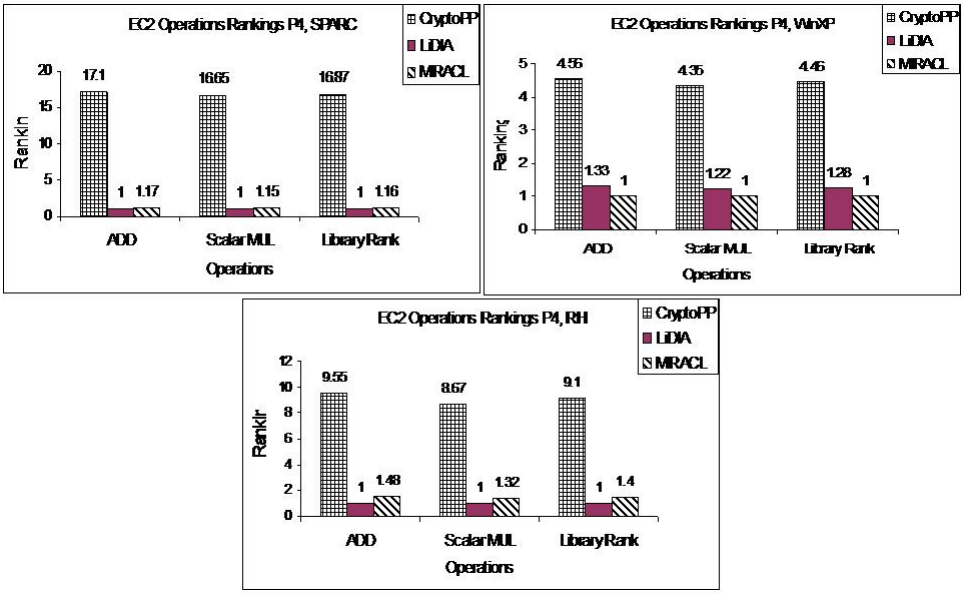
For ECP, under all platforms, OpenSSL has the best rank and CryptoPP has the worst rank.

## 6 Observations And Comments

### 6.1 Portability, Documentation and Ease of Use

The number of supported compilers was considered as a measure of portability of a given library. For CLN, GMP and LiDIA the only supported compiler is GNU

**Fig. 4.** EC2 Operations Rankings



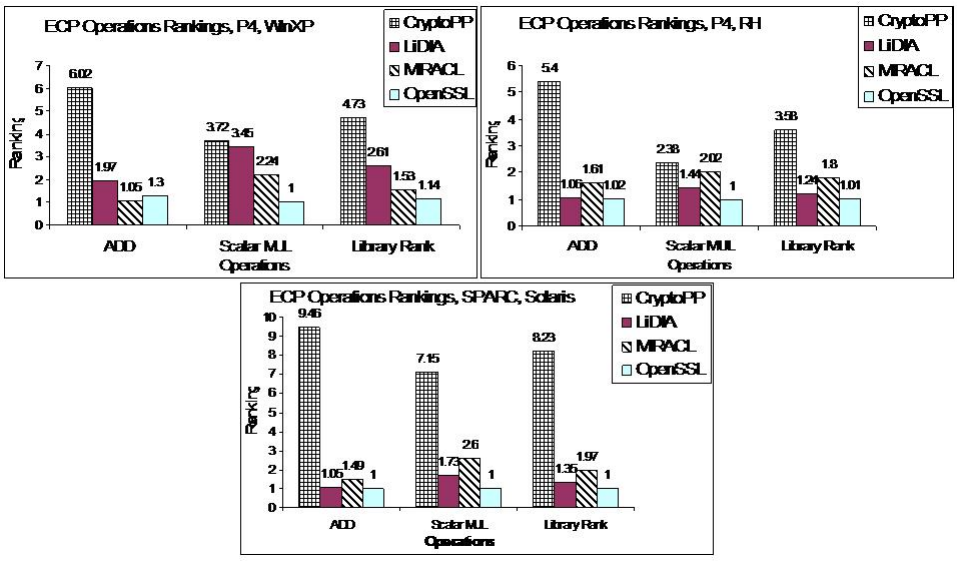

C/C++. PIOLOGIE supports the largest set of compilers followed by CryptoPP, OpenSSL, MIRACL and NTL.

In terms of documentation and ease of use, PIOLOGIE simple structure makes it the easiest among all libraries, on the other hand CryptoPP complex structure and insufficient documentation makes it the hardest among all libraries. CLN, GMP, LiDIA, MIRACL and NTL have complex structure but their documentation and documentation sample code and test suites decrease their difficulty.

