AUSTRALASIAN JOURNAL OF COMBINATORICS
Volume **33** (2005), Pages 291–305

RAK factoring algorithm *

Y. Bani Hammad[1]   G. Carter[2]   E. Dawson[1]

[1]*Information Security Research Centre*   [2]*School of Mathematics and Science*
Queensland University of Technology
G.P.O. Box 2434, Brisbane 4001
Australia

W. Müller

Institute of Mathematics
Klagenfurt University
Austria

Y. Hitchcock

Information Security Research Centre
Queensland University of Technology
G.P.O. Box 2434, Brisbane 4001
Australia

Abstract

Two hard mathematical problems provide the basis for the security of public key cryptosystems used today. The first is the factoring problem, that is, determining a prime factor of a given integer when the only factors of the given integer are large. The second is the discrete logarithm problem, that is, determining the index t given g and h in a group where $g^t = h$. The proposed algorithm, RAK, essentially converts the factoring problem to a discrete log problem, so it can be used for factoring as well for determining discrete logarithms. It is particularly useful for numbers that are the product of two large primes p and q of approximately the same order, such as an RSA modulus. RAK offers significant improvement over more traditional methods such as Fermat factorisation. After presenting RAK we compare its complexity with Fermat factorisation.

1 Introduction

The difficulty of the factoring problem, that is, given a large integer n with no small factors determine a prime factor of n, provides the security of many cryptosystems,

* RAK is the name of a city in the United Arab Emirates.

for example RSA [2] [10]. The first dedicated factoring technique was due to Fermat [4] and, because of its simplicity, is generally the first technique tried when a large number is to be factored after implementing trial divisions of the number n by primes less than some given bound. Hence, Fermat factoring algorithm is often used as a benchmark for the efficiency of factoring algorithms. Fermat factorization as it is known will be described in section 5. In this paper we present a new factoring algorithm, the RAK algorithm, which is highly efficient for numbers of the form pq where p and q are distinct primes. The algorithm essentially converts the factoring problem into a discrete logarithm problem. It is as efficient as Fermat factorisation where p and q are twin primes and becomes far superior to Fermat as the difference between primes p and q increases. It should be noted however that RAK is at its most efficient when p and q are "close" as is the case in Fermat factorisation, thus making a comparison with Fermat meaningful.

2 The general factorization technique

Factoring techniques fall into two categories, special and general. The special techniques are applied to a factor number n that has some particular properties or its factors have some particular properties. For example, Fermat factorisation, works particularly well when n has only two prime factors whose difference is "small". Similarly, there are techniques which can be applied to numbers which contain the factor p and $p - 1$ is the product of numerous "small" factors. On the other hand, general techniques can applied to factor numbers of any form. The most powerful general factoring algorithm is the general number field sieve [1]. With these definitions, the RAK technique would be described as special. The generic factoring technique can be described as follows. Given an integer n to be factored, first apply a compositeness test. If this test fails then n is prime and algorithm terminates. If n passes compositeness test, a special purpose factoring algorithm is applied. This algorithm is allowed to run either for a certain predetermined time or for a certain predetermined number of iterations. If n is factored the algorithm terminates. If n is not factored, then a general purpose algorithm is applied, again with a set time or number of iterations to be performed. The diagram below illustrates the process of factorisation.

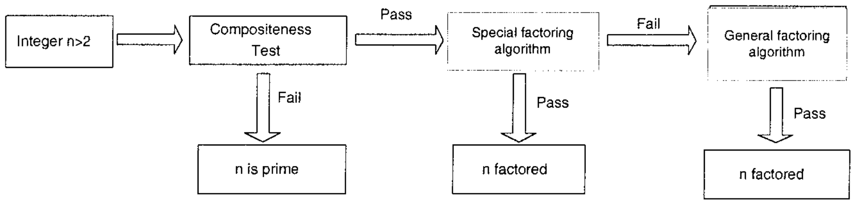

Figure 1: General factorization technique

3 The mathematics behind RAK Algorithm

The RAK algorithm relies on the following theorems:

Theorem 1 [8] *If p and q are distinct integer primes and $n = pq$, then $n = \Phi(n) + p + q - 1$, where $\Phi(n) = (p-1)(q-1)$, and $\Phi(n)$ is the Euler totient function.*

Theorem 2 Euler Totient theorem: [5] *If* $\gcd(a, n) = 1$ *and* $n = \prod p_i^{\alpha_i}$, *then* $a^{\Phi(n)} \equiv 1(\bmod n)$.

Theorem 3 Arithmetic-Geometric Mean: [8] *Given distinct numbers x and y then $\frac{x+y}{2} \geq \sqrt{xy}$.*

Theorem 4 [9] *If $a^2 \equiv b^2 \bmod n$ and $a \not\equiv \pm b \bmod n$ then $1 < \gcd(a \pm b, n) < n$.*

The following results can be deduced from these theorems.

