A New Deterministic RSA-Factoring Algorithm

Sattar J. Aboud¹ and Evon M. Abu-Taieh²

¹Graduate College for Computing Studies, Amman Arab University for Graduate Studies, Amman, Jordan
²Computer Information System Department, The Arab Academy, Amman, Jordan

Received 26 April 2004 ; accepted 15 June 2005

Abstract: The security of many cryptography techniques depends upon the intractability of the integer-factoring problem. However, in the recent years there has been a great deal of progress in the art of factoring, relaying mostly on non-deterministic methods. This research proposes a new deterministic factoring algorithm, that factors RSA \( n = p \times q \), the algorithm running time relays on the number of digits of \( n \) rather than the value of \( n \). The nature of the problem of factoring based on time, complexity and storage required. The proposed algorithm works on solving these problems by using the idea of long multiplication to limit the possible values of \( p \) and \( q \). In order to eliminate the storage problem, depth-first search was used with recursive implementation. In addition, the paper discussed the analysis of the proposed algorithm with their running time and complexity. Finally, the paper concludes with future work improvements to the algorithm.

Key words: Cryptography; RSA Factoring; Schoolboy Multiplication; Counting and Probability.

Introduction

Factoring a positive integer \( n \) means find its prime factors \( p \) and \( q \) such that the product of \( p \) and \( q \) equals \( n \), \( p \) and \( q \) are called factors and \( n \) is called composite. Factoring positive composite integer is believed to be a hard problem and most of the popular public key cryptosystems relay on this fact [1], but surely not impossible since the RSA-140 was factored using number Field Sieve [2], the RSA-120 was factored using Quadratic Sieve [3]. The RSA-155 was factored in 1999, The RSA 160 was factored in April 2003, and the RSA-576 was factored in December 2003 [4]. This relation between factoring and cryptosystems is one of the main reasons why people are interested in factoring algorithms [5].

In this paper, a proposed algorithm attempts to factor \( n \), which is a product of two primes \( p \) and \( q \), where \( p \) and \( q \) are roughly of the same size. The proposed algorithm could be implemented as hardware and the result will be faster and much better than the one currently used. Before describing the proposed factoring algorithm, we need to discuss some facts concerning a basic arithmetic operations and number theory. The two fundamentals required are schoolboy multiplication and the counting and probability.

The remaining of the paper is as follows. Section 2 we introduce schoolboy multiplication method, which is the simplest method, can compute the 2m-digit product (n). The counting and probability method is represented in section 3. The proposed algorithm id described in section 4. Section 5 presents the analysis of suggested algorithm. Section 6 concludes with paper notes and references.
Schoolboy Multiplication

The simplest multiplication method known is the schoolboy multiplication method. The Schoolboy multiplication algorithm can compute the 2m-digit product \(n = <n_{2m}, n_{2m-1}, \ldots, n_1>\) of two m-digit numbers \(p = <p_m, p_{m-1}, \ldots, p_1>\) and \(q = <q_m, q_{m-1}, \ldots, q_1>\). Yet this method is the slowest; it requires \(O(m^2)\) operations. Figure 1 shows the algorithm.

1. create a matrix \(x\) where rows = \(m\) and columns = \(2m-1\)
2. place \(p_i\) above column \(i\), \(i = 1\) to \(m\)
3. place \(q_j\) above \(p_i\), \(j = 1\) to \(m\)
4. compute all \(x_{i+j-1} = p_i \times q_j\), \(i = 1\) to \(m\), \(j = 1\) to \(m\)
5. set the first carry \(r_0 = 0\)
6. Sum each column \(c\) in matrix \(x\), where \(c = 1\) to \(2m - 1\)
   a. from right to left for each column in matrix \(x\)
   b. set \(n_c\) = least significant digit (summation of column \(c\) + previous carry \(r_{c-1}\))
   c. set the carry \(r_c = \text{summation of column } c \mod 10\)
7. set \(n_{2m} = r_{2m-1}\)

**Fig 1.** Schoolboy multiplication algorithm

In other words, we examine the digits of \(p\), from \(p_1\) to \(p_m\). For each digit \(p_i\) with a value greater than 0, we add \(q \times p_i\) into the product, but shift left by 1 position. For each digit \(q_j\) with a value of 0, we add in 0. Thus letting \(x_{i+j-1} = q_j \times p_i\), we compute \(n = p \times q = \Sigma x_{i+j-1}\) where \(i = 1\) to \(m\) and \(j = 1\) to \(m\). Each term \(x_{i+j-1}\) is called partial product positioned in column \(i+j-1\). When adding the partial products we must start \(x_1, x_2, \ldots, x_{2m-1}\). Here we must introduce the notion of the Carry, once we add what is in column \(i\) we take the least significant digit and store it in \(n_i\), then set the carry \(r_i\) to the sum of column \(x_i\) div 10. There are \(m^2\) partial products to sum, with digits in positions 1 to \(2m-1\). The carryout from highest digit yields the final digit in position \(2m\).

