بسم الله الرحمن الرحيم

Sudan University Of Science & Technology

College Of Graduate Studies

Big Integers And Their Use In Public Key Cryptosystem

By
Manar Ahmed Mohammed Hamza

Submitted In Partial Fulfillment Of The
Requirement For The Degree
M.Sc. In Computer Science

Supervisor
Dr. Mohsen Hassan Abd Allah Hashim

June 2005

Dedication

To my parents with love

Manar

Acknowledgment

First of all I should thank Almighty Allah for giving me strength, patience and help to complete this research.

I want to acknowledge people who contributed to completion the study and preparation of the research.

Actually, I'm indebted to my supervisor Dr. Mohsen Hassan Abd Allah Hashim to his valuable advice and keen guidance in preparing this project.

A gratefully acknowledgment must be extended to the participation of the staff in the department of computer science and information technology at Sudan University of Science and Technology for all technical help I had received.

Particular acknowledgement must be made to Miss Intesar for her contribution.

I would like to express my grateful thanks to my lovely family for their continuous encouragement and sentiments to complete this work.

ملخص البحث

في هذه الدراسة، تم تعريف نوع جديد من أنواع البيانات (*bigint*)، و صمم class باستخدام لغة C++ لتعريف هذا النوع الجديد و الذي يمكنه من تمثيل أي حجم من الأرقام، و هذا النوع (class) من أنواع البيانات قادر على إجراء العمليات الرياضية على الأرقام بمختلف الأحجام.

تم اختيار القوائم المتصلة ثنائية الاتجاه كوحدة تخزينية لتعريف نوع البيانات (*bigint*) .أي رقم في هذا النوع سيتم تمثيله بالأساس *1000* أي أنه سيتم تخزين ثلاث خانات في كل قائمة (عقدة).

و بالطبع سوف يتم تعريف و تحميل العمليات الرياضية و العلائقية لهذا النوع (*bigint*). تم تعريف و تكييف العديد من الدوال ذات الصلة بنظرية الأرقام.

هذا النوع (*bigint*) له أهمية في إجراء العديد من العمليات الرياضية و تجربتها لحساب الأرقام الكبيرة بإتقان . هذه العمليات الحسابية مطلوبة في علم التشفير اللامتماثل .

التقنيات الحديثة لعلم التشفير تعتمد على إمكانية تحديد أرقام أولية كبيرة ، و بالتالي ستتضمن هذه الدراسة طرق اختبار الأعداد الأولية.

تم اختيار خوارزمية ال RSA لاختبار و تطبيق هذا النوع الجديد من أنواع البيانات.

Abstract

In this study a *bigint* data type has been implemented and a C++ class is designed to represent an arbitrary sized numbers. It is a complete class capable of performing arithmetic on numbers of any size.

Double Linked List is selected as a storage structure in defining the data type *bigint*. In this class each integer is represented as a string of radix 1000 digits, three digits per node.

Certainly, arithmetic and relational operators are defined and overloaded for *bigint* class. Also, several functions related to number theory has been defined and adapted to the new data type.

This class is important to do multiple precision arithmetic to work out with large natural numbers calculation. This type of calculation is required in public key cryptography calculations.

Modern techniques of cryptography rely on the ability to determine primality for very large integers. Thus the appropriated primality testing mechanism is included.

An RSA public key cryptosystem represents an excellent case study to test *bigint* class.

Table of contents

Table of tables:

Table of figures

Introduction

The computational demands of modern cipher systems go around arithmetic of large integers. These days, that means several hundred decimal digits. In order to try out some of these cryptographic algorithms, we will need a facility that can do these computations. The security of the encryption scheme depends entirely on the difficulty of showing some mathematical problems. Factorizing large numbers is an example.

In this project we had created a fast, portable, and well documented class called *bigint*, with an RSA public key cryptosystem application.

Publicly available data type on large integers usually suffer from bad performance and limitation in the smallest and largest integers that it can represented. We present a flexible *bigint* class, in the development of which great care has been taken of speed, maintainability and portability of all available functions. It has been written in C++ code, a standardized high level language, providing both the ease of maintaining your program and the possibility of porting it to other platforms, as well as speed by means of advanced optimizations. [39]

The current functionality of the class can be grouped into the following categories: Input and Output large numbers, arithmetic operations (addition, subtraction, multiplication, squaring, integer division, modular) relational operations (greater than, greater than or equal, small than, small than or equal, equal to, not equal , assignment), greatest common divisor, modular inverse, converting to binary representation and high-level arithmetic (modular exponentiation, random number generation, generation of primes, 'strong' primes, and RSA key sets). [39]

Double linked list represents excellent storage structure for large numbers, any number will be store in a separate list, double linked list seems appropriate because we need to traverse the lists in both side. [39]

For operations on arbitrarily large, unsigned integers a representation in radix b notation is used, where b can be in principle any integer >= 2. That is, an arbitrary nonnegative integer x is represented as a sequence of radix b digits (x_0, x_1, x_2, x_3 ..., x_{k+1}), where [39]

$$x = \sum_{i=0}^{k-1} x_i b^i , \quad 0 \le x_i < b \quad \text{for } i = 0, 1, \ldots, k\text{-}1$$

The foundation of cryptography is essentially large integers. All the mathematics involved to help scramble the information is based on highly involved manipulations of large integers. Often the mathematics deals with special classes of integers including prime numbers.