Let $n = pq$ be a number to be factored, where p and q are distinct odd prime numbers. Now if $\gcd(a, n) = 1$ for any integer a, we have from theorems 1 and 2.

Corollary 1

$$a^{n+1} \equiv a^{p+q} \bmod n \tag{1}$$

Theorem 5 *$p + q \geq 2(\lfloor\sqrt{n}\rfloor + 1)$ where $\lfloor\sqrt{n}\rfloor$ is the integer part of $\sqrt{n}$.*

Theorem 6 *If n is of the form pq and $4|(n+1)$ then $4|(p+q)$.*

Theorem 7 *If n is of the form pq and $8|(n+1)$ then $8|(p+q)$.*

Theorem 8 *If p and q are numbers with $q > p$ such that $q = p + d$ where $d > 0$ then $D = p + q - 2(\lfloor\sqrt{pq}\rfloor + 1)$ is an increasing function of d for fixed p.*

Theorem 9 *If $n = pq$ and $q = p + 2$ then $\lfloor\sqrt{n}\rfloor = p$.*

Theorem 10 *If $n = pq$ and $q = p + 4$ then $\lfloor\sqrt{n}\rfloor = p + 1$.*

4 RAK Algorithm

In this section a description of the algorithm is given.

4.1 Verbal Description of RAK

RAK has a precomputation phase and an iterative phase. In the precomputation phase an array A is constructed of size e^2. The value of e will be discussed in the section 4.2. The array consists of elements of the form $(a^2)^z \bmod n$, $1 \leq z \leq e^2$. The choice for a will be discussed in section 4.3.
Definition: Let $c \equiv a^{n+1-2(\lfloor\sqrt{n}\rfloor+1)} \bmod n$.

Since $n+1$ is even

$$c \equiv (a^2)^{\frac{n+1}{2}-(\lfloor\sqrt{n}\rfloor+1)} \bmod n \tag{2}$$

The array is then searched for the value c calculated from equation (2). If c is found then there exists a z, $(1 \leq z \leq e^2)$ such that

$$c \equiv (a^2)^z \bmod n.$$

$$(a^2)^{\frac{n+1}{2}-(\lfloor\sqrt{n}\rfloor+1)} \equiv (a^2)^z \bmod n.$$

$$(a^2)^{\frac{n+1}{2}-(\lfloor\sqrt{n}\rfloor+1)-z} \equiv 1 \bmod n.$$

It follows that

$$1 < \gcd(a^{\frac{n+1}{2}-(\lfloor\sqrt{n}\rfloor+1)-z} \pm 1 \bmod n, n) < n$$

provided that $a^{\frac{n+1}{2}-(\lfloor\sqrt{n}\rfloor+1)-z} \not\equiv \pm 1 \bmod n$, and the algorithm terminates.

If c is not found in array A, then RAK proceeds to the iterative phase where

$$c \equiv c.((a^2)^{e^2})^{-1} \bmod n$$

and array A is searched for this new value of c. If c is found then there exists a z, $(1 \leq z \leq e^2)$ such that

$$(a^2)^z \equiv c \bmod n$$

that is

$$(a^2)^{\frac{n+1}{2}-(\lfloor\sqrt{n}\rfloor+1)-z-e^2} \equiv 1 \bmod n$$

and

$$1 < \gcd(a^{\frac{n+1}{2}-(\lfloor\sqrt{n}\rfloor+1)-z-e^2} \pm 1, n) < n$$

provided $a^{\frac{n+1}{2}-(\lfloor\sqrt{n}\rfloor+1)-z-e^2} \not\equiv \pm 1 \bmod n$, and the algorithm terminates. If c is not found then $c \equiv c.a^{-2e^2} \bmod n$ and the process repeats.

Clearly, at each step of the iterative phase the index of c is decreasing by e^2 from a maximum of $\frac{p+q}{2} - (\lfloor\sqrt{n}\rfloor + 1)$. Hence, the index of c must eventually decrease to a point where $1 <$ index of $c < e^2$, and so c must appear in array A. If c is found after m iterations then

$$(a^2)^{\frac{n+1}{2}-(\lfloor\sqrt{n}\rfloor+1)-z-me^2} \equiv 1 \bmod n$$

and

$$1 < \gcd(a^{\frac{n+1}{2}-(\lfloor\sqrt{n}\rfloor+1)-z-me^2} \pm 1, n < n)$$

provided

$$a^{\frac{n+1}{2}-(\lfloor\sqrt{n}\rfloor+1)-z-me^2} \not\equiv \pm 1 \bmod n$$

If after finding c,

$$a^{\frac{n+1}{2}-(\lfloor\sqrt{n}\rfloor+1)-z-me^2} \equiv \pm 1 \bmod n$$

then the algorithm determines

$$\gcd(b^{\frac{n+1}{2}-(\lfloor\sqrt{n}\rfloor+1)-z-me^2} \pm 1 \bmod n, n)$$

where b is the next prime $> a$. As with the case of the base "a",

$$1 < \gcd(b^{\frac{n+1}{2}-(\lfloor\sqrt{n}\rfloor+1)-z-me^2} \bmod n \pm 1, n) < n$$

provided

$$b^{\frac{n+1}{2}-(\lfloor\sqrt{n}\rfloor+1)-z-me^2} \not\equiv \pm 1 \bmod n$$

If

$$b^{\frac{n+1}{2}-(\lfloor\sqrt{n}\rfloor+1)-z-me^2} \equiv \pm 1 \bmod n$$

then repeat the above process for the next prime larger than b. This process can be controlled by limiting the range of b or the number of primes b to be tested.

