Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.18576/amis/110130

Factoring RSA Modulus with Primes not Necessarily Sharing Least Significant Bits

Hatem M. Bahig[*], *Dieaa I. Nassr and Ashraf Bhery*

Computer Science Division, Department of Mathematics, Faculty of Science, Ain Shams University, Cairo, Egypt

Received: 7 Aug. 2015, Revised: 19 Nov. 2015, Accepted: 22 Nov. 2015
Published online: 1 Jan. 2017

Abstract: The security of many public-key cryptosystems, such as RSA, is based on the difficulty of factoring a composite integer. Until now, there is no known polynomial time algorithm to factor any composite integer with classical computers. In this paper, we study factoring n when $n = pq$ is a product of two primes p and q satisfying that $p \equiv l^{k_1} \mod 2^{\theta_1}$ and $q \equiv l^{k_2} \mod 2^{\theta_2}$ for some positive integers $\theta_1, \theta_2, k_1, k_2 \leq \log n$ and l. We show that n can be factored in time polynomial in $\log n$ if $l < 2^{\theta}$ and either $|\frac{p-l^{k_1}}{2^{\theta_1}}||\frac{q-l^{k_2}}{2^{\theta_2}}| < l^k$ or $2^{\theta'} \geq n^{1/4}$, where $\theta = \min\{\theta_1, \theta_2\}$, $\theta' = \max\{\theta_1, \theta_2\}$ and $k = \min\{k_1, k_2\}$. We also show that the result of Steinfeld and Zheng [21] when the two primes p and q share least significant bits is a special case of our results. Our results point out the warring for cryptographic designers to be careful when generating primes for the RSA modulus

Keywords: Factoring Problem, RSA, Coppersmith's method, square root, least significant bits

1 Introduction

The integer factorization problem is to find a nontrivial factor p of a given composite integer n. It has received a lot of attention among mathematicians and computer scientists for the following reasons [1][15]:

1. It is one of fundamental problems in mathematics, in particular in Number Theory.
2. Its theoretical complexity is unknown. Its decision version : "has N a factor less than M?"; is known to belong to both NP and coNP [2]. From quantum complexity theoretic point of view, Shor [17] showed that integer factorization problem is in BQP (bounded error quantum polynomial time).
3. The security of many public-key cryptosystems and protocols is based on the difficulty of factoring an integer. Among them, the RSA cryptosystem [16]. It was invented by Rivest, Shamir and Adleman in 1978 and is currently the most widely known and widely used public key cryptosystem.
   In RSA, we choose randomly two large distinct primes p and q of the same bit-size with $p > q$. Then we compute $n = pq$ and $\phi(n) = (p-1)(q-1)$. The number n is called the modulus and $\phi(n)$ is called Euler's totient function. Then we choose an integer e such that $1 < e < \phi(n)$, and $\gcd(e, \phi(n)) = 1$. We calculate the multiplicative inverse d of e in $\mathbf{Z}_{\phi(n)}$, i.e., $ed \equiv 1 \mod \phi(n)$. The integer e is called the public key exponent, while the integer d is called the private key exponent. The integers p, q and $\phi(n)$ should be secrete or destroyed.
   To encrypt a message $m \in \mathbf{Z}_n^\star$ one computes $c \equiv m^e \mod n$ using the public key (n, e). To recover the message m, one computes $c^d \mod n$ using the private key d.
   The security of RSA is based mainly on factoring the modulus n. If someone factor n, then he/she will know p and q and can compute $\phi(n)$ and so d. In other words, if a fast factoring algorithm were invented, then RSA and many public-key cryptosystems would fall apart.

There are many factoring algorithms in literatures. They can be classified into two main types. The first one is called *special-purpose factoring* algorithms; they work fast on a positive integer n with factors that have some, special, properties. The running time of this type of factoring algorithms mainly depends on the size of the factors which they find. Table 1 shows examples of special-purpose factoring algorithms and their complexity times assuming that $p \leq \sqrt(n)$. The other type of

[*] Corresponding author e-mail: h.m.bahig@gmail.com

factoring algorithms is called general-purpose factoring algorithms; they work on any positive integer n. The running time of this type of factoring algorithms depends mainly on the size of n. Table 2 shows examples of general-purpose factoring algorithms and their complexity times.

In general, to factor a positive integer n, we first apply special-purpose factoring algorithms. If they fail to find a factor of n, then we apply general-purpose factoring algorithms.