Counting and Probability

The counting theory tries to answer the question “How Many?” without actually enumerating how many. As explained in many statistics books [6] a set of items that we wish to count can sometimes be expressed as a union of disjoint sets or Cartesian product of sets.

The rule of sum say that the number of ways to choose an element from one of two disjoints sets is the sum of cardinalities of the two set. That is, if \(A\) and \(B\) are two finite sets with no members in common, then \(|A \cup B| = |A| + |B|\).

The rule of product says that the number of ways to choose an ordered pair is the number of ways to choose the first element times the number of ways to choose the second element. That is, if \(A\) and \(B\) are two finite sets, then \(|A \times B| = |A| \times |B|\).

The Cartesian product of two sets \(A\) and \(B\), denoted \(A \times B\), is the set of all ordered pairs such that the first element of the pair is an element of \(A\) and the second is an element of \(B\). More formally: \(A \times B = \{(a, b) : a \in A \text{ and } b \in B\}\). When \(A\) and \(B\) are finite sets, the cardinality of their Cartesian product is: \(|A \times B| = |A|^{|B|}\). The Cartesian product of \(n\) sets \(A_1, A_2, \ldots, A_n\) is a set of \(n\)-tuples.
A_1 \times A_2 \times \ldots \times A_n = \{(a_1, a_2, \ldots, a_n) ; a_i \in A_i; \ i = 1, 2, \ldots, n\}, whose cardinality is |A_1 \times A_2 \times \ldots \times A_n| = |A_1|^{|A_2| \times \ldots \times |A_n|} if all sets are finite. We denote an n-fold Cartesian product over a single set A by the set A^n = A \times A \times A \times \ldots \times A. Whose cardinality is |A^n| = |A|^n if A is finite. An n-tuple also is viewed as a finite sequence of length n [6].

Based on the previous quote one can define a single finite set digits = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}. Let p be the m-fold Cartesian product over the set digits. Then one can denote P = digits^m = digits \times digits \times \ldots \times digits. Therefore, P = \{(p_1, p_2, \ldots, p_m) ; p_i \in Digit, i = 1, 2, \ldots, m\} and |P| = |\text{digits}|^m. Based on the previous formula, one can look at any integer number of m-digits as an m-tuple. For example, the long integer p = 123789456 of 9-digits can be the 9-tuple \((1, 2, 3, 4, 5, 6, 7, 8, 9)\) where \(m=9\) and \(p \in P\). If one wants to sequentially produce \(p\) integer of m-digits, it will take \(10^m\) iteration in for-next nested-loops.

The Proposed Factoring Algorithm

In the proposed factoring algorithm the purpose is to factor the RSA \(n\) where \(n\) is a large composite number (\(n = p \times q\) where \(p\) and \(q\) are large prime numbers and \(m\) the number of digit of \(n\)) and \(n = (n_m, n_{m-1}, n_{m-2}, \ldots, n_1)\). The prime number \(p = (p m_2, p_{(m-2)}-1, p_{(m-2)}-2, \ldots, n_1)\) and the prime number \(q = (q m_2, q_{(m-2)}-1, q_{(m-2)}-2, \ldots, q_1)\).

For example, if \(n = 106679\), then \(m = 6\) (\(m\) is the number of digits of \(n\)), \(p = 107\) and \(q = 997\) then one can write \(n\) as follows: \(n_6 = 1, n_5 = 0, n_4 = 6, n_3 = 6, n_2 = 7, n_1 = 9\) or \(n\) is 6-tuple \((1, 0, 6, 6, 7, 9)\) and write \(p\) as follows:

\[p = (p_6, p_5, p_4, p_3, p_2, p_1) \quad \text{where} \quad p_6 = 1, p_5 = 0, p_4 = 7, \quad \text{and} \quad p = 3 \text{-tuple} \quad (1, 0, 7)\]

and write \(q\) as follows:

\[q = (q_6, q_5, q_4, q_3, q_2, q_1) \quad \text{where} \quad q_6 = 9, q_5 = 9, q_4 = 7, \quad \text{and} \quad q = 3 \text{-tuple} \quad (9, 9, 7)\]