In section 4.4 we let $a = 2$ and $b = 3$ or 5. In section 4.6 we show how to extend the values of b beyond 5 if the algorithm fails to produce a result for 2,3 and 5. Section 4.7 also contains some statistics on the success of the algorithm using 2,3 and 5 only, and for those examples that did not resolve using 2,3 and 5, how large b had to be taken for these to be resolved.

4.2 The size of Array A

A technique that could be used is just to generate successive powers of a^2 until c is found. However, this would require a computationally expensive modular exponentiation each time and would be no more efficient than Fermat factorisation which increments, in its iterative step, by 1 each time. This technique would require a maximum number of $\frac{p+q}{2} - (\lfloor\sqrt{n}\rfloor + 1)$ modular exponentiations which is unknown since $p + q$ is unknown. However, there would be no storage requirements. What RAK does is create array A of fewer than $\frac{p+q}{2} - (\lfloor\sqrt{n}\rfloor + 1)$ elements when $\frac{p+q}{2} - (\lfloor\sqrt{n}\rfloor + 1)$ is large, and compares these elements to c, which is relatively inexpensive, computationally speaking. The size of the array is therefore a compromise between storage of array elements and the speed and efficiency of the search of the array. Ideally, array A should not be too "large" and should only depend on the size of n and not on the sizes of the factors p and q of n. As theorem 8 indicates, for given n of a particular size, the value of $\frac{p+q}{2} - (\lfloor\sqrt{n}\rfloor + 1)$, where p and q are the prime factors of n, is very dependent on the relative sizes of p and q.

Experimentation has indicated that an array size of e^2 where $e = \lfloor\frac{\ln n}{\ln 2}\rfloor$ is efficient for our purpose. Now for pq of RSA size, say 1024 bits, $\lfloor\frac{\ln pq}{\ln 2}\rfloor \approx 1024$ so array A will be approximately $(1024)^2 \approx 10^6$. If $\lfloor\frac{\ln n}{\ln 2}\rfloor^2 \geq \frac{p+q}{2} - \lfloor\sqrt{n}\rfloor + 1)$ then the initial value of c will occur in the array and the algorithm will terminate. If, on the other hand, $\lfloor\frac{\ln n}{\ln 2}\rfloor^2 < \frac{p+q}{2} - (\lfloor\sqrt{n}\rfloor + 1)$ the algorithm may proceed to the iterative phase. The search for an "optimal" array size is the subject of ongoing research.

4.3 The choice for a

The choice of a as the base in the calculation of the elements of array A is a tradeoff between the size of a and its order modulo n, the number to be factored. The original approach was to try a "small" value for a as this facilitated the calculation of the elements of array A. However, we speculate that a "small" value for "a" will have a large order mod n. This is based on a table in [6] where the author has stated that 2,3, and 5 are generators of the group Z_p, p prime, $p < 4000$, with high probability. If a has a large order then the number of iterative steps in the algorithm is likely to increase. We have selected $a = 2$ as the base element in the algorithm described in section 4.4. The choice of a "larger" base element with smaller order may reduce the number of iterative steps in the algorithm but it will increase the complexity of the calculation of array elements. These aspects, related to the choice of base element for the calculation of the array elements, are the subject of ongoing research.

4.4 Pseudo Code for RAK

```
Precomputation phase
$c \equiv 4^{\frac{n+1}{2}-(\lfloor\sqrt{n}\rfloor+1)} \bmod n$
If $c = 1$ then $x = \frac{n+1}{2} - (\lfloor\sqrt{n}\rfloor + 1)$ Go To Test
$e = \lfloor\frac{\ln n}{\ln 2}\rfloor$
$A = [4^z \bmod n]$, $1 \le z \le e^2$
Search $A$ for $c$. If $c \in A$ then $x = \frac{n+1}{2} - (\lfloor\sqrt{n}\rfloor + 1) - z$ Go To Test
$h \equiv (4^{e^2})^{-1} \bmod n$
$m = 1$ ($m$ keeps track of the number of iterations)
and go to iterative phase.
Iterative Phase
      Repeat
        $c \equiv c.h \bmod n$
        If $c \notin A$ then $m = m + 1$.
        If $c \in A$ then $x = \frac{n+1}{2} - (\lfloor\sqrt{n}\rfloor + 1) - me^2 - z$ Go To Test
Test
        while(($x$ is even)&($4^x \equiv 1 \bmod n$))
           $x = \frac{x}{2}$
         If $4^x \equiv -1 \bmod n$ then $x = 2x$
        If $\gcd(2^x \pm 1, n) \not\equiv \pm 1 \bmod n$ and $\gcd(2^x \pm 1, n) = p$ and $q$ Stop.
        If $3^x \not\equiv \pm 1 \bmod n$ and $\gcd(3^x \pm 1, n) \neq 1$ or $n$ Stop.
        If $5^x \not\equiv \pm 1 \bmod n$ and $\gcd(5^x \pm 1, n) \neq 1$ or $n$ Stop.
        Go To Second_Test (see section 4.6)
```

