FACTORING IS EQUIVALENT TO GENERIC RSA

Divesh Aggarwal and Ueli Maurer

Department of Computer Science,
ETH Zurich, 8092 Zurich, Switzerland
{divesha,maurer}@inf.ethz.ch

Abstract. We show that a generic ring algorithm for breaking RSA in $\mathbb{Z}_N$ can be converted into an algorithm for factoring the corresponding RSA-modulus N. Our results imply that any attempt at breaking RSA without factoring N will be non-generic and hence will have to manipulate the particular bit-representation of the input in $\mathbb{Z}_N$. This provides new evidence that breaking RSA may be equivalent to factoring the modulus.

1. Introduction

1.1. **RSA vs Factoring.** The security of the well-known RSA public key encryption and signature scheme [16] relies on certain number theoretic assumptions on the ring $\mathbb{Z}_N$, where N is a random variable chosen according to a certain probability distribution over the product of two primes p and q, for example an element chosen uniformly at random from the set of products of two random k-bit primes satisfying certain conditions. The following are some of the assumptions that have been studied in this context.

RSA Assumption: Given N, an integer $e > 1$ that is relatively prime to $\phi(N)$[1], and an element a chosen uniformly at random from $\mathbb{Z}_N^*$, no probabilistic polynomial time adversary can compute $x \in \mathbb{Z}_N^*$ such that $x^e = a (\text{mod } N)$ with non-negligible probability.[2]

Low-exponent RSA (LE-RSA) Assumption: This is the RSA assumption under the additional constraint that e is bounded by a constant.

'Hardness of Factoring' Assumption: Given N, no probabilistic polynomial time adversary can find a non-trivial factor of N with non-negligible probability.

It is easy to see that if the RSA assumption holds then factoring is hard. However it is a long-standing open problem whether the converse is true. It is therefore interesting to investigate reasonable restricted models of computation and prove that in such a model factoring is equivalent to the RSA problem. In a restricted model one assumes that only certain kinds of operations are allowed. Nechaev [14] and Shoup [17] introduced

[1] e can take any integer value, and so can in principle be much larger than N

[2] A function $f(k)$ is considered a negligible function of k, if for all $c > 0$ and sufficiently large k, $|f(k)| < \frac{1}{k^c}$. In this paper k is taken to be $log(N)$ i.e. the size of the input.

the concept of generic algorithms which are algorithms that do not exploit any property of the representation of the elements. They proved lower bounds on the complexity of computing discrete logarithms in the context of generic algorithms. Computing Discrete Logarithms is one of the two problems (the other is Integer Factorization) whose hardness assumption most existing public key cryptosystems rely on. Maurer [12] provided a simpler and more general model for modeling and analyzing representation-independent algorithms.

1.2. **Generic Model of Computation.** We give here a brief description of the model of [12], sufficient for our purpose. The model is characterized by a black-box $\mathbf{B}$ which can store values from a certain set S in internal state variables $x_0, x_1, x_2, \cdots$.

The initial state consists of the values of $[x_0, \cdots, x_\ell]$, which are set according to some probability distribution (e.g. the uniform distribution).

The black box $\mathbf{B}$ allows two types of operations: computation operations on internal state variables and queries about the internal state variables. We give a more formal description of these operations.

Computation operations. For a set Π of operations on S of some arities, a computation operation consists of selecting an operation $f \in \Pi$ (say t-ary) as well as the indices $i_1, \cdots, i_{t+1}$ of $t+1$ state variables. $\mathbf{B}$ computes $f(x_{i_1}, \cdots, x_{i_t})$ and stores the result in $x_{i_{t+1}}$.

Relation Queries. For a set Σ of relations (of some arities) on S, a query consists of selecting a relation $\rho \in \Sigma$ (say t-ary) as well as the indices $i_1, \cdots, i_t$ of t state variables. The query is replied by $\rho(x_{i_1}, \cdots, x_{i_t})$.

The complexity of an algorithm for solving any problem in this model can be measured by the number of interactions it can perform with the black-box $\mathbf{B}$.

For this paper, the set S is $\mathbb{Z}_N$. A generic ring algorithm is an algorithm that is just allowed to perform the ring operations, i.e., addition and multiplication as well as the inverse ring operations (negatives and multiplicative inverses), and can test for equality. Many results in the literature are restricted in that they exclude the inverse operations, but since these operations are easy to perform in $\mathbb{Z}_N$, they should be included as otherwise the results are only of limited interest.

