Abstract: One of the most popular public key cryptosystems RSA, relies on the fact that it is computationally difficult to factor a large integer into its component prime integers. The Number Field Sieve algorithm is the fastest known method for factoring large integers. The two most time consuming steps in NFS algorithm are Sieving and Linear algebra. In this paper, we present an implementation of Sieving step using a radically new system which was designed to solve computationally hard problems in algebra, number theory, geometry, called Magma.

1. INDEX TERMS:

Factorization, Number Field Sieve, Sieving, Magma.

2. INTRODUCTION:

Factoring a positive integer \( n \) means finding positive integers \( x \) and \( y \) such that the product of \( x \) and \( y \) equals to \( n \), such that both \( x \) and \( y \) are greater than 1. Such \( x \) and \( y \) are called factors of \( n \), and \( n = x \cdot y \) is called factorization of \( n \). Factorization of a composite number and prime factorization are the two main factorizations possible. Factorization of a composite number is not necessarily unique, but prime factorization of a number, written as a product of prime numbers is unique. Factoring a composite integer is believed to be a hard problem; this is evident from the fact that there is no fast and practical factoring algorithm, which can factor a large number.

The security of one of the most popular public key cryptosystems, RSA relies on the supposed difficulty of factoring. This relation between factoring and cryptography is one of the main reasons why people are interested in evaluating the practical difficulty of the integer factorization problem and started to keep track of the largest numbers factored so far. RSA relies on the fact that it is computationally difficult to factor a large integer into its component prime integers. If an efficient algorithm is developed that can factor any arbitrarily large integer in a reasonable amount of time, the security value of the RSA system would be nullified. When keys are generated, efficient algorithms are used to generate two very large prime numbers and multiply them together. The person who generated the keys knows these two numbers, but everyone else knows only the product. The product contains enough information to encrypt a message to the person; the two primes allow the recipient to decrypt it. There is no known way to decrypt it without using the primes, but by factoring, we can extract the two prime factors from the product and break the encryption.

At the time when RSA was invented, factoring integers started with as few as 80 decimal digits which was intractable; all known algorithms were either too slow or required the number to have a special form, this made, 256-bit keys relatively secure. The first major breakthrough was quadratic sieve, a relatively simple factoring algorithm which can factor numbers up to 100 digits and more. It's still the best known method for numbers under 110 digits or so; but for factoring large numbers, the Number Field Sieve is the fastest known algorithm (NFS) so far.
3. NUMBER FIELD SIEVE:

The GNFS also has fundamental similarities to simpler and previously known algorithms. At its core, as with the continued fraction method and the quadratic sieve, its goal is to find a congruence of squares. If \( N \), the number to be factored, is composed of two prime factors, each solution to \( x^2 \equiv y^2 \mod N \), has a 50% chance of producing a non-trivial factorization of \( N \). Whereas previous algorithms seek a congruence of squares through rather elementary means, the GNFS uses the theory of algebraic rings to find a congruence of squares in a more efficient way.

The algorithm consists of 5 main steps which are mutual dependent and can not be made in parallel, but some of the steps can be parallel internally.

A. Polynomial selection:

Selecting a usable polynomial is not difficult, but a good polynomial is hard to find. The yield of a polynomial refers to the number of smooth values it produces in its sieve region. There are two main factors which influence the yield of the polynomial, they are size and root properties. A polynomial is said to be good if it has a good yield i.e. has a good combination of size and root properties.

A polynomial selection step consists of the following steps:

i. Identify a large set of usable polynomials.

ii. Using heuristics remove obviously bad polynomials from the set.

iii. Do small sieving experiments on the remaining polynomials and choose the one with the best yield.

B. Factor bases:

The algorithm needs a well defined domain to work in, and this is specified with the factor bases. Factor base is a mathematical tool commonly used in integer factorization, more specifically algorithms involving extensive sieving of potential factors.

There are mainly two factor bases: Rational Factor Base (RFB), Algebraic Factor Base (AFB).