Table 1: Special purpose factoring algorithms [24][25]

Factorization Algorithm	Complexity Time
Trial division	$O(p(\log n)^2)$
Pollard's ρ-method	$O(p^{1/2}(\log n)^2)$
Pollard's $(P-1)$-method	$O(B\log B(\log n)^2)$ where p is B- smooth
Lenstra's Elliptic Curve Method	$O(exp(c\sqrt{\log p \log\log p})(\log n)^2)$, $c \approx 2$

Table 2: General purpose factoring algorithms [15][13][25]

Factorization Algorithm	Complexity Time
Lehman's method	$O(n^{1/3+\varepsilon})$
Shanks' Square Form Factorization method	$O(n^{1/4})$
Shanks' Class Group method	$O(n^{1/5+\varepsilon})$
Continued Fraction method	$O(exp(c\sqrt{\log n \log\log n}))$, $c=\sqrt{2}\approx 1.41421$
Multiple Polynomial Quadratic Sieve	$O(exp(c\sqrt{\log n \log\log n}))$, $c=\frac{3}{2\sqrt{2}}\approx 1.0606$
General Number Field Sieve	$O(exp(c\sqrt[3]{\log n}\sqrt[3]{(\log\log n)^2}))$, $c=(\frac{64}{9})^{1/3}$
Special Number Field Sieve	$O(exp(c\sqrt[3]{\log n}\sqrt[3]{(\log\log n)^2}))$, $c=(\frac{32}{9})^{1/3}$

Let $(b_t b_{t-1} ... b_1)_2$ with $b_i \in \{0,1\}$, be the binary representation of a positive integer x. Throughout this paper, we use the following notations:

– α-MSB(x) to refer to $b_t b_{t-1} ... b_\alpha$. If $\alpha = \lfloor t/2 \rfloor + 1$, we simply write $MSB(x)$.
– α-LSB(x) to refer to $b_\alpha ... b_2 b_1$. If $\alpha = \lceil t/2 \rceil$, we simply write $LSB(x)$.

In literature, there is a set of algorithms in special factoring algorithms class that concerns to factor $n = pq$ when α-LSB(p) (similarly for α-MSB(p)) satisfy some properties. Mainly these properties come from (1) a special way of selecting the primes p and q, i.e., they satisfy special relations, or (2) obtaining a part of the bits of one of the primes by performing a so-called side-channel attack. For examples:

1. When we know α-MSB(p). Coppersmith [5] showed that $n = pq$ can be factored in polynomial time in $\log n$ if we have an approximation p_0 of p such that $|p - p_0| < n^{1/4}$. That's means we have α-MSB$(p), \alpha \geq \frac{\log p}{2} = \frac{\log n}{4}$. This result is improved by Nassr and *et al.* [14] when the public exponent e is full sized.
2. When we know α-LSB(p). Boneh and *et al.* [3] (also in [11]) showed that $n = pq$ can be factored in polynomial time in $\log n$ given that $p_0 = p \mod r$, and $r \geq n^{1/4}$. For example, if $r = 2^\alpha, \alpha \geq \frac{\log n}{4}$, then p_0 is α-LSB(p).
3. When MSB(p) = MSB(q). De Weger [23] noted that Fermat's factoring algorithm [19] takes time $O(1)$ when the difference between the two primes $p - q < n^{1/4}$. Han and Xu [8] slightly improved this result to $|ip - jq| < n^{1/4}$ for two expected small integers i, and j.
4. When LSB(p) = LSB(q). Steinfeld and Zheng [21] showed that $n = pq$ can be factored in polynomial time in $\log n$ if p and q have exactly α equal least significant bits, where $\alpha \geq \frac{\log n}{4}$. Sun *et al.* [22] slightly improved the bound α to $\alpha \geq \frac{\log n}{4} - \frac{7}{2}$.

In this paper, we are interesting to study the factorization of n when α_1-LSB(p) and α_2-LSB(q) satisfy that α_1-LSB$(p) = l^{k_1}$ and α_2-LSB$(q) = l^{k_2}$ for some some positive integers l, k_1, k_2. In case of $k_1 = k_2$, then we get the result of Steinfeld and Zheng [21], i.e., the two primes share the least significant bits.

This paper is organized as follows. In Section 2, we review some facts used in our results The proposed factoring method with some examples are presented in Section 3. Finally, Section 4 includes the conclusion.

2 Preliminaries

In this section, we mention two results needed in this paper. The first one is a variant of Coppersmith's result [5] on integer factorization when we know an approximation p_0 of p such that $|p - p_0| \leq n^{1/4}$, where $n = pq$ is a product of two primes $p > q$. The second result is related to solving $x^2 \equiv a \mod m$ for some integers a, and m.

2.1 Factoring with Partial information of prime factors

The first result that we need states that n can be factored in polynomial time when the least-significant bits are given as a remainder modulo a positive integer m.