In this model of [12], for example, generic ring algorithms on $\mathbb{Z}_N^*$ correspond to $\Pi = \{+, -, \cdot, /\}$ and the set of relations being $\Sigma = \{=\}$. A *straight-line program (SLP)* on $\mathbb{Z}_N$ corresponds to the case where Σ is the empty set, i.e., no equality tests are possible.

In this paper, we show that under the assumption that factoring N is hard, given any e relatively prime to $\phi(N)$ and an element a chosen uniformly at random from $\mathbb{Z}_N^*$, no probabilistic polynomial-time generic ring algorithm can compute, with non-negligible probability, an $x \in \mathbb{Z}_N^*$ such that $x^e = a \pmod N$.

1.3. **Related Work and Contributions of this Paper.** Boneh and Venkatesan [2] showed that any straight line program that efficiently factors N given access to an oracle solving the LE-RSA problem (RSA problem when the public exponent e is small) can

be converted into a real polynomial-time algorithm for factoring N. This means that if factoring is hard, then there exists no straight-line reduction from factoring to LE-RSA.

Brown [3] showed that if factoring is hard then the LE-RSA problem is intractable for straight-line programs *without* multiplicative inverses, i.e., $\Pi = \{+, -, \cdot\}$. More precisely, he proves that an efficient SLP for breaking LE-RSA can always be transformed into an efficient factoring algorithm.

Leander and Rupp [10] generalized the result of [3] to generic ring algorithms which, as explained above, can test the equality of elements. Again, multiplicative inverses are excluded ($\Pi = \{+, -, \cdot\}$).

Another theoretical result about the hardness of the RSA problem is due to Damgard and Koprowski [7]. They studied the problem of root extraction in finite groups of unknown order and proved that the RSA problem is intractable with respect to generic group algorithms. This corresponds to excluding addition and multiplicative inversion from the set of operations ($\Pi = \{\cdot\}$).

Our results generalize the previous results in many ways. (Actually, Theorem 9 appears to be the most general statement about the equivalence of factoring and breaking RSA in a generic model.)

- First, compared to [3, 10] we consider the full-fledged RSA problem (not only LE-RSA) with exponent e of arbitrary size, even with bit-size much larger than that of N.
- Second, compared to [7, 3, 10] we consider the unresticted set of ring operations, including multiplicative inverses. This generalization is important since there are problems that are easy to solve in our generic ring model but are provably hard to solve using the model without multiplicative inversion. Actually, as has been pointed out by the author of [3] himself, computing the multiplicative inverse of a random element in $\mathbb{Z}_N$ is hard if $\Pi = \{+, -, \cdot\}$.
- Third, we allow for randomized generic algorithms.

1.4. **Organization of the Paper.** The rest of the paper is structured as follows: In Section 2, we introduce basic definitions and notations and show a few preliminary results. In Section 3, we show that under the assumption that factoring is hard, no adversary using a generic ring algorithm can solve the RSA problem. Section 4 provides some conclusions and lists some open problems.

2. Preliminaries

2.1. **Straight-Line Programs.** Straight-line programs are algorithms that correspond to $\Pi = \{+, -, \cdot, /\}$ and $\Sigma = \{\}$.
More concretely:

Definition 1. A *straight-line program(SLP)* of length L on $\mathbb{Z}_N$ and on input $\{y_1, \cdots, y_\ell\} \subset \mathbb{Z}_N$ is a sequence of (random) tuples $(a_k, b_k, \circ_k)$ for $\ell + 1 \leq k \leq L$, where, for all k, $(a_k, b_k, \circ_k)$ is chosen according to some distribution conditioned on $x_1 \cdots, x_{k-1}$ and x_i (i^{th} internal state variable) is given by $x_0 = 1$, $x_m = y_m$ for $1 \leq m \leq \ell$ and $x_m = x_{a_m} \circ_m x_{b_m}$ for $\ell + 1 \leq m \leq L$. The output of the SLP is x_L.

Note that our definition of a SLP is more general than the one used in [2, 3] because we include the inverse ring operations and allow randomization.

For any SLP P of length L, each x_k for $0 \leq k \leq L$ is of the form

$$x_k = \frac{P_k(y_1, \cdots, y_\ell)}{Q_k(y_1, \cdots, y_\ell)},$$

where P_k and Q_k are polynomials. The following lemma states that the degree of P_k and Q_k is at most exponential in k.