RFB: RFB consists of all the primes \( p_i \) up to some bound and we also store \( p_i \mod m \), so the rational factor base consists of pairs \( (p, p \mod m) \) i.e.

\[
RFB = (p_0, p_0 \mod m), (p_1, p_1 \mod m) \ldots .
\]

AFB: AFB consists of pairs \( (p, r) \) such that \( f(r) = 0 \mod p \) i.e.,

\[
AFB = (p_0, r_0),(p_1, r_1) \ldots .
\]

The size of the algebraic factor base should be 2-3 times the size of the rational factor base. In general, an integer is said to be \( y \)-smooth if all of its prime divisors are less than or equal to \( y \).

C. Sieving:

The purpose of sieving step is to find usable relations, i.e. elements \( (a, b) \) with the following properties

i. \( \gcd (a, b) = 1 \).

ii. \( a + bm \) is smooth over the rational factor base

iii. \( b^{\deg(f)}f(a/b) \) is smooth over the algebraic factor base

Finding elements with these properties can be done by various sieving methods like classical line sieving or faster lattice sieving.

D. Linear algebra(Matrix step):

From the sieving step we have a list of \( (a, b) \)'s which are RFB smooth and AFB smooth and we have to find a combination of elements from the relation set which has a product that is a square. For a number to be a square the elements in its factorization must have an even power and this property is what is used to find elements that are squares. This method involves solving a system of linear equations and this can be done by building a matrix. The matrix consists of factorization over RFB, and AFB. Finally this matrix is put on reduced echelon form and from this solutions are derived which yield squares. It is fast and easy to implement but this has a drawback, i.e. the matrix is very large, memory usage becomes a problem. Both in theory and practice the algorithm is faster for large instances.
E. Square root:

Products which are squares and can lead to a trivial or non-trivial factor of \( n \), can be obtained from linear algebra step. Rational square root \( Y \) and algebraic square root \( X \) is needed from the solution. The rational square root is trivial, although the product of the solution from the linear algebra step is large, but it is still an integer. The square root of an algebraic integer is the most complex part of the algorithm, but not the time consuming. Finding the square root of an algebraic integer is equivalent to finding the square root of a polynomial over an extension field.

In our project, we deal with the Sieving step.

4. SIEVING:

The dictionary meaning of sieving is: “pick out, select, or choose from a number of alternatives.” Similar to its original meaning, the main purpose or the actual task of the sieving step in NFS is to find usable relations, i.e. elements \((a,b)\) from many available relations. The sieving step results in a set \( R \) of relations.

The sieving step is not the theoretically most complex part of the algorithm, but it is the most time consuming part because it iterates over a large domain with some expensive calculations. In general, optimization of the sieving step will give the biggest reduction in actual running time of the algorithm. The relations \((a,b)\) that are determined in this step have the following properties*:

i. \( \gcd(a,b) = 1 \)

ii. \( a + bm \) is smooth over the rational factor base.

iii. \( b^{\deg(f)} f(a/b) \) is smooth over the algebraic factor base.

where \( f(x) \) is a monic irreducible polynomial of degree \( d \), which belong to \( \mathbb{Z}[x] \), found by the polynomial step, and \( m \) is the root of the polynomial \( f(x) \) such that \( f(m) \equiv 0 \mod n \), where \( n \) is the number to be factored.

Finding elements (relations) with these properties can be done by various sieving methods like “Line Sieving” or “Lattice sieving”.

We deal with the classical Line sieving procedure: Line sieving is done by fixing \( b \) and sieving over a range of \((a,b)\) for \( a \in [-u:u] \) (sieving interval for \( a \)). This means that the rational norm is calculated for all values of \((a,b)\) and then divided with elements from the RFB and the entries with a resulting 1 are smooth over the RFB and the same procedure is done for the algebraic norm over AFB.

It is not necessary to test whether each element on a sieving line factors completely over the factor bases by trial division with each element from the RFB and AFB. Instead it can be done in the following way:

A. Sorting few pairs \((a,b)\) from all possible pairs \((a,b) \in [-u,u], 0 < b \leq u\), which are ‘likely’ to be relations. This is done by fixing \( b \) to its initial value \((b=1)\), and starting from the lowest value for \((a=-u)\), an initial pair \((a,b)\) which is ‘likely’ to be a relation is found out and other ‘possible relation’ pairs are determined by adding \( \log_2 p \) \((p \in \text{RFB})\) to the location of the initial \((a,b)\) on the sieving line. This procedure reduces the number of pairs for which the three properties* are to be checked and thus increasing the speed of the sieving process.