Theorem 1.([3][11]) *Let $n = pq$, where p and q are of the same bit-size with $p > q$. Suppose that we know p_0 and m satisfying $p_0 = p \mod m$ and $m \geq n^{1/4}$. Then we can find the factorization of n in time polynomial in $\log n$.*

2.2 Square-Roots Modulo a power of 2

The second result that we need in the paper is finding square roots modulo a power of 2.

Theorem 2. *[20]*

1. *Let $a \equiv 1 \mod 8$ and $k \geq 3$. Then there are exactly four solutions in Z_{2^k} to the congruence $x^2 \equiv a \mod 2^k$. These solutions are of the form $x = \pm s + b.2^{k-1}$ with $b \in \{0,1\}$ and s is any solution to $x^2 \equiv a \mod 2^{k-1}$. Furthermore, there exists an algorithm that, given a and k, computes these four solutions in time $O(k^2)$ bit operations.*
2. *The set of solutions in Z_{2^k} to the modular equation $x^2 \equiv c \mod 2^k$ is summarized as follows. Let $c = 2^u v$ where v is odd.*
   (a) *If $k \leq u$, there are $2^{\lfloor k/2 \rfloor}$ solutions $x \equiv 0 \mod 2^{\lceil k/2 \rceil}$.*
   (b) *If $k > u$, there are no solutions if u is odd. Otherwise, if u is even, there are three subcases*
      – *If $k = u+1$, there are $2^{u/2}$ solutions $x \equiv 2^{u/2} \mod 2^{u/2+1}$.*
      – *If $k = u+2$, there are $2.2^{u/2}$ solutions $x \equiv \pm 2^{u/2} \mod 2^{u/2+2}$ if $v \equiv 1 \mod 4$ and none otherwise.*
      – *If $k \geq u+3$, there are $4.2^{u/2}$ solutions of the form $x \equiv 2^{u/2}(\pm s + b.2^{k-u-1}) \mod 2^{k-u/2}$ with $b \in \{0,1\}$ and s is any solution to $s^2 \equiv v \mod 2^{k-u}$ if $v \equiv 1 \mod 8$ and no solutions if $v \neq 1 \mod 8$.*

Theorem 2 can be summarized in Algorithm 1.

Algorithm 1

Input: *c (odd), k.*
Output: *a set $S = \{v : v^2 \equiv c \mod 2^k\}$.*
  $S \leftarrow$ *empty*
  if $c \mod 8 \neq 1$ *then return* S
  if $k = 1$ *then return* $S \leftarrow \{1\}$
  if $k = 2$ *then return* $S \leftarrow \{1,3\}$
  $v_1 \leftarrow 1, v_2 \leftarrow 3, v_3 \leftarrow 5, v_4 \leftarrow 7$
  $m \leftarrow 8$
  for $i = 4$ *to* k *do*
    for $j = 1$ *to* 4 *do*
      if $\frac{v_j^2 - c}{m} \mod 2 = 1$ *then* $v_j \leftarrow v_j + m/2$
    $m \leftarrow 2m$
  return $S \leftarrow \{v_1, v_2, v_3, v_4\}$

3 Factoring n

In this section, we study the factorization of $n = pq$ when the primes p and q satisfy the following: $p \equiv l^{k_1} \mod 2^{\theta_1}$, $q \equiv l^{k_2} \mod 2^{\theta_2}$, for some integers $\theta_1, \theta_2, k_1, k_2, l$. This study extends Steinfeld and Zheng method [21] to factor n when $LSB(p) = LSB(q)$.

Theorem 3. *Let $n = pq$ be a product of two primes p and q. Suppose that there are four positive integers θ_1, θ_2, k_1 and k_2 less than or equal to $\log n$ and satisfying $p \equiv l^{k_1} \mod 2^{\theta_1}$ and $q \equiv l^{k_2} \mod 2^{\theta_2}$ for some unknown positive integer $l < 2^{\theta}$ where $\theta = \min\{\theta_1, \theta_2\}$. If $|\frac{p-l^{k_1}}{2^{\theta_1}}||\frac{q-l^{k_2}}{2^{\theta_2}}| \leq l^k$, $k = \min\{k_1, k_2\}$, then we can find l to factor n in time polynomial in $\log n$.*

Proof. The proof consists of two steps:

1. We show that l can be found in polynomial time.
2. We show that n can be factored in time polynomial in $\log n$.

Now, we prove the first step.
Since $p \equiv l^{k_1} \mod 2^{\theta_1}$, $q \equiv l^{k_2} \mod 2^{\theta_2}$, and $\theta = \min\{\theta_1, \theta_2\}$, we have $p \equiv l^{k_1} \mod 2^{\theta}$ and $q \equiv l^{k_2} \mod 2^{\theta}$. Thus,

$$n \equiv l^{k_1+k_2} \mod 2^{\theta}.$$

Therefore, l is one of the solutions to the modular equation $x^{k_1+k_2} \equiv n \mod 2^{\theta}$. The integer l can be determined as follows.

