Chapter 16
A Software Library for Multi Precision Arithmetic

Kannan Balasubramanian
Mepco Schlenk Engineering College, India

Ahmed Mahmoud Abbas
The American University in Cairo, Egypt

ABSTRACT

The most prevalent need for multiple precision arithmetic, often referred to as “bignum” math, is within the implementation of public key cryptography algorithms. Algorithms such as RSA and Diffie-Hellman require integers of significant magnitude to resist known cryptanalytic attacks. As of now, a typical RSA modulus would be at least greater than $10^{309}$. However, modern programming languages such as ISO C and Java only provide intrinsic support for integers that are relatively small and single precision. This chapter describe the modules provided by one such library for the C Programming Language.

INTRODUCTION

The use of Public Key Algorithms like RSA require use of arbitrarily long numbers and arithmetic on those numbers. The existing programming languages cannot easily support the numbers handled by these algorithms. There are many tools for doing Multiple Precision arithmetic. This chapter discusses the use of one such tool called the ‘bignum’ math library. This library of routines can be easily integrated with any C program and the routines can be used to perform multiple precision arithmetic.

ARITHMETIC ON BIG INTEGERS

The most prevalent need for multiple precision arithmetic, often referred to as “bignum” math, is within the implementation of public key cryptography algorithms. Algorithms such as RSA (Rivest et al, 1978) and Diffie-Hellman (1976) require integers of significant magnitude to resist known cryptanalytic attacks. For example, at the time of this writing a typical RSA modulus would be at least greater than $10^{309}$.
10^{199}. However, modern programming languages such as ISO C (ISO/IEC 9899:1999) and Java (http://java.sun.com) only provide intrinsic support for integers that are relatively small and single precision.

The largest data type guaranteed to be provided by the ISO C programming language can only represent values up to $10^{19}$. On its own, the C language is insufficient to accommodate the magnitude required for the problem at hand. An RSA modulus of magnitude $10^{19}$ could be trivially factored on the average desktop computer, rendering any protocol based on the algorithm insecure. Multiple precision algorithms solve this problem by extending the range of representable integers while using single precision data types.

Most advancements in fast multiple precision arithmetic stem from the need for faster and more efficient cryptographic primitives. Faster modular reduction and exponentiation algorithms such as Barrett’s reduction algorithm, can render algorithms such as RSA and Diffie-Hellman more efficient. In fact, several major companies such as RSA Security, Certicom, and Entrust have built entire product lines on the implementation and deployment of efficient algorithms.

The benefit of multiple precision representations over single or fixed precision representations is that no precision is lost while representing the result of an operation that requires excess precision. For example, the product of two $n$-bit integers requires at least $2n$ bits of precision to be represented faithfully. A multiple precision algorithm would augment the precision of the destination to accommodate the result, while a single precision system would truncate excess bits to maintain a fixed level of precision.

A PACKAGE FOR MULTIPLE PRECISION ARITHMETIC

A package for using multiple Precision Arithmetic called the LibTomMath (Denis, 2006) that can be used with C programs is described. Appendix A (http://www.opensource.apple.com/source/Heimdal/Heimdal-247.7/lib/hcrypto/libtommath/tommath.h) lists all the function calls that can be made using this library. A multiple precision integer of $n$-digits shall be denoted as $x = (x_{n-1}, \ldots, x_1, x_0)_{\beta}$ and represents the integer $x \equiv \sum_{i=0}^{n-1} x_i \beta^i$. The term “mp_int” shall refer to a composite structure that contains the digits of the integer it represents, and auxiliary data required to manipulate the data. It is assumed that a “multiple precision integer” and an “mp_int” are assumed synonymous. The structure of mp_int is used in the LibTomMath package is given below:

```c
typedef struct {
  int used, alloc, sign;
  mp_digit *dp;
}mp_init;
```

- The \textit{used} parameter denotes how many digits of the array \textit{dp} contain the digits used to represent a given integer. The \textit{used} count must be positive (or zero) and may not exceed the \textit{alloc} count.
- The \textit{alloc} parameter denotes how many digits are available in the array to use by functions before it has to increase in size. When the \textit{used} count of a result exceeds the \textit{alloc} count, all the algorithms will automatically increase the size of the array to accommodate the precision of the result.
Related Content

Efficient Implementation of Digital Signature Algorithms
[www.igi-global.com/chapter/efficient-implementation-of-digital-signature-algorithms/188514?camid=4v1a](www.igi-global.com/chapter/efficient-implementation-of-digital-signature-algorithms/188514?camid=4v1a)

Cryptographic Voting Protocols
[www.igi-global.com/chapter/cryptographic-voting-protocols/188518?camid=4v1a](www.igi-global.com/chapter/cryptographic-voting-protocols/188518?camid=4v1a)

Securing Public Key Encryption Against Adaptive Chosen Ciphertext Attacks
[www.igi-global.com/chapter/securing-public-key-encryption-against-adaptive-chosen-ciphertext-attacks/188519?camid=4v1a](www.igi-global.com/chapter/securing-public-key-encryption-against-adaptive-chosen-ciphertext-attacks/188519?camid=4v1a)

Quantum Key Distribution Networks
[www.igi-global.com/chapter/quantum-key-distribution-networks/108026?camid=4v1a](www.igi-global.com/chapter/quantum-key-distribution-networks/108026?camid=4v1a)