Each digit \(p_i\) ranges from 0 to 9 for \(i = 1, 2, 3, \ldots, m/2\); and each digit \(q_i\) ranges from 0 to 9 for \(i = 1, 2, 3, \ldots, m/2\). But since \(p\) and \(q\) are primes and each prime is an odd number then one can say that \(p_i \in \{1, 3, 7, 9\}\) and \(q_i \in \{1, 3, 7, 9\}\). In other words, the least significant digit of each number \((p\ or\ q)\) that belongs to the set of prime numbers must belong to the set \(\{1, 3, 7, 9\}\).

The proposed factoring algorithm is made of two procedures. The first procedure \((\text{Producer})\) sequentially produces all the \(p\) and \(q\) \((i=1, 2, \ldots, m/2)\) that can produce the first half of \(n\). The second procedure \((\text{Eliminator})\) works on eliminating the \(p_i\) and \(q_i\) that will not produce the second half of \(n\). The result will be always one \(p\) and one \(q\) that when multiplied will produce \(n\).

The first procedure, \((\text{Producer})\), works on producing all \(p_i\) and \(q_i\) that satisfy the following condition for all \(n_i\) where \(i = 1\) to \(m/2\): \(n_i + r_i = (\sum p_i^* q_i + r_{i-1})\), where \(i = 1\) to \(m/2\) and \(x = 1\) to \(i\) and \(j = i\) to \(1\); \(r_0 = 0\) to \(i = 81 + r_{i-1} \text{ div } 10\). The second procedure, \((\text{Eliminator})\), works on eliminating all \(p_i\) and \(q_i\) that do not satisfy the following condition for all \(n_i\) where \(i = m/2+1\) to \(m\): \(n_i + r_i = (\sum p_i^* q_i + r_{i-1})\), where \(x = (i-m/2)+1\) to \(m/2\) and \(J = m/2\) to \((i-m/2)+1\). In the following two sections both \((\text{Producer})\) and \((\text{Eliminator})\) will be explained.

Algorithm

The algorithm will do two major operations producing the \(p_i\) and \(q_i\) and carries \(r_i \quad (i = 1, 2, \ldots, m)\) that when multiplied will produce the first half of the \(n\). The second major operation will eliminate all \(p_i\), \(q_i\), and carries \(r_i\) that will not produce the second part of \(n\), while producing the rest of the carries \(r_i\) \((i = m/2+1,\)
If $n$ is an odd number, we will take the side of $n/2$ lower bound.

Figure 2 shows the algorithm.

1. Compute $m$ = number of digit of $n$
2. Set the carry $r_0 = 0$
3. Compute all maximum carry $r_i$ for $n_i$, where $i=1$ to $m$, $r_i = ((i*81+r_{i-1}) \text{ div } 10)$.
4. Sequentially search for $(p_1, q_1, r_1)$ such that $p_1*q_1+r_0 = n_1+r_1$, where $p_1 \in \{1,3,7,9\}$ and $q_1 \in \{1,3,7,9\}$ and $0 \leq r_1 \leq 8$ and store it in table write
5. For each $(p_i, q_i, r_i)$ in the table write
   for each $n_i$ where $i = 2$ to $m/2$
      i. sequentially search for $(p_i, q_i, r_i)$ such that $n_i+r_i = (\sum p_x*q_j+r_{i-1})$, where $r_i \leq \text{max}_\text{carry}_\text{of}_i$ and $x = 1$ to $i$ and $j = i$ to 1
      ii. Store in table $(p_i, q_i, r_i)$, $c = 1, 2...i$
6. For each $n_i$, where $i = m/2+1$ to $m$
7. Keep the entries from table write $(p_i, q_i, r_i)$ such that $n_i+r_i = (\sum p_x*q_j)+r_{i-1}$, where $x = (i-m/2)+1$ to $m/2$ and $j = m/2$ to $(i-m/2)+1$

**Fig 2. Producer Algorithm**

Example:
Let $n = 1411$ where $p = 13$ and $q = 83$ (of course to factor $n$ we pretend not to know $p$ and $q$)

**Step 1:** $m = 4$
**Step 2:** $r_0 = 0$
**Step 3:** Compute all maximum carry $r_i$ for $n_i$, where $i = 1$ to $4$, $r_i = ((i*81+r_{i-1}) \text{ div } 10)$. Table 1 shows the results.