4.5 Numerical example

Let $n = 3711$ be a composite number to be factored.
$c \equiv 4^{1795} \bmod 3711 \equiv 2488$
$e = 11$
Calculate array A

4	16	64	256	1024	385	1540	2449	2374	2074	874
3496	2851	271	1084	625	2500	2578	2890	427	1708	3121
1351	1693	3061	1111	733	2932	595	2380	2098	970	169
676	2704	3394	2443	2350	1978	490	1960	418	1672	2977
775	3100	1267	1357	1717	3157	1495	2269	1654	2905	487
1948	370	1480	2209	1414	1945	358	1432	2017	646	2584
2914	523	2092	946	73	292	1168	961	133	532	2128
1090	649	2596	2962	715	2860	307	1228	1201	1093	661
2644	3154	1483	2221	1462	2137	1126	793	3172	1555	2509
2614	3034	1003	301	1204	1105	709	2836	211	844	3376
2371	2062	826	3304	2083	910	3640	3427	2575	2878	379

Since $2488 \notin A$ move to iterative phase
$h = 235$
$m = 1$
$c = 2053 \notin A$ and $m = 2$
$c = 25 \notin A$ and $m = 3$
$c = 2164 \notin A$ and $m = 4$
$c = 133 \in A$ and index(133)=75
$4^{1856-61-4\cdot121-75} \bmod 3711 \equiv 4^{1236} \bmod 3711 \equiv 1$
Since $2^{1236} \bmod 3711 \equiv 1$
$3^{1236} \bmod 3711 \equiv 2475$
$\therefore \gcd(2474, 3711) = 1237$ and $3711 = 1237 \cdot 3$
Note: The algorithm could terminate in precomputeation phase if we noticed that $4^{16} \bmod 3711 \equiv 625 \equiv 25^2$ and then $2449^2 \bmod 3711 \equiv 25^2$ so $(2449+25)(2449-25) \bmod 3711 \equiv 0$ and $3711 = 3 \cdot 1237$

4.6 Increasing the Likelihood of Success of RAK

If the algorithm in section 4.4 fails to resolve n into its factors we execute a second algorithm Second_Test which essentially enables us to increase the base to numbers larger than 5. In other words if 2,3 and 5 do not succeed, Second_Test enables us to use 7,11,13, ... up to some given bound. The pseudo code for Second_Test follows.

Second_Test

```
    $a = 5$
    Repeat
        $b$ is the next prime $> a$
        $c \equiv b^x \bmod n$
        if $c \not\equiv \pm 1 \bmod n$ then $\gcd(c \pm 1, n) = p$ or $q$ and stop
        else $a = b$
    Until $b >$ some given bound.
```

4.7 Failure of the RAK Algorithm

RAK will not always be able to resolve a number n into its factors. The RAK tests will not succeed when either

1. $b^x \equiv \pm 1 \bmod n$ for all choices of b.

2. $b^x \not\equiv \pm 1 \bmod n$ and $\gcd(b^x \pm 1, n) = 1$ or n for all choices of b.

For $b \leq 5$ we tested 1000 pairs of 20 digits primes and RAK was successful 94.7% of the time. By increasing b to a maximum of 29, the RAK algorithm was 100% successful. This is the subject of further research.

5 Fermat's Factoring method [4]:

Proposed in the 17th century by French mathematician Pierre de Fermat, Fermat factoring is the oldest systematic method of factoring a composite number into its prime factors. The algorithm looks for integers x and y such that $n = x^2 - y^2$ then $n = (x-y)(x+y)$. The algorithm works well when the difference between two prime factors p and q is small.

There is a generalization of the idea behind the Fermat factoring method, which leads to a much more efficient factoring method [9]. If we are able to find x and y such that $x^2 \equiv y^2 \bmod n$, where $x \not\equiv \pm y \bmod n$, then $\gcd(x \pm y, n)$ are two factors of n, and many factoring methods such as factor base [4], continued fraction [4], Quadratic Sieve (QS) [4], Multi Polynomial Quadratic Sieve (MPQS) [12],and General Number Field Sieve (GNFS) [1] use this technique to factor the number n.