Case 1: If $k_1 + k_2$ is odd, then l is unique and can be computed as follows:

$$l = n^t \mod 2^{\theta}, \text{ where } t = (k_1+k_2)^{-1} \mod 2^{\theta-1}$$

Case 2: If $k_1 + k_2$ is even, then $k_1 + k_2$ can be written as $k_1 + k_2 = 2^i r$, where $i \geq 1$, and r is odd. Thus, the solutions of the modular equation $x^{k_1+k_2} \equiv n \mod 2^{\theta}$ can be obtained by computing the square roots i-times for b, where

$$b \equiv n^a \mod 2^{\theta},$$
$$a = r^{-1} \mod 2^{\theta-1}.$$

Note that $x^{2^i} \equiv b \mod 2^{\theta}$. By Theorem 2, there are four roots can be computed in polynomial time in $\log n$ for each square root computation. Thus, in total, there are 4^i roots of the modular equation. Hence, we can find l in polynomial time in $\log n$ since $4^i < (k_1+k_2)^2$ which is bounded by a polynomial time in $\log n$.

Therefore, finding l takes polynomial time in $\log n$.

Now, we prove the second step, i.e., given l, n can be factored in polynomial time in $\log n$.
Since $p \equiv l^{k_1} \mod 2^{\theta_1}$, then there is an integer λ_p satisfies

$$p = 2^{\theta_1}\lambda_p + l^{k_1} \tag{1}$$

Similarly, since $q \equiv l^{k_2} \mod 2^{\theta_2}$, then there is an integer λ_q satisfies

$$q = 2^{\theta_2}\lambda_q + l^{k_2} \tag{2}$$

Using Eq.(1) and (2), we have

$$(p+q) - (l^{k_1} + l^{k_2}) = 2^{\theta_1}\lambda_p + 2^{\theta_2}\lambda_q \tag{3}$$

and

$$(p-q)-(l^{k_1}-l^{k_2})=2^{\theta_1}\lambda_p-2^{\theta_2}\lambda_q \quad (4)$$

By subtracting the squaring of Eqs.(3) and (4), we get

$$4n+4l^{k_1+k_2}-4(pl^{k_2}+ql^{k_1})=2^{\theta_1+\theta_2+2}\lambda_p\lambda_q$$

and so

$$pl^{k_2}+ql^{k_1}=n+l^{k_1+k_2}-2^{\theta_1+\theta_2}\lambda_p\lambda_q \quad (5)$$

Now set $a=pl^{k_2}+ql^{k_1}$. Thus

$$pl^{k_2}-ql^{k_1}=\sqrt{a^2-4nl^{k_1+k_2}}.$$

It follows that

$$2pl^{k_2}=a+\sqrt{a^2-4nl^{k_1+k_2}}.$$

Therefore, the computation of $\gcd(a+\sqrt{a^2-4nl^{k_1+k_2}},n)=\gcd(2pl^{k_2},n)$ returns a non-trivial divisor, p, of n. The factorization of n takes a polynomial time in $\log n$ since

1. l is found in polynomial time in $\log n$.
2. $\lambda_p\lambda_q$ can be computed in polynomial time in $\log n$ since by Eqs.(1) and (2) and the assumption, we have

$$|\lambda_p||\lambda_q|=|\frac{p-l^{k_1}}{2^{\theta_1}}||\frac{q-l^{k_2}}{2^{\theta_2}}|\leq l^k$$

also, we have

$$\lambda_p\lambda_q=\begin{cases}1, & \text{if } l=1;\\ 2^{-(\theta_1+\theta_2)}n \mod l^k, & \text{if } \lambda_p \text{ and } \lambda_q \text{ have the same sign;}\\ (2^{-(\theta_1+\theta_2)}n \mod l^k)-l^k, & \text{otherwise.}\end{cases} \quad (6)$$

Note that, from Eqs.(1) and (2), we have

$$n\equiv 2^{\theta_1+\theta_2}\lambda_p\lambda_q \mod l^k$$

3. computing $l^{k_1+k_2}$ is in polynomial time in $\log n$ since $l<2^{\theta}$ and $\theta,k_1,k_2\leq\log n$.

Remark. 1.The integer l should be odd; otherwise p and q are not primes since $\theta_1,\theta_2\geq 1$

2.To determine a set of solutions to $x^t\equiv c \mod 2^k$, one can use the following algorithm.