Lemma 2. *P_k and Q_k are polynomials of degree at most 2^k in each of the y_i's.*

Proof. We prove this by induction on k.
The result is trivially true for $k = 0$. We assume P_r and Q_r are polynomials of degree at most 2^r in each of the y_i's for all $r < k$.
$\frac{P_k}{Q_k} = \frac{P_{a_k}}{Q_{a_k}} \circ_k \frac{P_{b_k}}{Q_{b_k}}$ for some $a_k, b_k, \circ_k$.

- Case (i): $\circ_k \in \{+, -\}$.
  In this case $deg(P_k) \leq max(deg(P_{a_k}) + deg(Q_{b_k}), deg(Q_{a_k}) + deg(P_{b_k})) \leq 2^{a_k} + 2^{b_k} \leq 2^{k-1} + 2^{k-1} = 2^k$ and $deg(Q_k) \leq deg(Q_{a_k}) + deg(Q_{b_k}) \leq 2^{a_k} + 2^{b_k} \leq 2^{k-1} + 2^{k-1} = 2^k$.
- Case (ii): $\circ_k \in \{\cdot, /\}$.
  Here, $max(deg(P_k), deg(Q_k)) \leq max(deg(P_{a_k}), deg(Q_{a_k})) + max(deg(P_{b_k}), deg(Q_{b_k})) \leq 2^{a_k} + 2^{b_k} \leq 2^{k-1} + 2^{k-1} = 2^k$.

□

Next, we show that any SLP can be converted into a SLP that does not require the multiplicative inverse operation, i.e., for which $\Sigma = \{+, -, \cdot\}$, without increasing the complexity by more than a constant factor.

Lemma 3. *There exists a SLP of length $4L$ that uses only the operations $\{+, -, \cdot\}$ and contains $P_k(y_1 \cdots, y_\ell)$ and $Q_k(y_1 \cdots, y_\ell)$ for $1 \leq k \leq L$.*

Proof. We prove this by induction on L.
The result is trivial for $L = 0$. We suppose it is true for $L = L'$. Therefore there exists a SLP of length $len \leq 4L'$ that uses only the operations $\{+, -, \cdot\}$ and contains P_k and Q_k for $1 \leq k \leq L'$. Let this program compute the values x'_i for $1 \leq i \leq len$. Let $r = L' + 1$. Now consider the following cases:

- Case (i): $\circ_r \in \{+, -\}$.
  Let $x'_{len+1} = P_{a_r} \cdot Q_{b_r}$, $x'_{len+2} = P_{b_r} \cdot Q_{a_r}$, $x'_{len+3} = x'_{len+1} \circ_r x'_{len+2} = P_r$ and $x'_{len+4} = Q_{a_r} \cdot Q_{b_r} = Q_r$.
- Case (ii): $\circ_r \in \{\cdot\}$.
  Let $x'_{len+1} = P_{a_r} \cdot P_{b_r} = P_r$ and $x'_{len+2} = Q_{a_r} \cdot Q_{b_r} = Q_r$.
- Case (iii): $\circ_r \in \{/\}$.
  Let $x'_{len+1} = P_{a_r} \cdot Q_{b_r} = P_r$ and $x'_{len+2} = Q_{a_r} \cdot P_{b_r} = Q_r$.

Therefore, in each of the cases, we get a SLP of length at most $len + 4 \leq 4(L' + 1)$.

□

2.2. **Generic Ring Algorithms.** A generic ring algorithm is allowed to test for equality of elements in the black-box, in addition to performing the arithmetic operations. It corresponds to $\Pi = \{+, -, \cdot, /\}$ and $\Sigma = \{=\}$.
More concretely:

Definition 4. A *generic ring algorithm* of L steps in $\mathbb{Z}_N$ is an algorithm that takes as input some elements of the ring $\mathbb{Z}_N$, perform one operation in each step and outputs an element in $\mathbb{Z}_N$. Each operation performed is either a computation operation from the set $\Pi = \{+, -, \cdot, /\}$ on two previously computed values, or it is a query whether two of the previously computed values are equal.

2.3. **The Generic RSA Problem.** As mentioned earlier, in this paper we restrict our attention to the case where the adversary is only allowed to use a generic ring algorithm to solve the RSA problem. We refer to the RSA assumption in this case as the Generic RSA assumption.

Generic RSA Assumption: Given N, an integer $e > 1$ that is relatively prime to $\phi(N)$, and an element a chosen uniformly at random from $\mathbb{Z}_N^*$, no probabilistic polynomial time generic ring algorithm can compute $x \in \mathbb{Z}_N^*$ such that $x^e = a (\text{mod } N)$ with non-negligible probability.