5.1 Fermat Factoring Algorithm:

```
$r = \lfloor\sqrt{n}\rfloor$
$i = 1$
        Repeat
            $a = (r+i)^2 - n$
            If $a$ is not perfect square then $i = i + 1$
    $n = (r + i - \sqrt{a})(r + i + \sqrt{a})$
```

6 Complexity Comparisons

For n of the form pq where $p < q$, suppose that Fermat factorisation takes t iterations and RAK takes m iterations.

6.1 Fermat Complexity

In this case from Section 5.1

$$\lfloor\sqrt{n}\rfloor + t - \sqrt{a} = p$$

and

$$\lfloor\sqrt{n}\rfloor + t + \sqrt{a} = q$$

so

$$t = \frac{p+q}{2} - \lfloor\sqrt{n}\rfloor.$$

It follows that

$$p + q = 2(\lfloor\sqrt{n}\rfloor + 1) + 2(t - 1). \tag{3}$$

6.2 RAK Complexity

From Section 4.1 if c is found after m iterations then

$$2(\lfloor\sqrt{n}\rfloor + 1) + 2me^2 + 2z \leq p + q. \tag{4}$$

From (3) and (4) it follows that

$$t - 1 \geq me^2 + z$$

and

$$t \geq me^2 + z + 1. \tag{5}$$

Thus, in Fermat factorisation there are at least $me^2 + z + 1$ squarings and $me^2 + z + 1$ square root extractions whereas in RAK there are m or fewer modular multiplications and $me^2 + z$ comparisons.

6.3 RAK vs. Fermat

Table 1 compares Fermat algorithm with RAK.

Table 2 indicates running times in seconds in a number of trials for the RAK algorithm and the Fermat algorithm, in factoring numbers of the form pq where p and q are 20 digit primes.

Criteria	Fermat	RAK
Precomputation	$r = \lfloor\sqrt{n}\rfloor$	$c = 4^{\frac{n+1}{2} - (\lfloor\sqrt{n}\rfloor + 1)} \bmod n$ $e = \lfloor\frac{\ln n}{\ln 2}\rfloor$ Calculate the array A $m = 1$ $h \equiv 4^{-e^2} \bmod n$
Number of iterations	$\frac{p+q}{2} - \lfloor\sqrt{n}\rfloor$	$\leq \lfloor\frac{\frac{p+q}{2} - \lfloor\sqrt{n}\rfloor}{e^2}\rfloor$
Operations in each step	$r = r + 1$ $b = r^2$ $c = b - n$ Checking whether c is a perfect square or not	$c \equiv c.h \bmod n$ $m = m + 1$ Search for c in the Array A
Pre-iterative phase termination conditions	No	Yes

Table 1: Comparison of RAK and Fermat

In column 2 the first 15 rows of RAK times represent the average time to run the RAK algorithm with 1000, p, q pairs whose minimum difference is indicated in column 1. The next 2 RAK times are average times for 100 p, q pairs, while the remaining rows of RAK times are for a single p, q pair.

In column 3 the first 8 rows of Fermat times are average times to run 1000 p, q pairs. The next row is the time to run 200 p, q pairs and the remaining rows are the times to run one pair.

Note that in successive rows of the table the difference $q - p$ is increasing.

Note that when the minimum $q - p = 100000000002$ RAK becomes significantly faster than Fermat and that at minimum $q - p = 8800000000026$ RAK is 2056 times faster than Fermat. Looking at the last entries in columns 2 and 3 one sees that even when the difference between p and q is ≈ 1000 times greater RAK is still ≈ 2.5 times as fast as Fermat.

The running time of RAK is based on the algorithm which is described in section 4.4 and that running time can be improved if we apply the techniques for algorithm efficiencies which are described in section 8.

minimum $q - p$	RAK runing time	Fermat Runing time
100002	0.00469410	0.00297910
1000002	0.00468410	0.00308210
10000002	0.00467610	0.00318110
100000002	0.00463910	0.00325610
1000000002	0.00466110	0.00334610
10000000002	0.00466310	0.00380910
100000000002	0.00468210	0.00942310
500000000002	0.00607710	0.16537610
1000000000002	0.01019710	0.71425410
2200000000002	0.03861810	8.35613810
3300000000102	0.06413210	20.50635910
5500000000002	0.07504310	52.69080310
6600000000042	0.07442710	76.96297010
8800000000026	0.07524310	154.70656910
10000000000010	0.07507710	
100000000000002	0.08279710	
1000000000000002	0.84976710	
2000000000000020	6.47825910	
3000000000000014	14.29810610	
4000000000000004	17.23231110	
5000000000000010	39.42578310	
6000000000000206	47.95687910	
7000000000000042	60.98136210	

Table 2: Running Times for RAK and Fermat

7 Some Special Cases

In this section we describe three special cases.

