Libraries

Yung-Hsiang Liu

Department of Computer Science, National Chiao Tung University

ECC 2009
Outline

- Why we use the libraries?
- Multi Precision Numbers
  - GMP: GNU Multiple Precision Arithmetic Library
  - NTL: A Library for doing Number Theory
- Elliptic Curves Arithmetic
  - add, scalar multiplication, random point, …
- Libraries with ECC implementation
  - MIRACL: Multiprecision Integer and Rational Arithmetic C/C++ Library
  - Crypto++: Crypto++ Library
Why we use the libraries?

- In C/C++
  - unsigned long long: 64 bits

- RSA-1024
  - p and q are two primes of about 500 bits

- How about RSA-32?

- How about ECC-32?
  - RSA-1024 = ECC-160
Multi Precision Numbers

- In Java:
  - BigInteger

- In C/C++:
  - `int bignum[100]` ?
  - What are the operations, +, -, x, / ?
  - Efficient ?

- Use Libraries!!!
GMP

- [http://gmplib.org/](http://gmplib.org/)

**What is GMP?**

GMP is a free library for arbitrary precision arithmetic, operating on signed integers, rational numbers, and floating point numbers. There is no practical limit to the precision except the ones implied by the available memory in the machine GMP runs on. GMP has a rich set of functions, and the functions have a regular interface.

The main target applications for GMP are cryptography applications and research, Internet security applications, algebra systems, computational algebra research, etc.

GMP is carefully designed to be as fast as possible, both for small operands and for huge operands. The speed is achieved by using fullwords as the basic arithmetic type, by using fast algorithms, with highly optimised assembly code for the most common inner loops for a lot of CPUs, and by a general emphasis on speed.

GMP is faster than any other bignum library. The advantage for GMP increases with the operand sizes for many operations, since GMP uses asymptotically faster algorithms.

The first GMP release was made in 1991. It is continually developed and maintained, with a new release about once a year.

GMP is distributed under the [GNU LGPL](http://www.gnu.org/copyleft/lgpl.html). This license makes the library free to use, share, and improve, and allows you to pass on the result. The license gives freedoms, but also sets firm restrictions on the use with non-free programs.

GMP is part of the GNU project. For more information about the GNU project, please see the [official GNU web site](http://www.gnu.org/).

GMP’s main target platforms are Unix-type systems, such as GNU/Linux, Solaris, HP-UX, Mac OS X/Darwin, BSD, AIX, etc. It also is known to work on WinDoze in 32-bit mode.

GMP is brought to you by a team [listed in the manual](http://gmplib.org/manual/).

GMP is carefully developed and maintained, both technically and legally. We of course inspect and test contributed code carefully, but equally importantly we make sure we have the legal right to distribute the contributions, meaning users can safely use GMP. To achieve this, we will ask contributors to sign paperwork where they allow us to distribute their work.
GMP

- GMP provides arbitrary precision arithmetic
  - signed integers
  - rational numbers
  - floating point numbers

- GMP is for
  - cryptography applications and research
  - Internet security applications
  - algebra systems
  - computational algebra research
GMP

- Why GMP?
  - GMP is carefully designed to be as fast as possible
  - GMP is faster than any other bignum library

- Who uses GMP?

**Projects using GMP**

There are many interesting projects that rely on GMP. Here are a few examples, in alphabetic order:

- **CLN**, which uses low-level routines from GMP to build a different high-level user interface.
- **ECMNET**, a factorisation project initiated and coordinated by Paul Zimmermann. Please join it if you have free computer cycles!
- **Kaffe**, The well-known free Java machine.
- **LiDIA**, an object-oriented library for computational number theory.
- **Ish**, a project to develop a free (GPLed) implementation of the Secure Shell protocol V2.
- **MPFR**, an add-on library for floating-point arithmetic. This library's mpfr functions are usually to prefer before GMP's own mpf functions.
- **MPFR**, This "angry GMP fork" might be an alternative to the real GMP for some Windows users, but they'll have to deal with lots of anti-GMP sentiments.
- **NTL**, a number theory library.
- **PARI/GP**, a software package for computer-aided number theory.
- **Sage**, is a free mathematics software system, which is gradually becoming a viable alternative to non-free computer algebra systems.
- All well-known non-free computer algebra systems are now using GMP. While this is a recognition of GMP, these programs' non-free nature is unfortunate.
GMP

- [http://gmplib.org/manual/](http://gmplib.org/manual/)

- What we need?
  - Installing GMP
  - Integer Functions
    - +, -, *, /, ...
  - Random Number Functions
  - C++ Class Interface

- On windows?
  - MSYS, cygwin, …
NTL

- http://www.shoup.net/ntl/

**NTL: A Library for doing Number Theory**

NTL is a high-performance, portable C++ library providing data structures and algorithms for manipulating signed, arbitrary length integers, and for vectors, matrices, and polynomials over the integers and over finite fields.