<table>
<thead>
<tr>
<th>$i$</th>
<th>$r_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>17</td>
</tr>
<tr>
<td>3</td>
<td>26</td>
</tr>
<tr>
<td>4</td>
<td>35</td>
</tr>
</tbody>
</table>

**Step 4:** set $n_1 = 1$ and $r_1 = 0$ to 8, sequentially search for $(p_1, q_1, r_1)$ such that $p_1*q_1 = n_1+r_1$. Table 2 shows the results.

<table>
<thead>
<tr>
<th>$p_1$</th>
<th>$Q_1$</th>
<th>$r_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>2</td>
</tr>
</tbody>
</table>

**Step 5:** Sequentially search for $(p_2, q_2, r_2)$ where $p_1*q_2+p_2*q_1+r_1 = n_2+r_2$, where $n_2 = 1$ and $r_2 = 0$ to 17. Table 3 shows the result.
Step 6: Eliminate all rows in the table where \( q_2 \times q_2 + r_2 \neq n_3 + r_3 \times 10 \) where \( r_3 \leq 26 \). Table 4 shows the result.

### Table 3. Result of Step 5

<table>
<thead>
<tr>
<th>( p_1 )</th>
<th>( q_1 )</th>
<th>( r_1 )</th>
<th>( p_2 )</th>
<th>( q_2 )</th>
<th>( r_2 \leq 17 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>8</td>
<td>1</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>2</td>
<td>0</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>6</td>
<td>2</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>5</td>
<td>8</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>2</td>
<td>8</td>
</tr>
</tbody>
</table>

### Table 4. Result of Step 6

<table>
<thead>
<tr>
<th>( p_1 )</th>
<th>( q_1 )</th>
<th>( r_1 )</th>
<th>( p_2 )</th>
<th>( q_2 )</th>
<th>( r_2 \leq 17 )</th>
<th>( r_3 \leq 26 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td>2</td>
<td>1</td>
<td>8</td>
<td>6</td>
<td>10</td>
</tr>
</tbody>
</table>

**Producer Algorithm**

Initially this procedure will produce all \( p_1 \)s and \( q_1 \)s and \( r_1 \)s that produce \( n_1 \) and store these values in a file called `initial_write_file`. Figure 3 shows the algorithm.

```plaintext
Initial Producer
For each \( p_1 \) in \{1, 3, 7, 9\}
  For each \( q_1 \) in \{1, 3, 7, 9\}
    For \( r_1 = 0 \) to \( 8 \)
      If \( n_1 + r_1 \times 10 = p_1 \times q_1 + r_{i-1} \) then
        Store \( (p_1, q_1, r_1) \) in `initial_write_file`

Fig 3. Initial Algorithm Producer
```

This initial producer will produce all possible combinations of \( (p_1, q_1, r_1) \). For example if \( n_1 = 9 \), like in the previous example, then this code will produce the following values shown in table 5.

### Table 5. The possible values of \( p_1, q_1, \) and carry \( r_1 \) where \( n_1 = 9 \) (initial_write_file)

<table>
<thead>
<tr>
<th>( p_1 )</th>
<th>( q_1 )</th>
<th>( r_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>
Next, **Producer** will produce all \( p_i \)s and \( q_i \)s and \( r_i \)s that satisfy \( n_i \) = least significant digit \( \left( \sum \! p_x \cdot q_j \right) \) + \( r_{i-1} \), where \( i = 1 \) to \( m/2 \) and \( x = 1 \) to \( i \) and \( j = i \) to \( 1 \); \( r_i = 0 \) to \( r_i = 0 \) to \( \sum (i \cdot 81 \div 10) \). Needless to say one must make sure that no duplicates are produced. Compute all maximum carry \( r_i \) for \( n_i \), where \( i = 1 \) to \( m \), \( r_i = (i \cdot 81 + r_{i-1} \div 10) \). Figure 4 shows the algorithm.

### Fig 4. Producer Algorithm

**Eliminator Algorithm**

This procedure will only eliminate all the \( p_i \)s and \( q_i \)s that do not satisfy the following conditions: for all \( n_i \) where \( i = m/2+1 \) to \( m \). \( n_i + r_i \cdot 10 \) = least significant digit \( (\sum \! p_x \cdot q_j) + r_{i-1} \). Where \( x = (i-m/2)+1 \) to \( m/2 \) and \( j = m/2 \) to \( (i-m/2)+1 \). Figure 5 shows the algorithm.