Since the advent of public-key cryptography in 1975 many secure public-key algorithms have been proposed. An important subset is formed by the algorithms based on modular arithmetic with large unsigned integers, like Diffie-Hellman, RSA, ELGamal, Guillou-Quisquater, Schnorr, DSA. Our case study is RSA public key cryptosystem. [39]

A public key cryptosystem require two related, complementary keys. You can freely distribute one key, the public key to your friends, business associates, and even your competitors. You will maintain the second key, the private key, in a secure location (on your computer) and never release it to anyone. The private key unlocks the encryption that the public key creates. [2]

The public key protocol effectively eliminates the need for the secure channels required by conventional single-key cryptosystems. Two groups of individuals – Rivest, Shamir, and Adleman; and Diffie and Helman – designed the fundamental algorithms governing cryptography within a public-key cryptosystem. We will see RSA implementation in later chapters, because we will apply our new data type in its algorithm. [2]

Research problem:

In this research, we will be dealing with the most important problem. It's the limitation of the integer data type provided by programming language especially C/C++.

We've learned that computers, however powerful, are not infinitely precise in their calculations. For example, the computers with which you are most familiar (your home PC, for example, or the computers used in the labs) cannot represent integers with more than about 20 digits in them. For most practical applications this is simply a fundamental limitation of the computer hardware. [55]

A more pernicious problem with integer operations in C or C++ is that overflows and other errors in fundamental arithmetic operations often go undetected, especially with unsigned types. You can implement your own classes for "safe" ordinal or integer arithmetic. [55]

There are Varieties of integers. Many processors and programming languages directly supported four sizes and two kinds of integers producing eight integer data types that you should understand, all listed in Table 1.

Table 1: The eight common integer data types (language-independent)

Description	**Size**	**Range**
byte ordinal	1 byte, 8 bits	0 to 255
short ordinal	2 bytes, 16 bits	0 to 65,535
Ordinal	4 bytes, 32 bits	0 to 4,294,967,295
long ordinal	8 bytes, 64 bits	0 to 18,446,744,073,709,551,615
byte integer	1 byte, 8 bits	-128 to 127
short integer	2 bytes, 16 bits	-32,768 to 32,767
Integer	4 bytes, 32 bits	-2,147,483,648 to 2,147,483,647
long integer	8 bytes, 64 bits	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Table 1's list of types describes unsigned integer types, which don't allow negative numbers, as *ordinal* types. The C and C++ languages describe such types as *unsigned*.

Thereof, the integer basic data type provided by the C/C++ languages to represent integers has a limitation in the size of the data type in machine dependent and also limited in the smallest and largest integers that it can represent.

Actually, the need to represent big numbers exactly (and big integers, in particular) also arises in computer security. Modern techniques of cryptography rely on the ability to determine primality for very large integers (integers with hundreds of digits), and approximations just won't work in this case. Here, we need arbitrary-precision arithmetic.
This is just a fancy term meaning exact arithmetic for numbers of any size.

Research objectives:

The computational demands of modern cipher systems around ordinary arithmetic need large integers. These days, that means several hundred decimal

digits and unlimited digits is what we looking for it. In order to try out some of these cryptographic algorithms, we will need a facility that can do these computations.

There are more applications that are need large numbers for more secure. Such as: messages to be exchanged among an extensive range of applications involving the Internet, intelligent network, cellular phones, ground-to-air communications, electronic commerce, secure electronic services, interactive television, intelligent transportation systems, Voice Over IP and others.

So, we will create a C++ class (*bigint*) that represents numbers with unlimited size, for solving a problem we need to overload the natural operators: addition, subtraction, multiplication, division and modulation so that they will perform special operations related to the new class that we had created.

There are many algorithms related to number theory we overloaded such as obtain the greatest common divisor by using Euclidean Algorithm, check the prime number, inverse modulus using Extended Euclidean Algorithm and other functions.

This project help programmer to represent large numbers in any cryptographic application, speedup the encryption and decryption operations, and make sure it is simple to understand, run give fast computation and a high precision arithmetic operations result .

Also, one of the main objectives of this study is to provide our own created data type that will be used in cryptography. Since using other's created data type might affect secrecy.

Introducing chapters:

Here is the demonstration of whole chapters in this research:

Chapter 1: Background and literature review:

This chapter gives an overview of this project. It explains briefly how we will define the new data type, overloading operators, case study and literature review.

Chapter 2: Designing *bigint* class and overloading operators:

It concerns with defining *bigint* class in details and why we need large numbers. All overloaded operators will be demonstrated.

Chapter 3: Number Theory:

Here is an explanation of integer's properties. We will discuss some basic number theory such as obtaining the greatest common devisor, modular arithmetic, modular exponentiation, inverse modulo and primality tests to check if number is prime or not.

Chapter 4: RSA public key cryptosystem:

It is an implementation of this project. It demonstrates some terminologies related to cryptosystem, types of cryptosystem and RSA algorithm.

Chapter 5: Results:

This chapter shows whole results we had obtained from this study.

Chapter 6: Conclusion and Recommendations:

Here is the essence and counsel of this project with some recommendations.