Algorithm 2

Input: *$c(odd),t,k$.*
Output: *a set $S=\{v: v^t\equiv c \mod 2^k\}$.*
$S\leftarrow empty$
if t is odd then
  $r\leftarrow t^{-1} \mod 2^{k-1}$
  $v\leftarrow c^r \mod 2^k$
  return $S\leftarrow\{v\}$
if $c \mod 8\neq 1$ then return S
compute s,u where $t=2^u s$, s is odd
$r\leftarrow s^{-1} \mod 2^{k-1}$
$c_0\leftarrow c^r \mod 2^k$
set $S\leftarrow\sqrt{c_0} \mod 2^k$ (Algorithm 1)
for $j=1$ to $u-1$ do
  for every $v\in S$ define a set $S_v\leftarrow\sqrt{v} \mod 2^k$.
  $S\leftarrow\bigcup_v S_v$
return S.

3.We have n is odd, due to Theorem 2, if $n\neq 1 \mod 8$, then k_1+k_2 cannot be even.

4.It is not necessary that the prime factors p and q have the same bit-size .

Proposition 1.*Suppose that we have the same assumptions as in Theorem 3 but $k_1=k_2=0$. Then n can be factored in polynomial time in $\log n$.*

*Proof.*Since $k_1=k_2=0$ and $|\lambda_p||\lambda_q|=|\frac{p-l^{k_1}}{2^{\theta_1}}||\frac{q-l^{k_2}}{2^{\theta_2}}|\leq l^k=1$, then $\lambda_p=\lambda_q=1$. Thus p, and q have the form $p=2^{\theta_1}+1$, $q=2^{\theta_2}+1$. Therefore, p, and q can be easily computed in polynomial time since θ_1 and θ_2 are known.

Proposition 2.*Suppose that we have the same assumptions as in Theorem 3 but $k_1=0$ and $k_2\neq 0$ (or $k_2=0$ and $k_1\neq 0$). Then n can be factored in polynomial time in $\log n$.*

*Proof.*Suppose that $k_1=0$, and $k_2\neq 0$ (similarly for $k_1\neq 0$, $k_2=0$). Then we have either $|\lambda_p||\lambda_q|=0$ or $|\lambda_p||\lambda_q|=1$.

Case1:$|\lambda_p||\lambda_q|=0$. Since p, and q are two primes, then we have $\lambda_p\neq 0$ (otherwise we get $p=1$ which is a contradiction), and so $\lambda_q=0$. This implies that $p=\lambda_p 2^{\theta_1}+1$, and $q=l$. Since $l<2^{\theta}\leq 2^{\theta_1}$, and $n=pq=l\lambda_p 2^{\theta_1}+l$, then q can be directly computed from $n \mod 2^{\theta_1}=l$.

Case 2:$|\lambda_p||\lambda_q|=1$. Since p is prime, and $k_1=0$ we have $\lambda_p=1$, and so $p=2^{\theta_1}+1$. Therefore, p can be easily computed since θ_1 is known.

In both cases, n can be factored in polynomial time in $\log n$.

Example: Using NTL [18], we give an example for Theorem 3. We use the same symbols as in the theorem. Let $n=pq$ be 1024-bits,

$n=$ 15677276905294754645933151284582381211894469329571389619962733\
8473005656832699342267722370814161978050156563599629284579518\
2190297484702220060989004151405452194801462542294392500699908\
2157449079433751851733036982265355062485300851975033210006530\
8137806494728404783991033423157610105156954758450466558120152\
1281.

The parameters $k_1=87,k_2=83$, $\theta_1=80,\theta_2=70$. Since $k_1+k_2=2\times 85$, the modular equation $l^{k_1+k_2}\equiv n \mod 2^{\theta}$ has the following four solutions in 2^{θ} :

1. 65
2. 590295810358705651647
3. 1180591620717411303359
4. 590295810358705651777

Take $l=65$. By Eq.(6), $l\neq 1$, there are two candidates for $\lambda_p\lambda_q$

1. 898527611039292061965791483095663564517177094.
2. -296352504163715350769902319249212973281606390600197918 0198\
505543937845680342886558375307305890837153455573048 60993580\
26091502809478224601569005707135 31.

Let

$\lambda_p\lambda_q = 898527611039292061965791483095663564517177094.$

Then

$a =$ 313545538105895092918663025691647624237889386591427 7923992547\
694601131366539868453544474162832395610031312719272 7781429638\
963276463808054518126987536275367496992648979026099 4218420700\
690842563150795369889846698935964483041651541289147 2114083434\
818006866043358625572873982223498501298308838158845 9014892578\
1250.

and so $a+\sqrt{a^2-4nl^{k_1+k_2}}$ is

5882961654924967604893904353314715990177113584954889394207416\
6709494560258896682116197257107461067988024397625736877359729\
00390625000000000000000000000000000000000000000000000000000000\
0000000000000000.