### Fig 5. Eliminator Algorithm
Trace

To trace the previous algorithm, one must set an example of a smaller number $n = 1411$. The trace of the initial producer will produce the following result \((\text{initial_write_file})\). Tables 6 and 7 show the result.

**Table 6. A trace of initial producer algorithm**

<table>
<thead>
<tr>
<th>$n_1$</th>
<th>$p_1$</th>
<th>$q_1$</th>
<th>$r_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

The second trace of the producer algorithm

\[
\text{read_file} = \text{initial_write_file}
\]

\[
\text{stat\_index} = 3
\]

\[
m = 4
\]

**Table 7. A trace of the producer algorithm**

<table>
<thead>
<tr>
<th>$p_1$</th>
<th>$q_1$</th>
<th>$r_1 \leq 8$</th>
<th>$p_2$</th>
<th>$q_2$</th>
<th>$r_2 \leq 17$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>8</td>
<td>0</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>8</td>
<td>1</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>8</td>
<td>2</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>8</td>
<td>3</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>8</td>
<td>8</td>
<td>9</td>
<td>16</td>
</tr>
</tbody>
</table>

The third trace is for the eliminator algorithm

\[
i = 3 \text{ to } 4
\]

Number of records in read file = 13

\[
r_3 = 0 \text{ to } 1*81+17 \text{ div } 10 = 9
\]

Since the result is one record then the *answer* is shown in table 8.

**Table 8. Result of eliminator algorithm**

<table>
<thead>
<tr>
<th>$p_1$</th>
<th>$q_1$</th>
<th>$p_2$</th>
<th>$q_2$</th>
<th>$R_1$</th>
<th>$r_2$</th>
<th>$r_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>3</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

Analysis

In this section a proof is stated along with a discussion regarding the running time, complexity, and storage requirements. In addition to listing the advantages and disadvantages of this algorithm providing was also suggested some improvements on the algorithm.
Proof

To prove that \( n_i + r_i = \begin{cases} \sum_{i=1}^{m} p_x \cdot q_{j} & \text{where } j = m \text{ to } 1, \quad i \leq m \\ \sum_{x=i-m+1}^{m} px \cdot q_{j} & \text{where } j = 2 \cdot m \text{ to } i - m + 1, \quad i > m \end{cases} \)

\( m \) is the number of digits of \( p \) and \( q \) and \( r_i \) is the carry.

First, based on [6] one can define a single finite set \( \text{digits} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \). Let \( p \) be the \( m \)-fold Cartesian product over the set \( \text{digits} \). Then one can denote

\[ p = \text{digits}^m = \text{digits} \times \text{digits} \times \text{digits} \times \ldots \times \text{digits}. \]

\[ a \]

\[ q = \text{digits}^m = \text{digits} \times \text{digits} \times \text{digits} \times \ldots \times \text{digits} \]

\[ n = \text{digits}^{m+2} = \text{digits} \times \text{digits} \times \text{digits} \times \ldots \times \text{digits} \]

Therefore,

\[ P = \{ (p_1, p_2, \ldots, p_m) : p_i \in \text{digit}, \quad i = 1, 2, \ldots, m \} \]

\[ q = \{ (q_1, q_2, \ldots, q_m) : q_i \in \text{digit}, \quad i = 1, 2, \ldots, m \} \]

\[ n = \{ (n_1, n_2, \ldots, n_{m+2}) : n_i \in \text{digit}, \quad i = 1, 2, \ldots, m + 2 + 1 \} \]

and

\[ |p| = |\text{digits}|^m \]

\[ |q| = |\text{digits}|^m \]

Second, since \( n \) is product of two primes \( p \) and \( q \) [5] then

\[ n_i \in \{1, 3, 7, 9\} \]

\[ p_i \in \{1, 3, 7, 9\} \]

\[ q_i \in \{1, 3, 7, 9\} \]

Note that size of \( p \) that we are dealing with is \( |p| = |p_1| |\text{digits}|^{m-1} \) and that size of \( q \) that we are dealing with is \( |q| = |p_1| |\text{digits}|^{m-1} \).