We begin by proving the following lemma that shows that under the 'factoring is hard' assumption, if a problem is hard to solve using a straight line program, then it is also hard to solve using a generic ring algorithm. This generalizes the argument that Leander et al [10] used for generalizing the result of [3] from SLPs to generic ring algorithms (without division operation).

Lemma 5. *For any L-step generic ring algorithm $\mathcal{A}$ on $\mathbb{Z}_N$, for all $0 < c < 1$, either there exists an L-step SLP which, on random input from $\mathbb{Z}_N^{*\ell}$, gives the same output as $\mathcal{A}$ with probability at least $1 - c$, or there exists an algorithm that factors N and has expected running time $O(\frac{L^4 \cdot log^2(N)}{c^2})$.*

Proof. Let $N = pq$ and let $\{y_1, \cdots, y_\ell\} \subset \mathbb{Z}_N$ be randomly chosen elements that are given as input to $\mathcal{A}$. Since $\mathcal{A}$ is an L step algorithm the total number of values it computes in the ring can be at most L. Let $\{x_k | 1 \leq k \leq L'\}$(where $L' \leq L$) be the values computed by $\mathcal{A}$. Consider the following two cases.

CASE 1: $\exists r, s$ such that $\frac{c}{L} \leq Pr(x_r = x_s) \leq 1 - \frac{c}{L}$.

In this case we give an algorithm that factors N. The algorithm proceeds as follows:

Repeat:

(1) Generate random elements $y_1, \cdots, y_\ell \in \mathbb{Z}_N^*$.
(2) Run $\mathcal{A}$ to get x_k for $1 \leq k \leq L'$.
(3) For all $t, u \leq L'$, compute $g = gcd(x_t - x_u, N)$. If $g \notin \{1, N\}$, return g.

The algorithm is correct because it continues till we get a non-trivial factor of N. We now give a lower bound on the success probability of one run of the loop.

Let $Pr(x_r = x_s \bmod p)$ and $Pr(x_r = x_s \bmod q)$ be γ_p and γ_q respectively. It is easy to see that $x_r - x_s$ is a non-zero non-invertible element of $\mathbb{Z}_N$ if $x_r - x_s = 0$ modulo one of p and q and $x_r - x_s \neq 0$ modulo the other. Therefore the probability that one run of the algorithm is successful is at least $\gamma_p(1-\gamma_q) + \gamma_p(1-\gamma_q)$.
Clearly, $Pr(x_r = x_s \bmod N) = \gamma_p \cdot \gamma_q$, which implies $\frac{c}{L} \leq \gamma_p \cdot \gamma_q \leq 1 - \frac{c}{L}$.
$\gamma_p \cdot \gamma_q \geq \frac{c}{L} \Rightarrow \gamma_p \geq \frac{c}{L}$ and $\gamma_q \geq \frac{c}{L}$ (because γ_p and γ_q are at most 1).
Also, $\gamma_p \cdot \gamma_q \leq 1 - \frac{c}{L} \Rightarrow$ either $\gamma_p \leq 1 - \frac{c}{2L}$ or $\gamma_q \leq 1 - \frac{c}{2L}$ (because if $\gamma_p > 1 - \frac{c}{2L}$ and $\gamma_q > 1 - \frac{c}{2L}$, then $\gamma_p \cdot \gamma_q > (1 - \frac{c}{2L})^2 > 1 - \frac{c}{L}$).

So the success probability of one run of the loop is at least $\frac{c}{L} \cdot (1 - (1 - \frac{c}{2L})) = \frac{c^2}{2L^2}$. Hence the expected number of repetitions till we get a factor of N is $O(L^2/c^2)$. Each gcd computation can be performed in $O(log^2(N))$ time. Hence the expected time complexity of the algorithm is $O(\frac{L^2 \cdot L'^2 \cdot log^2(N)}{c^2}) = O(\frac{L^4 \cdot log^2(N)}{c^2})$.

CASE 2: $\forall r, s$ such that $0 \leq r, s \leq L'$, either $Pr(x_r = x_s) < \frac{c}{L}$ or $Pr(x_r = x_s) > 1 - \frac{c}{L}$.