Computing $\gcd(a+\sqrt{a^2-4nl^{k_1+k_2}},n)$ returns the prime factor q

$q =$ 296352504163715350769902319249212973281606390600197918 0198505\
543937845680342886558375307305890837153455573948310 8519185189\
06923924673804900657572398273.

Therefore, the other prime factor, p, of n is

$p =$ 529007741963742133333698758754798233118497507620765 7960353094\
802525550654767073937109821997721762496240290469121 5171251748\
16896482829162866187250286180472569 7.

Note that $p \equiv l^{k_1} \mod 2^{\theta_1}$ and $q \equiv l^{k_2} \mod 2^{\theta_2}$ ◇

We summarize the results of Theorem 3 and Propositions 1 and 2 in Algorithm 3. Given n with unknown prime factors, and four parameters $\theta_1,\theta_2,k_1,k_2$, the algorithm returns the prime factor p of n if these parameters satisfy the conditions of Theorem 3, Proposition 1 or Proposition 2. Otherwise it returns zero.

Algorithm 3

Input: $n,\theta_1,\theta_2,k_1,k_2$
Output: *a prime factor p of n (success) or 0 (failure).*
  if $k_1=k_2=0$ *then*
    if $2^{\theta_1}+1$ *divides n then return* $p=2^{\theta_1}+1$ *(success)*
    else return 0 (failure)
  else if $k_1=0$ *and* $k_2\neq 0$ *then*
      if $n \mod 2^{\theta_1}$ *divides n then return* $p=n \mod 2^{\theta_1}$ *(success)*
      else if $2^{\theta_1}+1$ *divides n then return* $p=2^{\theta_1}+1$ *(success)*
      else return 0 (failure)
  else if $k_1\neq 0$ *and* $k_2=0$ *then*
      if $n \mod 2^{\theta_2}$ *divides n then return* $p=n \mod 2^{\theta_2}$ *(success)*
      else if $2^{\theta_2}+1$ *divides n then return* $p=2^{\theta_2}+1$ *(success)*
      else return 0 (failure)
  $\theta\leftarrow\min\{\theta_1,\theta_2\}$
  $k\leftarrow\min\{k_1,k_2\}$
  a set $S\leftarrow \sqrt[k_1+k_2]{n} \mod 2^{\theta}$ *(Using Algorithm 2.)*
  for every $l\in S$ *do*
    $m\leftarrow l^k$
    if l=1 then $\lambda\leftarrow 1$
    else $\lambda\leftarrow 2^{-(\theta_1+\theta_2)}n \mod m$
    for i=1 to 2 do
      $a\leftarrow n+l^{k_1+k_2}-2^{\theta_1+\theta_2}\lambda$
      if $a^2>4nl^{k_1+k_2}$ *then*
        $b\leftarrow\sqrt{a^2-4nl^{k_1+k_2}}$
        $p\leftarrow\gcd(n,a+b)$
        if $p\neq 1$ *and* $p\neq n$ *then return p (success)*
      $\lambda\leftarrow\lambda-m$
  return 0 (failure)

In order to speed up the attack (Algorithm 3), one can skip some l's that cannot lead to a prime factorization. The following result determines a bound for l.

Proposition 3. *Suppose that we have the same assumption as in Theorem 3. A necessary condition to find l that lead to a prime factorization is*

$$max\{k_1,k_2\}\log l \leq \max\{\theta_1+\theta_2+(k+1)\log l, \log p+\log l\}.$$

Proof. Let $k'=\max\{k_1,k_2\}$. Since we have either $l^{k'-1}>p$ or $l^{k'-1}\leq p$.
If $l^{k'-1}>p$, then

$$\frac{l^{k'-1}}{2^{\theta_1+\theta_2}} < \frac{l^{k'-1}(l-1)}{2^{\theta_1+\theta_2}} = \frac{l^{k'}-l^{k'-1}}{2^{\theta_1}}\frac{1}{2^{\theta_2}}$$
$$< |\frac{p-l^{k_1}}{2^{\theta_1}}||\frac{q-l^{k_2}}{2^{\theta_2}}| < l^k.$$

Thus, $k'\log l < \theta_1+\theta_2+(k+1)\log l$.
If $l^{k'-1}\leq p$, then

$$k'\log l \leq \log p+\log l.$$

Therefore,

$$k'\log l \leq \max\{\theta_1+\theta_2+(k+1)\log l, \log p+\log l\}.$$

Remark. 1. In general, the value of $\log p$ is unknown. Its bound is $\log n^{1/2} < \log p < \log n$ (or $1<\log q<\log n^{1/2}$). But in RSA cryptosystem, n is a product of two primes of the same bit-size. Thus, $\log p = \frac{\log n}{2}$.
2. In practice, it is easy to check that whether $l^{k'-1}=p$ (in RSA, $l=p$) by dividing n by l before going to determine the correct l.