Third, based on the Schoolboy’s multiplication method [4] \( n = p^* q = \Sigma x_{i+j-1} \), where \( i = 1 \) to \( m \), \( j = 1 \) to \( m \) and \( x_{i+j-1} = p_i^* q_j \) then,

\[ n_i + r_i = \begin{cases} \sum_{i=1}^{m} p_x \cdot q_{j} & \text{where } j = m \text{ to } 1, \quad i \leq m \\ \sum_{x=i-m+1}^{m} px \cdot q_{j} & \text{where } j = 2 \cdot m \text{ to } i - m + 1, \quad i > m \end{cases} \]

where \( m \) is the number of digits of \( p \) and \( q \) and \( r_i \) is the carry.

Running Time

The running time of such an algorithm depends on \( m \) the number of digits of \( n \). The running time of each of the three main procedures goes as follows:

- Running Time of Initial Producer

The first loop iterates maximum 3 times. One may ask why? Any \( n_1 \in \{1, 3, 7, 9\} \) since it is a product of \( p \) and \( q \) prime numbers. Table 9 lists all the possible values.
Table 9. All possible values of $p_1$ and $q_1$ and $n_1$ for RSA $n = p*q$

<table>
<thead>
<tr>
<th>$n_1$</th>
<th>$p_1$</th>
<th>$q_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>9</td>
</tr>
</tbody>
</table>

The second loop will also iterate 3 times in the worst case for the same reason as above.

The third loop will iterate 9 times in the worst case. Therefore, the running time of the first procedure is $3*3*9 = 81$ iterations, which is constant.

- Running Time of Producer
  - The first loop will iterate $m/2$, where $m$ is number of digits of $n$.
  - The second loop will run stat index times, where stat index = the number of elements in read file.
  - The third loop ($p_i$ loop) run 10 times.
  - The fourth loop runs ($q_i$ loop) 10 times.
  - The multiplication loops in worst case $m/2$ multiplications and $m/2$ addition (two nested loops $(m/2)^2$)
  - The fifth loop ($r_i$ loop) runs $((i*81 + r_{i-1})$ div 10).
  - Total running time = $m/2*stat index*10*10*r_i*(m/2)^2$.

- Running time of eliminator
  - The first loop runs $(m-(m/2+1))$.
  - The second loop runs number of tuples in the file.
  - The multiplication loops in worst case $m/2$ multiplications and $m/2$ addition (two nested loops $(m/2)^2$).
  - The third loop runs $((m-i)*81 + r_{i-1})$ div 10), where $i = (m-(m/2+1))$ to $m$.
  - Total running time = $(m-(m/2+1)) + stat index + (m/2)^2 + (((m-i)*81 + r_{i-1})$ div 10) and in worst case $i = m/2+1$.

Total running time for both procedures is:

- $m/2*stat index*10*10*r_i*(m/2)^2 + (m-(m/2+1)) + stat index + (m/2)^2 + (((m-i)*81 + r_{i-1})$ div 10)
- $m/8*100*r_i*2*stat index + m/2+1+m^2/4 + (((m-i)*81 + r_{i-1})$ div 10) and since we know that in worst case $i = m/2+1$. Then
- $m/8*100*r_i*2*stat index + m/2+1+m^2/4 + (((m-(m/2+1))*81 + r_{i-1})$ div 10)
- $m/8*100*r_i*2*stat index + m/2+1+m^2/4 + ((m/2-1)*81 + r_{i-1})$ div 10

Let’s remember here that $m$ is the number of digits of $n$ and we should remember that the RSA latest $n$ is about 500 digits long.

Let’s remember here that $m$ is the number of digits of $n$ and we should remember that the RSA latest $n$ is about 500 digits long.
Complexity and Storage Requirements

The storage required for the proposed algorithm can be measured by looking at the Producer algorithm. One must ask how many \( p_i, q_i, \) and \( r_i \) are there to store? Looking at table 5, there are 6 different combinations for the tuple \((p_1, q_1, r_1)\). Each one of them will produce at least 5 and at most 10 new distinct tuples \((p_2, q_2, r_2)\). For example when \( n_1 = 1 \) and \( n_2 = 1 \), the different combinations of the second tuples where 20 distinct tuples. If we take the worst case which is each tuple producing 10 different combinations, then one can conclude that the number of different combination \( s = 3 \times 10^{(m/2)-1} \). For example, in the previous example, one can count all different combinations in the worst case to be \( 3 \times 10 \), where \( m = 4 \) (\( m \) number of digits of \( n \)).