### 6.2 CryptoPP GNU C/C++ vs. MS VC++ 6.0

The CryptoPP's performance was tested under MS VC++ 6.0 and the results were compared to the results obtained under GNU C/C++. The execution time ratios for GNU C/C++ vs MS VC++ 6.0 were computed for multiplication and modular exponentiation for three input sizes under the Pentium IV, Windows XP machine. It was found that CryptoPP compiled under MS VC++ 6.0 is more than twice as fast as that compiled under GNU C/C++. This is due to the library's optimization for Pentium IV processors under MS VC++ 6.0 versus its generic Pentium optimization under GNU C/C++. MIRACL and PIOLOGIE were also compiled under MS VC++ 6.0 with no significant change in their performance as compared to GNU C/C++.

**Fig. 5.** ECP Operations Rankings




### 6.3 Support For Public Key Cryptosystems

Support for public key cryptosystems is based on the support of primitive arithmetic and number theoretical operations needed by the three main categories of public key cryptosystems, and also on complete implementations of public key schemes present in the library.

**Support for Primitive Operations** Figure 7 summarizes the libraries support for primitive operations on large integers, ECP and EC2. Support for ECP operations is limited to CryptoPP, LiDIA, MIRACL and OpenSSL. Support for EC2 is limited to CryptoPP, LiDIA and MIRACL. Elliptic curve generation and point counting is supported by LiDIA and MIRACL only.

**Support for Cryptographic Schemes** Complete implementation of Cryptographic schemes is limited to three libraries, CryptoPP, MIRACL and OpenSSL. CryptoPP has the largest collection of public key cryptographic schemes followed by MIRACL and OpenSSL. Moreover, CryptoPP contains a collection of secret key ciphers, hash functions, and MAC functions and I/O support. Version 5.0.4 of the library has received FIPS 140-2 level 1 validation in 9/5/2003. MIRACL implements cryptographic primitives in IEEE P1363[8]. It also has implementations for AES and SHA (1, 256, 384, and 512). OpenSSL implements DH,

DSA and RSA, and a collection of secret key ciphers, hash functions and MAC functions.

**Fig. 6.** Support for Primitive Operations

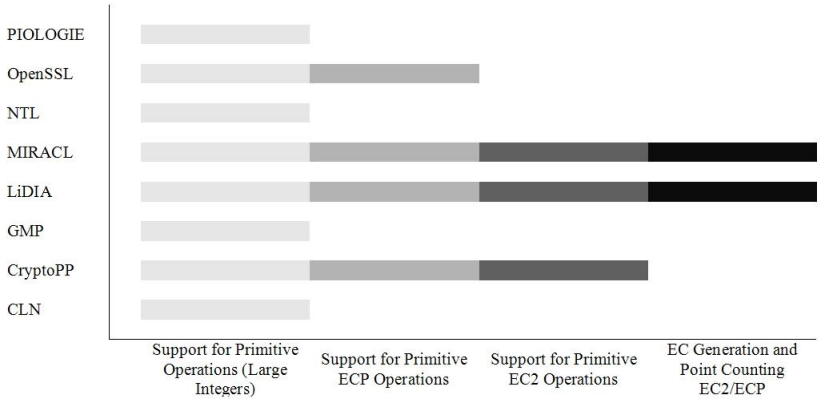



**Fig. 7.** Support vs Performance, Pentium IV, RedHat 9.0

| | | | | | |
|---|---|---|---|---|---|
| **Support** | **PKS**(1) | | CryptoPP | MIRACL OpenSSL | |
| | **EC2** | **PG**(2) **PC**(3) | CryptoPP | MIRACL LiDIA | |
| | **ECP** | **PG** **PC** | CryptoPP | MIRACL LiDIA OpenSSL | |
| | **LINT** | **High** | CryptoPP | MIRACL OpenSSL | CLN LiDIA NTL GMP |
| | | **Low** | PIOLOGIE | | |
| | | | low | Performance | high |

(1)PKS: Public Key Schemes.
(2)PG: Point Generation.
(3)PC: Point Counting.

## 7 Conclusion

In terms of support for operations on large integers, GMP, NTL, LiDIA, and CLN have the best performance under all platforms tested, with GMP being the fastest and CLN the slowest among the group. LiDIA is the only library in the group that needs a license for commercial use. For a developer targeting operations on large integers, GMP would be the best choice in terms of performance. The trade off however, is the amount of time and effort needed for implementation, and portability.

OpenSSL and MIRACL trail libraries from the first group in terms of overall performance. OpenSSL is faster than MIRACL for all operations except GCD and xGCD. While having an acceptable performance as compared to other libraries and support implementations of cryptographic schemes, this group is a good choice for fast development of public key cryptosystems based on operations on large integers.