7.1 Case $q = p + 2$ (twin primes) and $q = p + 4$

From equation (5) we can examine the case of $t = 1$, which corresponds to the two cases in question. From (5) $t = 1$ implies $m = 0$ and $z = 0$. This in turn implies that there is no need to construct the array in RAK. This is indeed so, since from theorem 9 we have

$$2p + 2 = 2(\lfloor\sqrt{n}\rfloor + 1)$$

and in the case $p + 4 = q$ we have from theorem 9

$$2p + 4 = 2(\lfloor\sqrt{n}\rfloor + 1)$$

In addition $p + q = 2p + 2$ in the twin prime case and $p + q = 2p + 4$ in the case $p + 4 = q$. Thus, $2(\lfloor\sqrt{n}\rfloor + 1) = p + q$ and from section 4.1

$$c \equiv 4^{\frac{-(p+q)}{2}}.4^{\frac{n+1}{2}} \bmod n$$

and from equation (1)

$$c \equiv 2^{-(p+q)}.2^{p+q} \bmod n \equiv 1$$

Thus, in the case where $t = 1$ in equation (5), $c = 1$ in RAK.

7.2 $\frac{\ln c}{\ln a} = I$ an integer

If $\frac{\ln c}{\ln a}$ is an even integer I then $c = a^I$ and from equations (1) and (2) we have

$$a^{n+1-2(\lfloor\sqrt{n}\rfloor+1)} \equiv a^I \bmod n$$

$$(a^2)^{\frac{n+1}{2}-(\lfloor\sqrt{n}\rfloor+1)-\frac{I}{2}} \equiv 1 \bmod n$$

$$1 < \gcd(a^{\frac{n+1}{2}-(\lfloor\sqrt{n}\rfloor+1)-\frac{I}{2}} \pm 1, n) < n$$

provided $a^{\frac{n+1}{2}-(\lfloor\sqrt{n}\rfloor+1)-\frac{I}{2}} \not\equiv \pm 1 \bmod n$.
Now, if I is an odd integer or in the case I is even, and $a^{\frac{n+1}{2}-(\lfloor\sqrt{n}\rfloor+1)-\frac{I}{2}} \bmod n \equiv \pm 1 \bmod n$, with high probability $\gcd((a \pm 1)^{n+1-2(\lfloor\sqrt{n}\rfloor+1)-I} \pm 1 \bmod n, n) = p$ or q.

7.3 $p > \alpha$ and $q > \beta$ for some known α and β, both odd or both even

If $p > \alpha$ and $q > \beta$ for some known α and β, both odd or both even as well as $2(\lfloor\sqrt{n}\rfloor + 1) < \alpha + \beta$, we can use $\alpha + \beta$ instead of $2(\lfloor\sqrt{n}\rfloor + 1)$ in equation (2). Thus, $c \equiv (a^2)^{\frac{n+1}{2}-(\frac{\alpha+\beta}{2})} \bmod n$. This reduces the number of iterations required as $\frac{n+1}{2} - \frac{\alpha+\beta}{2}$ is $< \frac{n+1}{2} - (\lfloor\sqrt{n}\rfloor + 1)$ making it closer to $2e^2$, the maximum index in the array.

8 Other Algorithm Efficiencies

1. In generating the array A one can check whether or not there exists a $z > \frac{e}{2}$ such that $(a^2)^z \bmod n \equiv m^2$, $0 < m \leq \lfloor\sqrt{n}\rfloor$. If such a z exists then by theorem 4, $1 < \gcd(a^z \pm m, n) < n$ provided $a^z \not\equiv -m \bmod n$.

2. In generating array A one can check whether or not there exists a z and an even s such that $(a^2)^z \equiv b^s \bmod n$, where $z > \frac{e}{2}$ and $1 < b^s < n$. If such a z and s exist then $(a^2)^z - b^s \equiv 0 \bmod n$ and it that $(a^z - b^{\frac{s}{2}})(a^z + b^{\frac{s}{2}}) \equiv 0 \bmod n$ and if $a^z \not\equiv \pm b^{\frac{s}{2}} \bmod n$, then $1 < \gcd(a^z \pm b^{\frac{s}{2}}, n) < n$.

   Both of the above checks can be performed at the precomputation phase and hence may preclude the need for the iterative phase.

3. Sorting the array A in ascending order makes it quicker to search. In addition, calculating the inverse of each element in the sorted array as in the following table,

Index	Element	Inverse
b_1	a_1	c_1
b_2	a_2	c_2
...	...	...
b_{e^2}	a_{e^2}	c_{e^2}

   where $a_1 < a_2 < \ldots < a_{e^2}$, and checking if there exist an i and a j such that $a_i = c_j$ for some i and j where $1 \leq i, j \leq e^2$, then we have

$$(a^2)^{-i} \equiv 4^j \bmod n$$

$$a^{2(i+j)} \equiv 1 \bmod n$$

   so by theorem 4

$$\therefore 1 < \gcd(n, a^{i+j} \pm 1) < n$$

   provided $a^{i+j} \not\equiv \pm 1 \bmod n$

4. Let $\alpha = \lfloor \frac{n}{\#n+1} \rfloor$ and $m = 0$ where $\#n$ is number of digits of n. α is another value that is not "too large" for an array size. Create a second array B