Hence, the question “Is this good?” will not be avoided. The answer is Yes, in comparison to sequential search, where the combination for each tuple \((p_1, q_1, r_1) = 10 \times 10 \times 9 = 900 \) in comparison to 3, where the proposed algorithm produced. The second tuple combination will be sequentially searched \( 900 \times 10 \times 10 \times 17 = 1530000 \). The proposed algorithm reduced the combinations to 30. Going back to the storage requirements, the number of combinations required is \( 3 \times 10^{(m/2)-1} \) and all are digits from 0 to 9 then the storage required needs maximum: number of combinations of \( m/2 \)-tuple* (4 bits for \( p_i \) + 4 bits for \( q_i \) + 10 bits for \( r_i \)). Where \( p_i \) and \( q_i \) are both digits range in value from 0 to 9, and the carry \( r_i \) ranges in value from 0 to 899 when \( m = 200 \). Therefore, the total storage needed \( 3 \times 10^{(m/2)-1} \times 18 \) bit. For example, if \( m = 200 \) then the storage needed is: Storage \( = 3 \times 10^{99} \times 18 \) bit \( = 54 \times 10^{99} / 8 \) byte, which is almost 6.139 \( 10^87 \) tira bytes (tera =1024^4).

Why it is Different?

The proposed algorithm is sequential in nature, since it sequentially looks for \( p_i \) and \( q_i \), and so it limits the different combinations by using the \( r_i-1 \). For example, in sequential factoring algorithm when factoring 6-digit \( n \) one will have to produce minimum \( 100 \times 100 \times 100 = 10^6 \) different combinations of \((p_m p_{m-1} \ldots p_1)\) and \((q_m q_{m-1} \ldots q_1)\). On the other hand when using the proposed algorithm the number of combinations is reduced to \( 3 \times 10^2 \times 10^{-3} \) combinations, which is a considerable improvement.

Advantages and Disadvantages

When factoring the RSA \( n \) it is always a trade off between time and storage. The proposed algorithm relays heavily on the storage due to the fact that storage is becoming bigger and cheaper. Another problem was finding the length of \( p \) and \( q \), which was solved [7]. Next, propose solutions to the two problems.

Solution of Storage

To treat the problem of the storage, we devised this algorithm to recursive. With the file swapping and no more than one two arrays to be dealt with and a vision of tree structure, the recursive version will work on depth-first search method elevating the complexity and storage. This algorithm is being tested on the computer and so far it is performing remarkably.
- Solution of the Length of p and q

It is a well-known fact that the number of digits of \( n \) depends on the number of digits of \( p \) and \( q \), where \(|n| \leq |p| + |q| +1\). A method of predicting the length of \( p \) and \( q \) can be solved by using \( p < \sqrt{n} \times 29/30 \). Now one can easily predict the number of digits of \( p \) and therefore, predict the number of digits of \( q \). Then devise \( m \) condition previously listed, and search sequentially for \( p \) and \( q \).

q Testing

The proposed algorithm was tested on PC P4 2 GHz with 256 Kb Ram, under Microsoft Windows 2000, using Visual Basic programming language. Table 10 shows some of the results.

### Table 10. Result, of the proposed algorithm

<table>
<thead>
<tr>
<th># of Digits</th>
<th>( n )</th>
<th>( p )</th>
<th>( q )</th>
<th>Time</th>
<th>Iterations</th>
<th># of ((p, q)) Experimented</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>106679</td>
<td>107</td>
<td>997</td>
<td>00:00:04</td>
<td>246</td>
<td>222</td>
</tr>
<tr>
<td>6</td>
<td>772561</td>
<td>941</td>
<td>821</td>
<td>00:00:01</td>
<td>33</td>
<td>29</td>
</tr>
<tr>
<td>8</td>
<td>17,762,981</td>
<td>9929</td>
<td>1789</td>
<td>00:03:26</td>
<td>5216</td>
<td>4740</td>
</tr>
<tr>
<td>8</td>
<td>59,545,127</td>
<td>6473</td>
<td>9199</td>
<td>00:02:34</td>
<td>3332</td>
<td>3027</td>
</tr>
<tr>
<td>8</td>
<td>42,551,263</td>
<td>8863</td>
<td>4801</td>
<td>00:00:02</td>
<td>96</td>
<td>85</td>
</tr>
<tr>
<td>8</td>
<td>32,011,687</td>
<td>4903</td>
<td>6529</td>
<td>00:03:03</td>
<td>2538</td>
<td>2305</td>
</tr>
<tr>
<td>10</td>
<td>1,034,535,127</td>
<td>86161</td>
<td>12007</td>
<td>00:14:04</td>
<td>14786</td>
<td>13439</td>
</tr>
</tbody>
</table>

q Comparison

Considering that many factoring algorithms were developed, we compared some of the outcomes of this paper with some of the well-known factoring algorithms. Bearing in mind, however, that most algorithms have been developed based on the value of \( n \), whereas, the proposed algorithm relays on the number of digits of \( n \) rather on \( n \) value. Table 11 shows the number of steps of each algorithm studied.