CryptoPP is the best choice for the fast development based on the complete implementations of a wide range of cryptographic schemes involving large integers; the drawback is its performance as compared to other libraries.

For elliptic curves over binary fields, the competition is between LiDIA, MIRACL and CryptoPP. LiDIA has the best performance under Pentium IV-RedHat 9.0 and UltraSPARC-Solaris. Under Pentium IV, Windows XP, MIRACL has the best performance. CryptoPP is the slowest under all platforms.

For elliptic curves over prime fields, OpenSSL has the best performance under all platforms, LiDIA performance is better than MIRACL on Pentium IV-RedHat 9.0 and UltraSPARC-Solaris, while under Pentium IV-Windows XP, MIRACL is better than LiDIA. Again CryptoPP has the lowest performance. For a developer targeting ECP cryptosystems, OpenSSL is a good choice since it has the best performance, and is portable and free.

Although public key schemes implemented in CryptoPP, MIRACL and OpenSSL were not compared for performance, one can estimate their performance based on the performance of primitive operations. Accordingly, OpenSSL is expected to have better performance followed by MIRACL and CryptoPP respectively, with CryptoPP having the richest collection of cryptographic schemes.

Figure 7 summaries the libraries support of public key cryptosystems versus their performance under Pentium IV, RedHat 9.0. The x-axis represents the performance scale from low performance to high performance, y-axis represents support starting from support for large integers (LINT) and ending with support for public key schemes (PKS). GMP has the highest performance and lowest support, while CryptoPP has the highest support and lowest performance.

This work has been done before the introduction of eBATS [24], and currently we are trying to use it to test the various asymmetric operations implemented in the eight libraries . We hope that this will give more insight on some of the reasons why one libaray cryptographic operation might perform better than the others under a specific platform.

## References


1. A. Karatsuba and Yu. Ofman, Multiplication of Multidigit Numbers on Automata. Soviet Physics-Doklady, 7 (1963), 595-596.
2. R. E. Bryant and D. O'Hallaron. Computer Systems, A Programmer's Perspective. Prentice-Hall, 2003.
3. CLN: Class Library for Numbers http://www.ginac.de/CLN/
4. P. G. Comba. Exponentiation cryptosystems on the IBM PC. IBM Systems Journal, vol. 29, n. 4, pp. 526538, 1990.
5. Crypto++ Library 5.1: a Free C++ Class Library of Cryptographic schemes http://www.eskimo.com/ weidai/cryptlib.html
6. The GNU MP Library http://www.swox.com/gmp/
7. HiPiLib Piologie http://www.hipilib.de/piologie.htm
8. IEEE P1363: Standard Specifications for Public-Key Cryptography. http://grouper.ieee.org/groups/1363/
9. Intel Corporation. IA-32 Intel Architecture, Software Developers Manual, vol 2B: Instruction Set Reference, N-Z http://developer.intel.com/design/pentium4/manuals/25366713.pdf
10. Tudor Jebelean. A Generalization of the Binary GCD Algorithm. ISSAC 93, 111-116.
11. D.E. Knuth.The Art in Computer Programming. Vol2 : Seminumerial Algorithms. Addison-Wesley, 2nd.Ed. 1981.
12. C.K. Koc. Analysis of sliding window techniques for exponentiation. Computers and Mathematics with Applications, vol.30, n.10, pp.1724,195.
13. D. H. Lehmer. Euclid's Algorithm for Large Numbers. American Mathematical Monthly. 45 (1938), 227-233.
14. LiDIA: A C++ Library For Computational Number Theory http://www.informatik.tu-darmstadt.de/TI/LiDIA/
15. A. Menesez, P. van Oorschot, and S. Vanstone. Handbook of Applied Cryptography. CRC Press, 1997.
16. B. Möller. Algorithms for multi-exponentiation. Selected Areas in Cryptography SAC 2001 (2001), S. Vaudenay and A.M. Youssef (Eds.), LNCS 2259, pp. 165180.
17. NTL: A Library for doing Number Theory http://www.shoup.net/ntl/
18. OpenSSL: The Open Source toolkit for SSL/TLS http://www.openssl.org/
19. R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public Key Cryptosystems. Communications of the ACM, vol. 21, no.2, pp. 158-164, 1978.
20. Shamus Software Ltd MIRACL http://indigo.ie/ mscott/
21. Standards for Efficient Cryptography Group. SEC 2: Recommended Elliptic Curve Domain Parameters. Version 0.6, 1999.
22. S. Y. Yan. Number Theory for Computing. Springer-Verlag 2000
23. T. Jebelean. A Generalization of the Binary GCD Algorithm. ISSAC 93, pp. 111-116.
24. eBATS: ECRYPT Benchmarking of Asymmetric Systems http://www.ecrypt.eu.org/ebats/