B. Finally the required Relations are determined by testing the three properties* on the sorted pairs from the sieving line.

The above figure shows the pairs of elements which are likely to be Relations on the sieving line. Sieving line consists of all possible pairs \((a,b)\) for each fixed value of \( b \)’ and \( a \in [-u:u] \).
INPUT & OUTPUT:
For the implementation of the sieving step, the inputs and outputs are as given below:
Inputs: RFB, AFB, polynomial f(x), root m of f(x)\(\text{mod}\ n\), integer(u) (for defining sieving interval).
Output: List of Relations: \{(a_0, b_0)\ldots\ldots(a_t, b_t)\}

ABOUT MAGMA:
Magma is a radically new system designed to solve computationally hard problems in algebra, number theory, geometry and combinatorics. Magma provides a mathematically rigorous environment for computing with algebraic and geometric objects. Magma is both computer algebra system and a programming language. The syntax of the language resembles that of many well known programming languages. The specialty about magma is the provision of mathematical data types such as groups, rings, fields, sets, sequences and mappings, together with a large collection of functions for performing standard tasks in algebra. Information about the magma elements is stored in a mathematically powerful way, making advanced symbolic algebraic computation feasible.
The commands in Magma are interpreted rather than compiled for dynamic interaction in a shell. It makes available a huge library of mathematical data structures together with high performance algorithms for their manipulation. The code can be written in Magma language as packages can be attached by users at startup time to expand on functionality.

Commands in Magma are known as statements. Each complete statement must finish with a semicolon ‘;’. Whenever Magma is ready to receive a statement from the user, it displays a prompt symbol ‘>’ on the left of the input line.

Printing Expressions: An expression is a piece of Magma code which may be evaluated to return a value. The most direct way to determine the value of an expression is to print it, using a print statement.

print expression;

The print statement is able to evaluate and print several expressions, provided they are separated by commas.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Magma</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x+y</td>
<td>Sum of elements x and y</td>
</tr>
<tr>
<td>2</td>
<td>-x</td>
<td>Additive inverse of x</td>
</tr>
<tr>
<td>3</td>
<td>x-y</td>
<td>Difference of elements x and y</td>
</tr>
<tr>
<td>4</td>
<td>n*x</td>
<td>x added to itself n times</td>
</tr>
</tbody>
</table>

Table 1: Additive Operators

<table>
<thead>
<tr>
<th>S.No</th>
<th>Magma</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x*y</td>
<td>Product of elements x and y</td>
</tr>
<tr>
<td>2</td>
<td>x^-1</td>
<td>Multiplicative inverse of x</td>
</tr>
<tr>
<td>3</td>
<td>x/y</td>
<td>x (\times) (y^-1)</td>
</tr>
<tr>
<td>4</td>
<td>x \text{div} y</td>
<td>Quotient when x is divided by y</td>
</tr>
<tr>
<td>5</td>
<td>x \text{mod} y</td>
<td>Remainder when x is divided by y</td>
</tr>
</tbody>
</table>

Table 2: Multiplicative Operators
**Function Calls:** Magma contains a very large number of standard predefined functions and procedures, known as intrinsics. An intrinsic function is evaluated for specific values of its arguments by typing its name followed by the list of arguments, separated by commas and enclosed within parentheses. If the intrinsic function does not have any arguments, the parentheses must still appear. Almost all the intrinsics begin with a capital letter. The process of applying an intrinsic function to a particular set of values is often referred to as *invoking or calling* the function.

**Printing Text:** It is often desirable to print some explanatory text together with a value. The text must be enclosed between double quotation marks ("… "). This can also be done using *print* statement. The object formed by enclosing characters in double quotation marks is called a *string*.

**Identifier:** The main role of an identifier is that they allow the construction of more general program statements than those that can be formed using constants alone. Because of their utility, it is usual for many identifiers to be used in a large Magma program. Identifier names are not restricted to a single letter, in fact choosing a longer and meaningful name, so as to remind the reader of the value named by this identifier is a good idea. An identifier may be constructed from letters, digits or underscore. There are traps to avoid choosing identifier.

i. A magma reserved word such as print or mod should not be used as an identifier.

ii. Magma distinguishes between upper and lower case letters.