### Table 11. Steps of each algorithm studied

<table>
<thead>
<tr>
<th>Name of Algorithm</th>
<th>Number of Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>General field sieve</td>
<td>( \Theta \left( \exp \left( \left( \frac{64/9}{n} \right)^{1/3} \left( \log n \right)^{2/3} \right) \right) ) [8]</td>
</tr>
<tr>
<td>Trail division</td>
<td>( P ) where ( n = p \times q ), both primes [9]</td>
</tr>
<tr>
<td>Richard Schroeppel</td>
<td>( \exp \left( \sqrt{\ln(n) \cdot \ln(\ln(n))} \right) ) [10]</td>
</tr>
<tr>
<td>The proposed algorithm</td>
<td>( m^3/8 \times 100 \times r,2^2 \times \text{stat_index}+m/2-1+m^2/4+((m/2-1)^81+r,i,1) \div 10 )</td>
</tr>
</tbody>
</table>

So, rather than a running time that will depend on the value of \( n \), which can be a very huge number i.e. \( n \) of 200 digits, the proposed algorithm relies on the number of digits of \( n \), which is in our example 200. Note that \( m \) in the proposed algorithm is only the number of digits of RSA \( n \). As mentioned before, the latest RSA \( n \) is about 500 digits, and one cannot compare 500 to value of \( n \) with 500 digits. As for Trail Division method, it can only work for certain length of \( p \) and \( q \) then fails when the length exceeds certain number (on the PC 15 digits). Richard Schroeppel’s method is not published but mentioned in [10], we only have the running time of the algorithm and it is obvious from the running time that it depends on the value of \( n \).
Conclusion

The idea of the algorithm is not far fetched in fact Strassen showed that FTT techniques can be utilized to factor \( n \) in \( O(n^{1/3+\varepsilon}) \) steps [11]. This paper is an ongoing research, which is believed by the researchers to lead to a better performance of the algorithm. A recursive program was developed and is working which solved the problem of the storage and the complexity.

References

7. Aboud S., (2003), Lecture Notes, Arab Academy, Jordan.
RSA
خوارزمية حتمية جديدة لتحليل معامل الـ RSA

ستار جبار عبود1 وأييون فرحان أبوتوية2

1 كلية الدراسات الحاسوبية العليا، جامعة عمان العربية للدراسات العليا، عمان، الأردن
2 قسم نظام المعلومات الحاسوبية، الأكاديمية العربية، عمان، الأردن

الملخص: العديد من أساليب حماية الشبكة تعتمد على مشكلة عددية تخفيض هوية التحليل على عدد الأعداد الصحيحة، على كل حال في السنوات الأخيرة هناك تقدم كبير في حفرة تحليل العوامل بالاعتماد على طرق غير حتمية، هذه الورقة تتضمن خوارزمية جديدة تحليل معامل الـ RSA (n = p * q) تقوم بتحليل العوامل تقوم بتحليل المعامل زمن تنفيذ الخوارزمية المبسطة يعتمد على عدد أرقام العوامل (n) وليس على قيمة معينة. إن طبيعة مشكلة التحليل تعتمد على حساب الزمن، ودرجة التعمية والسلامة الخفية للخوارزمية، بنفس الخوارزمية المبسطة تقوم على تحليل هذه المشكلة من خلال استخدام فكرة الضرب الطويل لتحديد القيمة الممكّنة لكل من (p) (p) و (q).

وقد تتبّع هذه المشكلة السهولة المحاسبة الخفيفة تم استخدام طريقة البث الحسابي أو مع تطبيق أساليب المعاداة، إضافة إلى ذلك فإن الورقة تضمنت دراسة تحليل الخوارزمية المبسطة زمن تنفيذها ودرجة تعقيدها، وأخيراً فإن ما اكتسبته الورقة هو إمكانية القيام بأعمال تحسينات على الخوارزمية كعمل مستقبلي.