In the following theorem, we study a case similar to Theorem 3 but n is a product of two primes of the same bit-size (as in RSA) and we replace the condition $|\frac{p-l^{k_1}}{2^{\theta_1}}||\frac{q-l^{k_2}}{2^{\theta_2}}| < l^k$ with $2^{\theta'}\geq n^{1/4}$, $\theta'=\max\{\theta_1,\theta_2\}$.

Theorem 4. *Let* $n=pq$ *be a product of two primes p and q of the same bit-size with* $p>q$. *Suppose that there are four positive integers* θ_1,θ_2,k_1 *and* k_2 *less than or equal to* $\log n$ *and satisfying* $p\equiv l^{k_1} \mod 2^{\theta_1}$ *and* $q\equiv l^{k_2} \mod 2^{\theta_2}$ *for some unknown positive integer* $l<2^{\theta}$ *where* $\theta=\min\{\theta_1,\theta_2\}$. *If* $2^{\theta'}\geq n^{1/4}$, $\theta'=\max\{\theta_1,\theta_2\}$, *then we can find l to factor n in time polynomial in* $\log n$.

*Proof.*The proof of the first part of the theorem, i.e., finding l, is the same as in Theorem 3.

Now we show that l can be used to factor n in time polynomial in $\log n$.

We have either $\theta' = \theta_1$ or $\theta' = \theta_2$.

Case 1:$\theta' = \theta_1$. We have $p \equiv l^{k_1} \mod 2^{\theta'}$ and $2^{\theta'} \geq n^{1/4}$. Therefore, we can apply Theorem 1 by taking

$$p_0 = l^{k_1} \mod 2^{\theta'}$$

and $m = 2^{\theta'} \geq n^{1/4}$ to factor n in time polynomial in $\log n$.

Case 2:$\theta' = \theta_2$. We have $q \equiv l^{k_2} \mod 2^{\theta'}$, and so, $p \equiv nl^{-k_2} \mod 2^{\theta'}$ and $2^{\theta'} \geq n^{1/4}$. Therefore, we can apply Theorem 1 by taking

$$p_0 = nl^{-k_2} \mod 2^{\theta'}$$

and $m = 2^{\theta'} \geq n^{1/4}$ to factor n in time polynomial in $\log n$.

Example: Using NTL [18], we give an example for Theorem 4. We use the same symbols as in the theorem. Let $n = pq$ be 1024-bits,

$n =$ 1314593070978057234392159450005339380862008554202222803551122\
9601450930582535189684113824182982945655478574879276932141846\
3894820913167012516035218668368716534516522624183155148958768\
8856215617365045505631932444897766092772422045071340919415561\
9390469640589491346548588064097885053401836186308668732959070\
7371.

The parameters $k_1 = 7, k_2 = 4,\ \theta_1 = 270, \theta_2 = 65$. Since $k_1 + k_2 = 11$, the modular equation $l^{k_1+k_2} \equiv n \mod 2^{\theta}$ has exactly one solution $l = 4494515958772142739$. By taking

$$p_0 = l^{k_1} \mod 2^{\theta_1} = 4494515958772142739^7 \mod 2^{270}$$

$$m = 2^{\theta_1} = 2^{270} > n^{1/4}$$

By using Theorem 1, we get

$p =$ 1158361683689280660182536292485488975550382241951107177632174\
3018809828986043788856498670259547708549666353475147515382884\
26143097251552476022072329641061

$q =$ 1134872716776329538646715142653894638967505025453806813043049\
4857726792580703282951027388963453337189960553239804579712433\
19023068315392542864914483735220 9

$\diamondsuit$

In the following corollary, we show that the result of Steinfeld and Zheng [21] is a special case of Theorem 4

Corollary 1.*Let $n = pq$ be a product of two large primes where p and q are of the same bit-size. If $LSB(p) = LSB(q)$, then n can be factored in time polynomial in $\log n$.*

*Proof.*Let $l = LSB(p) = LSB(q)$. Then we can write $p = 2^{\theta} p_h + l$ and $q = 2^{\theta} q_h + l$ where θ is the bit-size of l. Thus, $0 < l < 2^{\theta}$ and $2^{\theta} \geq n^{1/4}$. Therefore, we apply Theorem 4 by taking $\theta_1 = \theta_2 = \theta$ and $k_1 = k_2 = 1$ to factor n in time polynomial in $\log n$.