- A Tour of NTL
- Download NTL
- Trouble-shooting guide
- Contact Info and Mailing Lists
- Related Links

**Now available: NTL 5.5.1**

[More detailed information about recent changes]

Back to Victor Shoup's Home Page
NTL

- NTL provides data structures and algorithms for
  - signed, arbitrary length integers
  - vectors
  - matrices
  - polynomials over the integers
  - polynomials over finite fields

- NTL has
  - clean and consistent interface to a large variety of classes representing mathematical objects
NTL

• NTL is
  o a good environment for easily and quickly implementing new number-theoretic algorithms, without sacrificing performance

• Who uses NTL?
### NTL

- **http://www.shoup.net/ntl/doc/tour-modules.html**

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GF2</strong></td>
<td>class $GF2$: integers mod 2</td>
</tr>
<tr>
<td><strong>GF2X</strong></td>
<td>class $GF2X$: polynomials over $GF(2)$ (much more efficient than using $zz_pX$ with $p=2$); includes routines for GCDs and minimal polynomials</td>
</tr>
<tr>
<td><strong>GF2XFactoring</strong></td>
<td>routines for factoring polynomials over $GF(2)$; also includes routines for testing for and constructing irreducible polynomials</td>
</tr>
<tr>
<td><strong>GF2XVec</strong></td>
<td>class $GF2XVec$: fixed-length vectors of fixed-length $GF2X$s; less flexible, but more efficient than $vec_GF2X$</td>
</tr>
<tr>
<td><strong>GF2E</strong></td>
<td>class $GF2E$: polynomial extension field/ring over $GF(2)$, implemented as $GF(2)[X]/(P)$</td>
</tr>
<tr>
<td><strong>GF2EX</strong></td>
<td>class $GF2EX$: polynomials over $GF2E$; includes routines for modular polynomials arithmetic, modular composition, minimal and characteristic polynomials, and interpolation.</td>
</tr>
<tr>
<td><strong>GF2EXFactoring</strong></td>
<td>routines for factoring polynomials over $GF2E$; also includes routines for testing for and constructing irreducible polynomials</td>
</tr>
<tr>
<td><strong>BNF</strong></td>
<td>routines for computing the Hermite Normal Form of a lattice</td>
</tr>
<tr>
<td><strong>LLL</strong></td>
<td>routines for performing lattice basis reduction, including very fast and robust implementations of the Schnorr-Euchner LLL and Block Korkin Zolotarev reduction algorithm, as well as an integer-only reduction algorithm. Also, there are routines here for computing the kernel and image of an integer matrix, as well as finding integer solutions to linear systems of equations over the integers.</td>
</tr>
<tr>
<td><strong>RR</strong></td>
<td>class $RR$: arbitrary-precision floating point numbers.</td>
</tr>
<tr>
<td><strong>ZZ</strong></td>
<td>class $ZZ$: arbitrary length integers; includes routines for GCDs, Jacobi symbols, modular arithmetic, and primality testing; also includes small prime generation routines and in-line routines for single-precision modular arithmetic</td>
</tr>
<tr>
<td><strong>ZZVec</strong></td>
<td>class $ZZVec$: fixed-length vectors of fixed-length $ZZ$s; less flexible, but more efficient than $vec_ZZ$</td>
</tr>
<tr>
<td><strong>ZZX</strong></td>
<td>class $ZZX$: polynomials over $ZZ$; includes routines for GCDs, minimal and characteristic polynomials, norms and traces</td>
</tr>
<tr>
<td><strong>ZZXFactoring</strong></td>
<td>routines for factoring univariate polynomials over $ZZ$</td>
</tr>
<tr>
<td><strong>ZZ_p</strong></td>
<td>class $ZZ_p$: integers mod $p$</td>
</tr>
<tr>
<td><strong>ZZ_pE</strong></td>
<td>class $ZZ_pE$: ring/field extension of $ZZ_p$</td>
</tr>
<tr>
<td><strong>ZZ_pEX</strong></td>
<td>class $ZZ_pEX$: polynomials over $ZZ_pE$; includes routines for modular polynomials arithmetic, modular composition, minimal and characteristic polynomials, and interpolation.</td>
</tr>
</tbody>
</table>
Elliptic Curves Arithmetic