$\beta_1 \equiv (a^2)^{\alpha} \bmod n$	β_1^{-1}
$\beta_2 \equiv (a^2)^{2\alpha} \bmod n$	β_2^{-1}
$\beta_3 \equiv (a^2)^{3\alpha} \bmod n$	β_3^{-1}
...	...
$\beta_{\#n} \equiv (a^2)^{\#n\alpha} \bmod n$	$\beta_{\#n}^{-1}$

   If β_i or $\beta_i^{-1} \in A$, where $1 \leq i \leq \#n$ then there exists a k, $1 \leq k \leq \#n$, such that

$$(a^2)^{k\alpha} \equiv (a^2)^z \bmod n$$

$$(a^{k\alpha - z})^2 \equiv 1 \bmod n$$

$$\therefore 1 < \gcd(a^{k\alpha - z} \pm 1, n) < n$$

provided $a^{k\alpha-z} \not\equiv \pm 1 \bmod n$ If β_i and $\beta_i^{-1} \notin A$ set $m = m+1$ in the iterative part of algorithm 4.4 and

$\beta_1 \equiv \beta_1.(a^2)^{e^2} \bmod n$	$\beta_1^{-1} \equiv \beta_1^{-1}.(a^2)^{e^2} \bmod n$
$\beta_2 \equiv \beta_2.(a^2)^{e^2} \bmod n$	$\beta_2^{-1} \equiv \beta_2^{-1}.(a^2)^{e^2} \bmod n$
$\beta_3 \equiv \beta_3.(a^2)^{e^2} \bmod n$	$\beta_2^{-1} \equiv \beta_3^{-1}.(a^2)^{e^2} \bmod n$
...	...
$\beta_{\#n} \equiv \beta_{\#n}.(a^2)^{e^2} \bmod n$	$\beta_{\#n}^{-1} \equiv \beta_{\#n}^{-1}.(a^2)^{e^2} \bmod n$

Thus, we are no longer just searching A for c but for β_i, $(1 \leq i \leq \#n)$ as well, and so the chances of finding a match are increased.

5. Choose larger e. Because each iterative step increments the index of c by $2e^2$, the larger e is, the fewer iterative steps that are needed to find c in the array A. The upper bound for e is $\frac{p+q}{2} - \lfloor\sqrt{n}\rfloor - 1$.

6. From theorem 6 and equation (1) if $4|(n+1)$ then build array $A = [(a^4)^z \bmod n]$, $1 < z < e^2$. Determine $c \equiv (a^4)^{\frac{n+1}{4} - \lfloor\frac{(\lfloor\sqrt{n}\rfloor+1)}{2}\rfloor} \bmod n$ and $h \equiv (a^4)^{-e^2} \bmod n$ and follow algorithm 4.4. If c is found after m iterations then $c \equiv (a^4)^z \bmod n$ for some z and so we have

$$(a^4)^{\frac{n+1}{4} - \lfloor\sqrt{n}\rfloor+1) - me^2 - z} \equiv 1 \bmod n$$

and $1 < \gcd((a^2)^{\frac{n+1}{4} - (\lfloor\sqrt{n}\rfloor+1) - me^2 - z} \pm 1 \bmod n, n) < n$ provided $(a^2)^{\frac{n+1}{4} - (\lfloor\sqrt{n}\rfloor+1) - me^2 - z} \not\equiv \pm 1 \bmod n$. If $(a^2)^{\frac{n+1}{4} - (\lfloor\sqrt{n}\rfloor+1) - me^2 - z} \equiv \pm 1 \bmod n$ then
$1 < \gcd(a^{\frac{n+1}{4} - (\lfloor\sqrt{n}\rfloor+1) - me^2 - z} \pm 1 \bmod n, n) < n$ provided
$a^{\frac{n+1}{4} - (\lfloor\sqrt{n}\rfloor+1) - me^2 - z} \not\equiv \pm 1 \bmod n$.

If $a^{\frac{n+1}{4} - (\lfloor\sqrt{n}\rfloor+1) - me^2 - z} \equiv \pm 1 \bmod n$ then proceed as in section 4.6.

In this scenario the number of iterations required decreases. Applied to the example in section 4.5 the number of iterations is halved to 2.