Corollary 2.*Let $n = pq$ be the RSA modulus, where p and q are two large primes of the same bit-size. Suppose that $p \equiv l^{k_1} \mod 2^{\theta_1}$ and $q \equiv l^{k_2} \mod 2^{\theta_2}$ for some positive integers $k_1, k_2, \theta_1, \theta_2 \leq \log n$ and $l < 2^{\theta}$, $\theta = \min\{\theta_1, \theta_2\}$. Then n can be factorized in time polynomial in $\log n$ in the following cases:*

1.$|\frac{p-l^{k_1}}{2^{\theta_1}}||\frac{q-l^{k_2}}{2^{\theta_2}}| \leq l^k$, $k = \min\{k_1, k_2\}$,
2.$2^{\theta'} \geq n^{1/4}$, $\theta' = \max\{\theta_1, \theta_2\}$,

*Proof.*The proof comes directly from Theorems 3 and 4

4 Conclusion

In this work, we extended the attack of Steinfeld and Zheng [21] on RSA when the two primes p and q have equal least significant bits. We have studied factoring n in polynomial time in $\log n$ when the least significant bits of p and q can be written as a power (not necessarily the same) of a positive integer l modulo a power (not necessarily the same) of 2.

References

[1] E. Bach, Annual Reviews Comput. Sci., 119-127 (1990).

[2] K. Bimpikis, and R. Jaiswal, A Technical Report Presented to the University of California, San Diego, 1-15 (2005).

[3] D. Boneh, G. Durfee, Y. Frankel, Lecture Notes in Computer Science **1514**, 2534 (1998).

[4] R. P. Brent, Lecture Notes in Computer Science **1685**, 1-22 (1999).

[5] D. Coppersmith, Journal of Cryptology **10(4)**, 223-260 (1997).

[6] J. Faugère, R. Marinier, and G. Renault, Lecture Notes in Computer Science **6056**, 70-87 (2010).

[7] Y. Filmus, Factorization Methods: Very Quick Overview (2010), http://www.cs.toronto.edu/ yuvalf/Factorization.pdf.

[8] L. Han, G. Xu, Asia-Pacific Conference on Information Processing (APCIP 2009), 445-449 (2009).

[9] H. W. Jr. Lenstra, Ann. Math. **126**, 649-673 (1987).

[10] A. K. Lenstra, H. W. Jr. Lenstra, The Development of the Number Field Sieve, Springer-Verlag (1993).

[11] A. May, New RSA vulnerabilities using lattics reduction methods, Ph.D. Dissertation. University of Paderborn. Germany (2003).

[12] A. May, M. Ritzenhofen, Lecture Notes in Computer Science **5443**, 1-14 (2009).

[13] P.L. Montgomery, A survey of modern integer factorization algorithms, Techincal Report at CWI, Amsterdam (1995).

[14] D. Nassr, H. Bahig, A. Bhery, and S. Daoud, Proceeding of the 6th ACS/IEEE International Conference on Computer Systems and Applications, 694 701 (2008).

[15] C. Pomerance, Proceedings of symposia in applied mathematics **42**, 27-47 (1990).

[16] R. Rivest, A. Shamir, L. Adleman, Communication of ACM **21**, 120-126 (1978).

[17] P. Shor, SIAM J. Comput. **26(5)**, 1484-1509 (1997)

[18] V. Shoup, NTL: A Library for doing Number Theory, online available at http:// www.shoup.net/ntl/index.html.

[19] J. H. Silverman, A friendly introduction to number theory, 2nd edition, Prentice Hall (2001).

[20] R. Steinfeld and Y. Zheng, Lecture Notes in Computer Science **2020**, 52-62 (2001).

[21] R. Steinfeld, Y. Zheng, Appl. Algebra Eng. Commun. Comput. **15**, 179-200 (2004).

[22] H. SUN, M. WU, C. YANG, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, **E92.A(8)**, 2137-2138 (2009).

[23] B. de Weger, Appl. Algebra Eng. Commun. Comput., **13(1)**, 17-28 (2002).

[24] R. V. Yampolskiy, Int. J. Bio-Inspired Computation, **2(2)**, 115-123 (2010).

[25] Song Y. Yan, Primality Testing and Integer Factorization in Public-Key Cryptography. 2nd edition, Springer (2009).

Hatem M. Bahig received the MSc degree in Computer Science from Faculty of Science, Ain Shams University, Egypt, in 1998. In 2003, he received the DSc degree from Graduate School of Science, Tokyo Metropolitan University, Japan. His research interests includes computational number theory, and cryptography.

Dieaa I. Nassr received the MSc degree in Cryptography at Faculty of Science, Ain Shams University. Now, he is a PhD student and is working a assistant lecturer at Faculty of Science, Ain Shams University. His research interests includes cryptography, discrete mathematics, and computational number theory.

Ashraf M. Bhery is a lecturer of Computer Science at Ain Shams University, Faculty of Science (Egypt). He received the PhD degree in Computer Science at Tokyo Institute of Technology (Japan). His research interests are: Cryptography, Formal Methods, Computational Intelligence, and Models of Computations.