*ShowIdentifier(),* prints a list of all the identifiers that are currently assigned. *ShowValues(),* prints all these identifiers with their values.

If a magma session has involved the creation of very large objects, or large numbers of small objects, process space may be exhausted. Memory may be freed using the delete statement to remove unwanted data.

**Assignment:** Initially if an identifier is assigned a value and then another value is assigned to the same identifier, then the old value is lost and cannot be recovered.

A statement of the form *identifier: = identifier o expression;* can be written as

\[ \text{identifier } o: = \text{expression}; \]

where o is an operator which signifies the function it performs.

To access the values returned by a multi value assignment a multiple assignment statement is provided. The syntax is:

\[ \text{identifier, … , identifier : = multiple value expression;} \]

Often Magma expressions include the same sub expression more than once, it would be faster to assign that expression to a temporary identifier, this can be accomplished with the where construction. The syntax is:

\[ \text{expression where identifier is expression} \]

**Comments:** A comment is included in a computer program to describe its purpose and its internal logic. Comments are ignored by the computer. Magma ignores anything between the symbols /* and */ and the symbol // on a line will be ignored.

**Recalling previously printed values:** Magma provides a facility for recalling values that have been obtained by evaluating expression lists in print statements. The list of values displayed by the most recent print statement corresponds to $1$. The list of values in the next most recent print statement corresponds to $2$ and so on….

**Loading input from a file:** Short text of code can easily be typed directly into Magma. For long text, it is usually more efficient to create blocks of code as text files external to Magma. The contents of the file may be loaded into Magma by typing:

```
load "filename";
```

Magma also provides a package mechanism, which automatically loads the files. Given a file containing the users definitions of functions and procedures in a special syntax, it is able to incorporate these routines into Magma as user intrinsics.
### S.NO | Function | Syntax
---|---|---
1 | Assignment statement | IDENTIFIER := EXPRESSION;
2 | For statement | for IDENTIFIER in DOMAIN do statement; ……… statement; end for;
3 | Factorization | Factorization(n); Or Factorization(n); where n is an integer
4 | If | if CONDITION then statements; end if;
5 | Break | break; or break IDENTIFIER;
6 | Include | Include(~S, x);
7 | Logarithm | Log(b, x); where b is the base and x is the non zero element.
8 | Exclude | Exclude(~S, x);
9 | Evaluate | Evaluate(f, p);
10 | GCD | GCD(D1,D2);
11 | Rounds up the value to the nearest integer. | Round(x);
12 | Truncate function discards the decimal part of a number. | Truncate(x);

### Table3: Functions

**Booleans:** One of the basic structures available in Magma is the Boolean structure. It contains two values: true and false.

The Boolean operator corresponds fairly closely to the ordinary meanings of the corresponding words, except that or is inclusive, where as xor is exclusive.

The above description and details about Magma are only introductory steps which help to get an overview and a basic idea about Magma language. Also most of steps in our Sieving algorithm are implemented using these basic rules and syntax. More description, functions and characteristics of magma can be found in Handbook of Magma [].

### S.NO | Operator | Symbol | Its Function
---|---|---|---
1 | eq | x eq y | true iff x is equal to y
2 | ne | x ne y | true iff x is not equal to y
3 | lt | x lt y | true iff x is less than y
4 | le | x le y | true iff x is less than or equal to y
5 | gt | x gt y | true iff x is greater than y
6 | ge | x ge y | true iff x is greater than or equal to y

Table 4: Boolean Expressions

### S.NO | Operator | Symbol | Its Function
---|---|---|---
1 | not | not a | true iff a is false
2 | and | a and b | true iff both a and b are true
3 | or | a or b | true iff at least one of a and b are true
4 | xor | a xor b | true iff exactly one of a and b is true

Table 5: Boolean Operators

### 7. CONCLUSION:

Factoring of large numbers has always been a great interest for cryptanalysts. As part of the interest, the popular fastest known algorithm to factor large integers, GNFS, has been implemented using many programmable languages so far. In this paper, we implemented one of the most time consuming steps of NFS, the Sieving step using Magma, which is both a computer algebraic system and a programmable language.
8. REFERENCES:


