Certifying Variant of RSA with Generalized Moduli

Yao Lu[1], Noboru Kunihiro[1], Rui Zhang[2,4](✉), Liqiang Peng[2,3](✉), and Hui Ma[2](✉)

[1] The University of Tokyo, Tokyo, Japan
[2] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China
{r-zhang,pengliqiang,mahui}@iie.ac.cn
[3] Data Assurance and Communication Security Research Center, Chinese Academy of Sciences, Beijing 100093, China
[4] School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China

Abstract. Let N be an arbitrary integer with unknown factorization. In Asiacrypt 2012, Kakvi et al. proposed an algorithm that, given prime $e \geq N^{\frac{1}{4}} + \epsilon$, certifies whether the RSA function $\mathrm{RSA}_{N,e}(x) := x^e \bmod N$ defines a permutation over $\mathbb{Z}_N^*$ or not. In this paper, we extend Kakvi et al.'s work by considering the case with generalized moduli $N = \prod_{i=1}^{n} p_i^{z_i}$. Surprisingly, when $\min\{z_1, \ldots, z_n\} \geq 2$, we show that it can be efficiently decided whether the RSA function defines a permutation over $\mathbb{Z}_N^*$ or not even for the prime $e < N^{\frac{1}{4}}$. Our result can be viewed as an extension of Kakvi et al.'s result.

Keywords: Coppersmith's method · Lattices · RSA
Public key cryptosystem · LLL algorithm

1 Introduction

RSA function is one of the most well known cryptographic primitives, it is defined as $\mathbf{RSA}_{N,e} : \mathbb{Z}_N^* \rightarrow \mathbb{Z}_N^*$, $x \rightarrow x^e \bmod N$, where N is a public modulus and e is an exponent integer. Moreover, it is believed that RSA function (with the appropriate choice of parameters) defines a family of trapdoor permutations, which has a number of applications in public-key cryptosystems.

In Crypto'92, Bellare and Yung [2,3] introduced a new primitive called certified trapdoor permutations, compared with standard trapdoor permutation, it requires an additional efficient permutation checking procedure, roughly speaking, a trapdoor permutation is certified if one can verify from the public key that it is actually a permutation. Using certified trapdoor permutations as a building block, we can construct many useful cryptographic protocols: ZAPS and verfiable PRF [6]; Sequential aggregate signatures [1,12]. More importantly, NIZK protocols for any NP-statement can be built from certified trapdoor permutations [8].

Among all the known candidate trapdoor permutations (factoring-based), RSA trapdoor function is the most efficient certified trapdoor permutation currently known. It is well known that RSA trapdoor function defines a permutation over the domain $\mathbb{Z}_N^*$ iff $\gcd(e, \phi(N)) = 1$ where $\phi(\cdot)$ is Euler's totient function i.e. the number of positive integers less than or equal to N that are coprime to N. So we only need to check whether $\gcd(e, \phi(N)) = 1$ to tell whether RSA trapdoor function defines a permutation or not.

Generally speaking, RSA function is not a certified trapdoor permutation. In [2,3], Bellare and Yung proposed a generalized approach that can transform every trapdoor permutation into a certified trapdoor permutation. Using their method, we can easily make RSA function to be certified, however, since Bellare-Yung transformation brings an additional computational overhead, which makes their method relatively inefficient. Besides, in order to keep the same data structure, we prefer to use directly method rather than artificial method. In [4,12], the authors showed that if prime $e > N$, the RSA function is a certified permutation (since e is a prime and $\phi(N) < N$, thus $\gcd(e, \phi(N)) = 1$), but in practice, using large exponent e ($e > N$) will bring heavy costs for modular exponentiation. Therefore, the question naturally arises: Is the RSA function certified for the case of $e < N$?

There exist several research results on the above problem. Suppose that N is a RSA modulus i.e. $N = pq$ where p and q are of the same bit-size. If prime $e \geq N^{1/4}$, we can efficiently decide whether e divides $\phi(N)$ or not by using Coppersmith's result [5]. On the other hand, if prime $e < N^{1/4}$, it is hard to decide whether e divides $\phi(N)$ or not, which is called Phi-Hiding Assumption, by Cachin, Micali and Stadler [4] in the context of efficient single database private information retrieval, which has found a lot of applications in cryptography.

Later, in Asiacrypt 2008, Schridde and Freisleben [15] showed that the Phi-Hiding Assumption does not hold for special composite integers of the form $N = pq^{2k}$ for $k > 0$. Such integers are often used in cryptography to speed up certain operations [16].

Suppose that N is an arbitrary integer with unknown factorization. Recently, in Asiacrypt 2012, Kakvi et al. [10] proposed an algorithm that, given prime $e \geq N^{1/4+\epsilon}$, efficiently decides whether e divides $\phi(N)$ or not. Kakvi et al. [10] gave an efficient certification procedure that works for any prime exponent $e > N^{1/4}$ (rather than $e > N$). However, until now, if prime $e < N^{1/4}$, we do not know that whether Phi-Hiding Assumption holds or not (we know the results for RSA moduli [4] and moduli of the form pq^{2k} with $k > 1$ [15], but we know nothing for arbitrary integer with unknown factorization).

1.1 Our Contributions

In this paper, given an arbitrary modulus N with unknown factorization and a prime $e < N^{1/4}$, we can extract more information than previously expected, which enable us to efficiently decide whether e divides $\phi(N)$ or not in some circumstances, that means we can further improve [10]'s result in this circumstances.

In particular, using our algorithm, if prime e satisfies $N^{1/4r} < e < N^{\min\{1/4(r-1),1\}}$ (r is a positive integer), we can check whether exists secret factor p s.t. $e|p-1$ and $N \equiv 0 \bmod p^r$. Note that when $r=1$, that is exactly [10]'s result. Thus, our result can be viewed as a generalization of [10]'s result.

Although when $e < N^{1/4}$, we can not directly decide whether $\gcd(e, \phi(N)) = 1$ or not (since we can only check the factor p: $N \equiv 0 \bmod p^r$, r is related to the size of exponent e), we can identify the scenarios of $\gcd(e, \phi(N)) \neq 1$ if moduli of form $N \equiv 0 \bmod p^r$ are used and p hides e. In addition, let $N = \prod_{i=1}^{n} p_i^{z_i}$, for the case of $\min\{z_1, ..., z_n\} \geq 2$, we can improve [10]'s result: for example, for $N = p^2q^3$, we can improve [10]'s result to 1/8. Therefore, using moduli of this form, we can further decrease the size of exponent e while publicly verifying the permutation. However, on the other hand, this indicates that cryptographic schemes using moduli of this form and relying on the Phi-Hiding Assumption must be handled with care.

Our technique is similar to Kakvi et al. [10], we also use Coppersmith's method [5] to find prime divisors p of N in a specific range, and the key problem is to show that the number of invocations of Coppersmith's algorithm in our certification algorithm is polynomial-time.

Like [10]'s algorithm, our algorithm also only works for prime e and how to extend to arbitrary integers e of unknown factorization is still an open problem.

2 Preliminary

2.1 Certified Trapdoor Permutation

Definition 1. *A family of trapdoor permutations is a tripe of algorithms* (G, E, D) *such that:*

- $G(\cdot)$ *is a randomized algorithm that takes no input and generates a pair* (pk, sk), *where* pk *is a public key and* sk *is a secret key;*
- $E(\cdot)$ *is a deterministic algorithm such that, for every fixed public key* pk, *the mapping* $x \to E(pk, x)$ *is a bijection;*
- $D(\cdot)$ *is a deterministic algorithm such that for every possible pair of keys* (pk, sk) *generated by* $G(\cdot)$ *and for every* x *we have*

$$D(sk, E(pk, x)) = x$$

A family of permutations Π is said to be certified if the permutation can be verified in polynomial-time given pk. We follow the definition in [10,12].

Definition 2. $\Pi = (G, E, D, C)$ *is called a family of certified trapdoor permutations if* (G, E, D) *is a family of trapdoor permutations and* $C(\cdot)$ *is a deterministic polynomial-time algorithm such that, for an arbitrary* pk *(potentially not generated by* $G(\cdot)$*), returns* 1 *iff* $(E(pk, \cdot))$ *defines a permutation over domain* Dom_{pk}.

2.2 Coppersmith's Method

Let us introduce Coppersmith's algorithm for finding small roots of modular polynomial equations. Our main algorithm uses Coppersmith's algorithm as subroutine.

Theorem 1 (Coppersmith [5], May [13]). *Let N be an integer of unknown factorization, which has a divisor $p \geq N^{\beta}$, $0 < \beta \leq 1$. Let $0 < \mu \leq \frac{1}{7}\beta$. Furthermore, let $f(x)$ be a univariate monic polynomial of degree δ. Then we can find all solutions x_0 for the equation:*

$$f(x_0) = 0 \bmod p \qquad \textit{with} \qquad |x_0| \leq \frac{1}{2} N^{\frac{\beta^2}{\delta} - \mu}$$

This can be achieved in time $\mathcal{O}(\mu^{-7}\delta^5 \log^2 N)$. The number of solutions x_0 is bounded by $\mathcal{O}(\mu^{-1}\delta)$.

In the rest of our paper, one of our main algorithms is to find small roots of polynomial equation $f(x) = x + a = 0 \bmod p$, where p is unknown that satisfies $N = 0 \bmod p^r$. We can model this problem as the univariate polynomial with degree r:

$$f(x) = (x + a)^r \bmod p^r$$

A direct application of Theorem 1 can get the desired result. However, we notice that the form of polynomial $f(x) = (x + a)^r$ is kind of special, actually we can use a smarter way to solve this type of equation. In this paper, we exploit Lu et al's. [11] algorithm because of its better performance and lower complexity.

Theorem 2 (Lu et al. [11]). *For every $0 < \mu < \beta$, let N be a sufficiently large composite integer (of unknown factorization) with a divisor p^r ($p \geq N^{\beta}$ and r is a positive integer: $r \geq 1$). Let $f(x) \in \mathbb{Z}[x]$ be a univariate monic linear polynomial. Then we can find all the solutions x_0 of the equation $f(x) = 0 \bmod p$ with $|x_0| \leq N^{\gamma}$ if*

$$\gamma < r\beta^2 - \mu$$

The time complexity is $\mathcal{O}(\mu^{-7} \log^2 N)$.

For completeness, we give the whole proof here.

Proof. Consider the following univariate linear polynomial:

$$f_1(x) = a_0 + a_1 x \bmod p$$

where N is known to be a multiple of p^r for known r and unknown p. Here we assume that $a_1 = 1$, since otherwise we can multiply f_1 by $a_1^{-1} \bmod N$. Let $f(x) = a_1^{-1} f_1(x) \bmod N$.

We define a collection of polynomials as follows:

$$g_k(x) := f^k(x) N^{\max\{\lceil \frac{t-k}{r} \rceil, 0\}}$$

for $k = 0, \ldots, m$ and integer parameters t and m with $t = \tau m$ $(0 \le \tau < 1)$, which will be optimized later. Note that for all k, $g_k(y) \equiv 0 \bmod p^t$.

Let $X := N^{r\beta^2 - \mu} (= N^{\gamma})$ be the upper bound on the desired root y. We will show that this bound can be achieved for any chosen value of μ by ensuring that $m \ge m^* := \lceil \frac{\beta(2r+1-r\beta)}{\mu} \rceil - 1$.

We build a lattice $\mathcal{L}$ of dimension $d = m+1$ using the coefficient vectors of $g_k(xX)$ as basis vectors. We sort these polynomials according to the ascending order of g, i.e., $g_k < g_l$ if $k < l$.

From the triangular matrix of the lattice basis, we can compute the determinant as the product of the entries on the diagonal as $\det(\mathcal{L}) = X^s N^{s_N}$ where

$$\begin{aligned} s &= \sum_{k=0}^{m} k = \frac{m(m+1)}{2} \\ s_N &= \sum_{k=0}^{t-1} \lceil \frac{t-k}{r} \rceil = \sum_{k=0}^{t-1} \left(\frac{t-k}{r} + c_k \right) \\ &= \frac{\tau m(\tau m+1)}{2r} + \sum_{k=0}^{t-1} c_k . \end{aligned}$$

Here we rewrite $\lceil \frac{t-k}{r} \rceil$ as $\left(\frac{t-k}{r} + c_k \right)$ where $c_k \in [0, 1)$. To obtain a polynomial with short coefficients that contains all small roots over integer, we apply LLL-basis reduction algorithm to the lattice $\mathcal{L}$. Lemma 1 gives us an upper bound on the norm of the shortest vector in the LLL-reduced basis.

Lemma 1 (LLL [7]). *Let $\mathcal{L}$ be a lattice of dimension w. Within polynomial-time, LLL-algorithm outputs a set of reduced basis vectors v_i, $1 \leqslant i \leqslant w$ that satisfies*

$$||v_1|| \leqslant ||v_2|| \leqslant \cdots \leqslant ||v_i|| \leqslant 2^{\frac{w(w-1)}{4(w+1-i)}} \det(\mathcal{L})^{\frac{1}{w+1-i}}$$

If the bound is smaller than the bound given in Lemma 2 (below), we can obtain the desired polynomial.

Lemma 2 (Howgrave-Graham [9]). *Let $g(x_1, \cdots, x_k) \in \mathbb{Z}[x_1, \cdots, x_k]$ be an integer polynomial that consists of at most w monomials. Suppose that*

1. $g(y_1, \cdots, y_k) = 0 \bmod p^m$ for $|\, y_1 \,| \leqslant X_1, \cdots, |\, y_k \,| \leqslant X_k$ and
2. $||g(x_1X_1, \cdots, x_kX_k)|| < \frac{p^m}{\sqrt{w}}$

Then $g(y_1, \cdots, y_k) = 0$ holds over integers.

We require the following condition:

$$2^{\frac{d-1}{4}} \det(\mathcal{L})^{\frac{1}{d}} < \frac{N^{\beta\tau m}}{\sqrt{d}}$$

where $d = m+1$. We plug in the values for $\det(\mathcal{L})$ and d, and obtain

$$2^{\frac{m(m+1)}{4}} (m+1)^{\frac{m+1}{2}} X^{\frac{m(m+1)}{2}} < N^{\beta\tau m(m+1) - \frac{\tau m(\tau m+1)}{2r} - \sum_{k=0}^{t-1} c_k}$$

To obtain the asymptotic bound, we let m grow to infinity. Note that for sufficiently large N the powers of 2 and $m+1$ are negligible. Thus, we only consider the exponent of N. Then we have

$$X < N^{2\beta\tau - \frac{\tau(\tau m+1)}{r(m+1)} - \frac{2\sum_{k=0}^{t-1} c_k}{m(m+1)}}$$

Setting $\tau = r\beta$, and noting that $\sum_{k=0}^{t-1} c_k \leq t$, the exponent of N can be lower bounded by

$$r\beta^2 - \frac{\beta(1-r\beta)}{m+1} - \frac{2r\beta}{m+1}$$

We appropriate the negative term $\frac{*}{m+1}$ by $\frac{*}{m}$ and obtain

$$r\beta^2 - \frac{\beta(2r+1-r\beta)}{m}$$

Enduring that $m \geq m^*$ will then gurantee that X satisfies the required bound for the chosen value of μ.

The running time of our method is dominated by LLL-algorithm, which is polynomial in the dimension of the lattice and in the maximal bit-size of the entries. We have a bound for the lattice d

$$d = m+1 \geq \lceil \frac{\beta(2r+1-r\beta)}{\mu} \rceil$$

Since $r\beta < 1$, then we obtain $d = \mathcal{O}(\mu^{-1})$. The maximal bit-size of the entries is bounded by

$$\max\{\frac{t}{r}\log(N), dr\beta^2\log(N)\} = \max\{\beta d\log(N), dr\beta^2\log(N)\}$$

Since $r\beta < 1$ and $d = \mathcal{O}(\mu^{-1})$, the bit-size of the entries can be upper bounded by

$$\max\{\mathcal{O}(\beta\mu^{-1})\log(N), \mathcal{O}(\beta\mu^{-1})\log(N)\} = \mathcal{O}(\mu^{-1}\log(N))$$

Nguên and Stehlé [14] proposed a modified version of the LLL-algorithm called L^2-algorithm. The L^2-algorithm achieves the same approximation quality for a shortest vectors as the LLL-algorithm, but has an improved worst case running time analysis. Its running time is $\mathcal{O}(d^5(d+\log b_d)\log b_d)$, where $\log b_d$ is the maximal bit-size of an entry in lattice. Thus, we can obtain the running time of our algorithm

$$\mathcal{O}\left(\left(\frac{1}{\mu}\right)^5 \left(\frac{1}{\mu} + \frac{\log N}{\mu}\right)\frac{\log N}{\mu}\right)$$

Therefore, the running time of our algorithm is $\mathcal{O}(\mu^{-7}\log^2 N)$. Eventually, the vector output by LLL-algorithm gives a univariate polynomial $g(x)$ such that $g(y) = 0$, and one can find the root of $g(x)$ over the integers.

3 Our Main Result

In this section we give our main result.

Theorem 3. *Let N be an integer of unknown factorization and $e < N$ (suppose $\gamma = \log_N e$) be a prime integer and $\gcd(e, N) = 1$. First determine a positive integer r such that $\frac{1}{4r} < \gamma < \min\{\frac{1}{4(r-1)}, 1\}$. Let $\epsilon = \gamma - \frac{1}{4r}$, then we can decide whether exists secret factor p s.t. $e|p-1$ and $N \equiv 0 \bmod p^r$ in time $\mathcal{O}(\epsilon^{-8}\log^2 N)$.*

Before we provide a proof for Theorem 3, we would like to interpret its implications. Notice that Theorem 3 yields in the special case $r = 1$ the bound $\frac{1}{4} < \gamma < 1$ that corresponds to [10]'s result. Thus, our result can be viewed as a generalization of [10]'s result.

On the other hand, we would like to give an example to clarify our result. If $\frac{1}{8} < \gamma < \frac{1}{4}$, we can check whether exists secret factor p s.t. $e|p-1$ and $N \equiv 0 \bmod p^2$, which means that if some secret factor p hides e $(e|p-1)$ and p^2 divides modulus N $(N \equiv 0 \bmod p^2)$, our proposed algorithm can recover such factor p. Although our result does not guarantee that $\gcd(e, \phi(N)) = 1$ (since we can only check the factor p: $N \equiv 0 \bmod p^2$), we have $\gcd(e, \phi(N)) \neq 1$ if our algorithm outputs a factor p. In addition, let $N = \prod_{i=1}^{n} p_i^{z_i}$, for the case of $\min\{z_1, ..., z_n\} \geq 2$, we can identify whether $\gcd(e, \phi(N)) = 1$ or not, which improve [10]'s result to $\gamma > \frac{1}{8}$.

Proof. At first we get the value of integer r from the exponent e. Then we have

$$\phi(N) = \prod_{i=1}^{n} p_i^{z_i - 1}(p_i - 1)$$

Let us focus on the case $e|(p_i - 1)$ and $N \equiv 0 \bmod p_i^r$ for some i. Denote that $p := p_i$, there exists an $x_0 \in \mathbb{Z}$ s.t.

$$ex_0 + 1 = p$$

Next our goal is to find x_0, which is a small root of the polynomial equation $f(x) = ex + 1 \bmod p$ $(N \equiv 0 \bmod p^r)$.

In order to run Coppersmith's algorithm, we have to know the parameter β: the bitsize of unknown divisor. However, we do not know the exact value of β here. To overcome this problem, we give a lemma that can be used to check whether $e|p-1$ for some p $(N \equiv \bmod p^r)$ in a specific range. This following lemma can be regard as an extension of Lemma 5 of [10].

Lemma 3. *Let N be an integer of unknown factorization with divisor p^r: $p \geq N^{\beta}$ $(\beta \in (0, 1])$ and $r \geq 1$. Further, let $e = N^{\gamma}$ with $e|p-1$. Then there is an algorithm that, given (N, e, β, μ), outputs p in time $\mathcal{O}(\mu^{-7}\log^2 N)$ provided that*

$$p \leq N^{r\beta^2 + \gamma - \mu}$$

If this algorithm can not find a non-trivial factor of N, it outputs $\perp$.

Proof. We can easily get the result from Theorem 2. We have $e|p-1$, then we get the polynomial $f(x) = ex + 1$ has the root x_0 modulo p. By multiplying e^{-1} modulo N, we can get a monic polynomial. Since $p \leq N^{r\beta^2+\gamma-\mu}$, we have

$$x_0 = \frac{p-1}{e} < \frac{N^{r\beta^2+\gamma-\mu}}{N^\gamma} = N^{r\beta^2-\mu}$$

Therefore, we can recover x_0 by Theorem 2 in time $\mathcal{O}(\mu^{-7}\log^2 N)$. For every candidate of x_0, we check whether $\gcd(ex_0+1, N)$ gives us the divisor p. Since the number of the candidate is bounded by $\mathcal{O}(\mu^{-1}r)$, this can be done in $\mathcal{O}(\mu^{-1}r\log^2 N)$, which can be negligible compared to the running time of Coppersmith's algorithm.

Using Lemma 3, we can check whether $e|p-1$ for some p ($N \equiv \bmod p^r$) in the range $[N^\beta, N^{r\beta^2-\mu+\gamma}]$. However, it is not enough, our target range is $p \in [e, N^{\frac{1}{r}}]$, much larger than the search range of Lemma 3. Next we apply [10]'s idea to solve the above problem.

At first, we set $r\beta^2 - \mu + \gamma = \frac{1}{r}$, which implies $\beta = \frac{\sqrt{1-r(\gamma-\mu)}}{r}$. Then we can check p in the interval $[N^{\frac{\sqrt{1-r(\gamma-\mu)}}{r}}, N^{\frac{1}{r}}]$ using Lemma 3. If we can not find such p, we turn to the range $[e, N^{\frac{\sqrt{1-r(\gamma-\mu)}}{r}}]$, then we use Lemma 3 again, and set $r\beta^2 - \mu + \gamma = \frac{\sqrt{1-r(\gamma-\mu)}}{r}$, which defines a new lower bound β. We then repeat this process.

To summary, we cover the target range by a sequence of intervals $[N^{\beta_1}, N^{\beta_0}], ..., [N^{\beta_n}, N^{\beta_{n-1}}]$ where the β_i are defined by the recurrence relation

$$\beta_{i+1} = \max\{\sqrt{\frac{\beta_i - (\gamma-\mu)}{r}}, \gamma\} \quad \text{with} \quad \beta_0 = \frac{1}{r}.$$

Here we suppose that

$$\frac{1}{4r} < \gamma - \mu < \gamma < \min\{\frac{1}{4(r-1)}, 1\}$$

We have to prove that the number of invocations of Lemma 3 is polynomial. At first, we show by induction that the sequence of the β_i is monotone decreasing.

It is obvious that $\beta_1 < \beta_0$ since $\gamma < \frac{1}{r} = \beta_0$ and $\sqrt{\frac{\beta_0-(\gamma-\mu)}{r}} < \sqrt{\frac{\beta_0}{r}} = \beta_0$. We now show that if $\beta_i \leq \beta_{i-1}$ for all $i \leq m$, we have $\beta_{m+1} \leq \beta_m$.

Since $\beta_m \leq \beta_{m-1}$, we have

$$\frac{\beta_m - (\gamma-\mu)}{r} \leq \frac{\beta_{m-1} - (\gamma-\mu)}{r}$$

which implies

$$\sqrt{\frac{\beta_m - (\gamma-\mu)}{r}} \leq \sqrt{\frac{\beta_{m-1} - (\gamma-\mu)}{r}}$$

That means

$$\max\{\sqrt{\frac{\beta_m - (\gamma - \mu)}{r}}, \gamma\} \leq \max\{\sqrt{\frac{\beta_{m-1} - (\gamma - \mu)}{r}}, \gamma\}$$

Thus $\beta_{m+1} \leq \beta_m$.

Since the sequence of $\{\beta_i\}$ is monotone decreasing and bounded below by γ, it converges. Now we investigate the length of interval $[\beta_i, \beta_{i-1}]$. Define a function $\Delta(\beta_{i-1}) = \beta_{i-1} - \beta_i \geq 0$, which is the length of the i^{th} interval. We have

$$\Delta(\beta_{i-1}) = \beta_{i-1} - \beta_i = \beta_{i-1} - \sqrt{\frac{\beta_{i-1} - (\gamma - \mu)}{r}}$$

We calculate the first two derivations of $\Delta(\beta)$ as follows

$$\Delta^{'}(\beta) = 1 - \frac{1}{2\sqrt{r}}(\beta - (\gamma - \mu))^{-\frac{1}{2}}$$

and

$$\Delta^{''}(\beta) = \frac{1}{4\sqrt{r}}(\beta - (\gamma - \mu))^{-\frac{3}{2}} > 0$$

It is clear that $\Delta(\beta)$ achieves its minimum at the point $\beta^{(0)} = \frac{1}{4r} + \gamma - \mu$ (when $\Delta^{'}(\beta) = 0$). And the length of interval $\Delta(\beta)$ is of size at least

$$\Delta(\beta^{(0)}) = \gamma - \mu - \frac{1}{4r}$$

Let

$$k := \lceil \frac{\frac{1}{r} - \gamma}{\gamma - \mu - \frac{1}{4r}} \rceil + 1$$

That means after the number of k steps, the sequence β_i stabilize at the point $\beta_k = \gamma$.

Therefore, if we run the algorithm of Lemma 3 at most k times, we can test the entire range $[e, N^{\frac{1}{r}}]$.

Before we give the running time of our algorithm, we would like to discuss the choice of the parameter μ. Since $\epsilon := \gamma - \frac{1}{4r}$, we have the condition $\frac{1}{4r} < \gamma - \mu$, thus $\mu < \gamma - \frac{1}{4r} = \epsilon$, we simply choose $\mu = \frac{1}{c}\epsilon$ where c is a positive integer. In practice, if we choose larger c, which means smaller value of μ, then the value of k is smaller, and the same time, Lemma 3 may take more running time; if c is smaller, by contrast, means larger value of k and less running time of Lemma 3. However, note that the running time of our algorithm is mainly decided by the running time of Lemma 3, thus we would like to choose $c = 2$ in practice.

At last we give the total running time of our algorithm, we have to run the algorithm of Lemma 3 at most k times, which can be bounded as

$$k := \lceil \frac{\frac{1}{r} - \gamma}{\gamma - \mu - \frac{1}{4r}} \rceil + 1 \leq \lceil \frac{\frac{1}{r}}{(c-1)\mu} \rceil + 1 = \mathcal{O}(\mu^{-1}) = \mathcal{O}(\epsilon^{-1})$$

Table 1. Example: concrete values for $r = 2$

$r = 2$	β_0	β_1	β_2	β_3	β_4	β_5	β_6	β_7
$\gamma = 0.187500$ $\mu = 0.031250$	0.500000	0.414578	0.359394	0.314396	0.281199	0.249949	0.216447	0.187500
$\gamma = 0.187500$ $\mu = 0.015625$	0.500000	0.405046	0.341446	0.291179	0.244238	0.190214	0.187500	
$\gamma = 0.200000$ $\mu = 0.020000$	0.500000	0.400000	0.331662	0.275374	0.218374	0.200000		

and each iteration takes time $\mathcal{O}(\mu^{-7}\log^2 N)$, finally we obtain the total runnning time of our algorithm

$$\mathcal{O}(\mu^{-8}\log^2 N) = \mathcal{O}(\epsilon^{-8}\log^2 N)$$

In Table 1, we give three examples to show the concrete values of $\{\beta_i\}$. Note that if μ is smaller, we can take fewer steps to search the whole target range.

4 Conclusion

In this paper, we extend Kakvi et al.'s work by considering the case with generalized moduli $N = \prod_{i=1}^{n} p_i^{z_i}$. Surprisingly, when $\min\{z_1, \ldots, z_n\} \geq 2$, we show that it can be efficiently decided whether the RSA function defines a permutation over $\mathbb{Z}_N^*$ or not even for the prime $e < N^{\frac{1}{4}}$. Our result can be viewed as an extension of Kakvi et al.'s result.

Acknowledgments. The authors thank anonymous reviewers for their valuable comments. This work was supported in part by National Natural Science Foundation of China (Nos. 61632020, 61472416, 61772520,61702505, 61732021), Key Research Project of Zhejiang Province (No. 2017C01062), Fundamental theory and cutting edge technology Research Program of Institute of Information Engineering, CAS (No. Y7Z0321102, Y7Z0341103), National Cryptography Development Fund (MMJJ20170115), JST CREST (No. JPMJCR14D6), and JSPS KAKENHI (No.16H02780).

References

1. Bellare, M., Namprempre, C., Neven, G.: Unrestricted aggregate signatures. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 411–422. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73420-8_37
2. Bellare, M., Yung, M.: Certifying cryptographic tools: the case of trapdoor permutations. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 442–460. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-48071-4_31
3. Bellare, M., Yung, M.: Certifying permutations: noninteractive zero-knowledge based on any trapdoor permutation. J. Cryptol. **9**(3), 149–166 (1996)

4. Cachin, C., Micali, S., Stadler, M.: Computationally private information retrieval with polylogarithmic communication. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 402–414. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_28
5. Coppersmith, D.: Small solutions to polynomial equations, and low exponent RSA vulnerabilities. J. Cryptol. **10**(4), 233–260 (1997)
6. Dwork, C., Naor, M.: Zaps and their applications. In: 41st Annual Symposium on Foundations of Computer Science, FOCS 2000, Redondo Beach, California, USA, 12–14 November 2000, pp. 283–293 (2000)
7. Giuliani, K.: Factoring polynomials with rational coefficients. Math. Ann. **216**(4), 515–534 (1982)
8. Goldreich, O.: Basing non-interactive zero-knowledge on (enhanced) trapdoor permutations: the state of the art. In: Goldreich, O. (ed.) Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation. LNCS, vol. 6650, pp. 406–421. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22670-0_28
9. Howgrave-Graham, N.: Finding small roots of univariate modular equations revisited. In: Darnell, M. (ed.) Cryptography and Coding 1997. LNCS, vol. 1355, pp. 131–142. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0024458
10. Kakvi, S.A., Kiltz, E., May, A.: Certifying RSA. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 404–414. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34961-4_25
11. Lu, Y., Zhang, R., Peng, L., Lin, D.: Solving linear equations modulo unknown divisors: revisited. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015. LNCS, vol. 9452, pp. 189–213. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48797-6_9
12. Lysyanskaya, A., Micali, S., Reyzin, L., Shacham, H.: Sequential aggregate signatures from trapdoor permutations. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 74–90. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24676-3_5
13. May, A.: Using LLL-reduction for solving RSA and factorization problems. In: Nguyen, P., Vallée, B. (eds.) The LLL Algorithm - Survey and Applications, pp. 315–348. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-02295-1_10
14. Nguên, P.Q., Stehlé, D.: Floating-point LLL revisited. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 215–233. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_13
15. Schridde, C., Freisleben, B.: On the validity of the Φ-hiding assumption in cryptographic protocols. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 344–354. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89255-7_21
16. Takagi, T.: Fast RSA-type cryptosystem modulo $p^{\mathrm{k}}q$. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 318–326. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0055738