- Elliptic curves over $\mathbb{Z}_p$ ($p>3$)
  - Let $a, b$ in $\mathbb{Z}_p$ and $4a^3+27b^2!=0 \mod p$
  - Define:

$$E_{a,b}(\mathbb{Z}_p) = \{(x, y) \in \mathbb{Z}_p^2 : y^2 = x^3 + ax + b \} \cup \{O\}$$

where $O$ is an identity point at infinity

Ex:

$$p = 23, \ a = 1, \ b = 0$$

$$E_{1,0}(\mathbb{Z}_{23}) = \{(x, y) \in \mathbb{Z}_{23}^2 : y^2 = x^3 + x \} \cup \{O\}$$
Elliptic Curves Arithmetic

- Elliptic curves over $\mathbb{Z}_p$ ($p>3$)

Given two points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$, their sum $P + Q = (x_3, y_3)$ is defined by:

$$x_3 = \lambda^2 - x_1 - x_2$$
$$y_3 = \lambda(x_1 - x_3) - y_1$$

where

$$\lambda = \begin{cases} 
\frac{y_2 - y_1}{x_2 - x_1}, & \text{if } P \neq Q \\
\frac{3x_1^2 + a}{2y_1}, & \text{if } P = Q
\end{cases}$$

And the equation of the curve is:

$$y^2 = x^3 + ax + b \Rightarrow 2yy' = 3x^2 + a \Rightarrow y' = \frac{3x^2 + a}{2y}$$
Elliptic Curves Arithmetic

- Elliptic curves over $GF(2^m)$

- Over $GF(2^n)$, Elliptic Curve can be written in the form:

  $$E : y^2 + xy = x^3 + ax^2 + b, \ b \neq 0$$

- Points on Elliptic Curve $E/ GF(2^n)$:

  $$E(GF(2^n)) = \{(x, y) \mid x, y \in GF(2^n), y^2 + xy = x^3 + ax^2 + b\} \cup \{O\}$$

  $O$ is an identity point at infinity
Elliptic Curves Arithmetic

- Elliptic curves over GF($2^m$)
  - Define $-P$
    \[
    \text{Let } P = (x_1, y_1) \in E(GF(2^m)) \text{ ; then } -P = (x_1, y_1 + x_1)
    \]
    \[P + (-P) = O \text{ , } O \text{ is an identity point at infinity}\]
  - Define $P+Q$
    \[
    \text{If } Q = (x_2, y_2) \in E(GF(2^m)) \text{ and } Q \neq -P
    \]
    \[\text{then } P + Q = (x_3, y_3), \text{ where}
    \]
    \[
    x_3 = \lambda^2 + \lambda + a + x_1 + x_2
    \]
    \[
    y_3 = (x_3 + x_1) \lambda + x_3 + y_1
    \]
    \[
    \begin{cases} 
    \lambda = \frac{y_2 + y_1}{x_2 + x_1}, & P \neq Q \\
    \lambda = \frac{x_1^2 + y_1}{x_1}, & P = Q 
    \end{cases}
    \]
Elliptic Curves Arithmetic

- How we calculate $kP$
  - $P+P+P+P+...$ ?
  - Double and Add !!!

- How we choose a random point on an elliptic curve
  - Choose $x$, and compute the suitable $y$
  - Choose $y$, and compute the suitable $x$

- Can I get more functions from the libraries?
MIRACL

- http://www.shamus.ie/

**MIRACL**

**Multiprecision Integer and Rational Arithmetic C/C++ Library**

MIRACL is a Big Number Library which implements all of the primitives necessary to design Big Number Cryptography into your real-world application. It is primarily a tool for cryptographic system implementors. RSA public key cryptography, Diffie-Hellman Key exchange, DSA digital signature, they are all just a few procedure calls away. Support is also included for even more esoteric Elliptic Curves and Lucas function based schemes. The latest version offers full support for Elliptic Curve Cryptography over GF(p) and GF(2^m) - see the links on this page for more details. Less well-known techniques can also be implemented as MIRACL allows you to work directly and efficiently with the big numbers that are the building blocks of number-theoretic cryptography. Although implemented as a C library, a well-thought out C++ wrapper is provided, which greatly simplifies program development. Most example programs (25+ of them) are provided in both C and C++ versions.
MIRACL


MIRACL offers full support for Elliptic Curve Cryptography (ECC) over the prime field GF(p), and the field GF(2^m), including four programs for point-counting. For more information on ECC see "Elliptic Curves in Cryptography", Blake, Seroussi & Smart, London Mathematical Society Lecture Notes Series 265, Cambridge University Press, ISBN 0 521 65374 6. This can be ordered directly from the publisher. Another good source of information is from the IEEE P1363 standards documents.

Elliptic Curves over GF(p) and GF(2^m) offer many advantages over standard methods such as RSA.

- An Elliptic Curve provides an ideal match for the AES (Advanced Encryption Standard) block encipherment (which is implemented within MIRACL). Using a 256 bit prime provides the same security as 128-bit AES. Similarly 384 bit ECC matches 192-bit AES, and 512 bit ECC matches 256-bit AES. Note that to offer the same level of security as 512-bit ECC would require the use of
## Crypto++

- [http://www.cryptopp.com/](http://www.cryptopp.com/)

### What is it?

Crypto++ Library is a free C++ class library of cryptographic schemes. Currently the library contains the following algorithms:

<table>
<thead>
<tr>
<th>algorithm type</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>authenticated encryption schemes</td>
<td>GCM, CCM, EAX</td>
</tr>
<tr>
<td>high speed stream ciphers</td>
<td>Panama, Sosemanuk, Salsa20, Xsalsa20</td>
</tr>
<tr>
<td>AES and AES candidates</td>
<td>AES (Rijndael), RC6, MARS, Twofish, Serpent, CAST-256</td>
</tr>
<tr>
<td>other block ciphers</td>
<td>IDEA, Triple-DES (DES-EDE2 and DES-EDE3), Camellia, SEED, RC5, Blowfish, TEA, XTEA, Skipjack, SHACAL-2</td>
</tr>
<tr>
<td>block cipher modes of operation</td>
<td>ECB, CBC, CBC ciphertext stealing (CTS), CFB, OFB, counter mode (CTR)</td>
</tr>
<tr>
<td>message authentication codes</td>
<td>VMAC, HMAC, CMAC, CBC-MAC, DMAC, Two-Track-MAC</td>
</tr>
<tr>
<td>hash functions</td>
<td>SHA-1, SHA-2 (SHA-224, SHA-256, SHA-384, and SHA-512), Tiger, WHIRLPOOL, RIPEMD-128, RIPEMD-256, RIPEMD-160, RIPEMD-320</td>
</tr>
<tr>
<td>public-key cryptography</td>
<td>RSA, DSA, ElGamal, Nyberg-Rueppel (NR), Rabin-Williams (RW), LUC, LUCELG, DLIES (variants of DHAES), ESIGN</td>
</tr>
<tr>
<td>padding schemes for public-key systems</td>
<td>PKCS#1 v2.0, OAEP, PSS, PSSR, IEEE P1363 EMSA2 and EMSA5</td>
</tr>
<tr>
<td>key agreement schemes</td>
<td>Diffie-Hellman (DH), Unified Diffie-Hellman (DH2), Menezes-Qu-Vanstone (MQV), LUCDIF, XTR-DH</td>
</tr>
<tr>
<td>elliptic curve cryptography</td>
<td>ECDSA, ECNR, ECIES, ECDH, ECMQV</td>
</tr>
<tr>
<td>insecure or obsolescent algorithms retained for backwards compatibility and historical value</td>
<td>MD2, MD4, MD5, Panama Hash, DES, ARC4, SEAL 3.0, WAKE, WAKE-OFB, DESX (DES-XEX3), RC2, SAFER, 3-WAY, GOST, SHARK, CAST-128, Square</td>
</tr>
</tbody>
</table>
So…

- Try it!

Next time

- Find an elliptic curve suitable for cryptographic use
- Compute pairings by use of libraries