7. From theorem 7 and equation (1) if $8|(n+1)$ then build array $A = [(a^8)^z \bmod n]$, $1 < z < e^2$. Determine $c \equiv (a^8)^{\frac{n+1}{8} - \lfloor\frac{(\lfloor\sqrt{n}\rfloor+1)}{4}\rfloor} \bmod n$ and $h \equiv (a^8)^{-e^2} \bmod n$ and follow algorithm 4.4. If c is found after m iterations then $c \equiv (a^8)^z \bmod n$ for some z and so we have

$$(a^8)^{\frac{n+1}{8} - \lfloor\sqrt{n}\rfloor+1) - me^2 - z} \equiv 1 \bmod n$$

and $1 < \gcd((a^4)^{\frac{n+1}{8} - (\lfloor\sqrt{n}\rfloor+1) - me^2 - z} \pm 1 \bmod n, n) < n$ provided $(a^4)^{\frac{n+1}{8} - (\lfloor\sqrt{n}\rfloor+1) - me^2 - z} \not\equiv \pm 1 \bmod n$. If $(a^4)^{\frac{n+1}{8} - (\lfloor\sqrt{n}\rfloor+1) - me^2 - z} \equiv \pm 1 \bmod n$ then
$1 < \gcd((a^2)^{\frac{n+1}{8} - (\lfloor\sqrt{n}\rfloor+1) - me^2 - z} \pm 1 \bmod n, n) < n$ provided
$(a^2)^{\frac{n+1}{8} - (\lfloor\sqrt{n}\rfloor+1) - me^2 - z} \not\equiv \pm 1 \bmod n$.

If $(a^2)^{\frac{n+1}{8}-(\lfloor\sqrt{n}\rfloor+1)-me^2-z} \equiv \pm 1 \bmod n$ then $1 < \gcd(a^{\frac{n+1}{8}-(\lfloor\sqrt{n}\rfloor+1)-me^2-z} \pm 1 \bmod n, n) < n$ provided $a^{\frac{n+1}{8}-(\lfloor\sqrt{n}\rfloor+1)-me^2-z} \not\equiv \pm 1 \bmod n$.

If $a^{\frac{n+1}{8}-(\lfloor\sqrt{n}\rfloor+1)-me^2-z} \equiv \pm 1 \bmod n$ then proceed as in section 4.6.

In this scenario the number of iterations required also decreases. Applied to the example in section 4.5 no iterations are needed as c appears in the precomputation phase.

Improvements 6 and 7 suggest a generalisation for the construction of array A. However, in the case $16|(n+1)$ it does not necessarily follow that $16|(p+q)$. p of the form $16g+3$ and q of the form $16f+5$ provide a counter example.

9 Conclusion

9.1 Summary

The new algorithm, RAK, for factoring numbers of the form $n = pq$, p and q distinct primes has been presented. The speed of the algorithm is largely dependent on the difference $q - p$, of primes p and q. However the base element "a" and its order mod n also play a part as indicated in section 4.3. One advantage of the algorithm is the terminating conditions in the precomputation phase. In addition a number of techniques can be used to speed up the algorithm making it more efficient, as detailed in section 8. The algorithm offers significant improvement over the Fermat factoring algorithm especially where the difference between p and q is significant. However, it should be noted that RAK like Fermat, is most efficient where p and q are close. RAK can fail but does so with small probability.

9.2 Future Work

There are some areas of the algorithm that need more investigation and these will be the subject of future work. There is more investigation to be done on determining the optimal size of the array A and giving this aspect a more theoretical basis. In addition, the choice of the base element in the array and the trade off between its size and its order modulo n needs further research. We plan to investigate the instances of failure of RAK and determine the type of numbers that cause failure. We also plan to adapt the algorithm for composite moduli of the form p^2q [13] and pqr as these have been proposed as alternative moduli for RSA type algorithms. Finally we will invetigate the methods to improve the running time of the algorithm as described in section 8.

References

[1] A.K. Lenstra and H.W. Lenstra, Jr., *The Development of the Number Field Sieve*, Spring-Verlag, 1993.

[2] Chales P. Pfleeger, *Security in Computing*, Prentice Hall, 1997.

[3] David M. Burton, *Elementary Number Theory*, 1989, Wm. C.Brown.

[4] Hans Riesel, *Prime Numbers and Computer Methods for Factorization*, Birkhauser, 1985.

[5] H.E. Rose, *A Course in Number Theory*, Oxford Science Publications, 1994.

[6] I.M. Vinogradove, *Elements of Number Theory*, Dover, 1954.

[7] J.H. Loxton, *Number Theory and Cryptography*, Cambridge University Press, 1990.

[8] M. Abramowitz and I.A. Stegun, *Handbook of Mathematical Functions With Formulas, Graphs and Mathematical Tables*, U.S. Govt. Print. Off. 1964.

[9] N. Koblitz, *A Course in Number Theory and Cryptography*, Springer, 1991.

[10] R.L. Rivest, A. Shamir and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*; Communications of the ACM, Vol. 21, Nr.2, 1978, S.120–126.

[11] R.A. Mollin, *Fundamental Number Theory with Applications*, CRC Press, 1998.

[12] R.A.Mollin, *RSA and Public-Key Cryptography*, CRC Press, 2003.

[13] T. Okamoto and S. Uchiyama, An efficient public-key cryptosystem, in *Advances in Cryptology—Eurocrypt 98*, Springer-Verlag, 1998.

(Received 5 Feb 2004)