In this case, we give an L' step SLP that computes the values $x_0, \cdots, x_{L'}$ and hence gives the same output as $\mathcal{A}$ with probability at least c. Let the SLP be given by x'_k for $0 \leq k \leq L'$. The other parameters, as in the definition of SLP, are given by $a_k, b_k, \circ_k$ for $0 \leq k \leq L'$. What we want essentially is that the SLP should compute the same elements of $\mathbb{Z}_N^*$ as $\mathcal{A}$ without using any query of the equality relation.The SLP proceeds as follows. For each computation operation performed by $\mathcal{A}$, $x_k = x_{a_k} \circ_k x_{b_k}$ the SLP performs the computation operation $x'_k = x'_{a_k} \circ_k x'_{b_k}$. For each equality test in $\mathcal{A}$ of some two elements, say x_i and x_j, if $Pr(x_i = x_j) < \frac{c}{L}$, then the SLP assumes $x_i \neq x_j$, else (if $Pr(x_r = x_s) > 1 - \frac{c}{L}$) the SLP assumes $x_i = x_j$.

Now to compute the probability that $x_k = x'_k$ for $0 \leq k \leq L'$. Clearly this happens if all the equality test results that the SLP assumes are correct. The assumption on the result of each equality test is incorrect with probability at most c/L. There are a total of at most L equality tests, and so the SLP differs from $\mathcal{A}$ in at least one computed value with probability at most c.

□

Therefore, if factoring is hard, then any problem that can be solved using a generic algorithm can also be solved using a SLP. Since we assume that factoring is hard for all results in this paper, for the rest of the paper, we can restrict our attention to SLPs.

3. Generic RSA is equivalent to factoring

In this section, we prove that under the assumption that factoring N is hard, the Generic RSA assumption holds. The arguments used to prove the results of this section are motivated by those in [3].

Lemma 6. *Let p be a prime. A random degree d monic polynomial $f(x) \in \mathbb{Z}_p[x]$ is irreducible in $\mathbb{Z}_p[x]$ with probability at least $\frac{1}{2d}$ and has a root in $\mathbb{Z}_p$ with probability at least $1/2$.*

Proof. From the distribution theorem of monic polynomials (see, e.g., [11]), it follows that the number of monic irreducible polynomials of degree d over F_p is at least $\frac{p^d}{2d}$. Therefore $f(x)$ will be an irreducible polynomial over $\mathbb{Z}_p$ with probability at least $\frac{1}{2d}$.

The number of monic polynomials over $\mathbb{Z}_p$ with at least one root is:

$$\sum_{l=1}^{d}(-1)^{l-1}\binom{p}{l}p^{d-l}. \tag{1}$$

This can be seen by applying the principle of inclusion and exclusion. The terms in this summation are in decreasing order of their absolute value. So, taking the first two terms, this sum is greater than $\binom{p}{1}p^{d-1} - \binom{p}{2}p^{d-2}$ which is greater than $\frac{p^d}{2}$. Hence the probability that $f(x)$ has a root in $\mathbb{Z}_p$ is at least $1/2$.

□

For any polynomials $A(x), B(x) \in \mathbb{Z}_N[x]$, let $gcd_p(A(x), B(x))$ and $gcd_q(A(x), B(x))$ be the gcd of the polynomials modulo p and q respectively. We state the following observation that is easy to see.

Observation 7. *Let $h_1(x), h_2(x) \in \mathbb{Z}_N[x]$. Then:*

- *If Euclid's algorithm, when run on $h_1(x)$ and $h_2(x)$, fails[3], some step of the algorithm yields a non-trivial non-invertible element of $\mathbb{Z}_N$. We denote this element as $H(h_1(x), h_2(x))$.*
- *If $deg(gcd_p(h_1(x), h_2(x))) \neq deg(gcd_p(h_1(x), h_2(x)))$, then the Euclid's algorithm, when run on $(h_1(x), h_2(x))$, fails.*

Lemma 8. *Let α be an element chosen uniformly at random from $\mathbb{Z}_N$. Let $\mathcal{A}$ be an L step SLP that takes as input α and a positive integer $e > 1$ such that $(e, \phi(N)) = 1$ and outputs an element $\beta \in \mathbb{Z}_N$ such that $Pr(\beta^e = \alpha) \geq \mu$. Then there exists an algorithm that factors N and whose expected running time is $O(\frac{L^4 + log^4(e) + log^2(N)}{\mu})$.*

Proof. Since the only possible operations in a SLP are the arithmetic operations, the only functions of the input that can be computed by $\mathcal{A}$ are rational functions in α. Let $\beta = \frac{f(\alpha)}{g(\alpha)}$, where $f(x)$ and $g(x)$ are polynomials in $\mathbb{Z}_N[x]$. Let $P(x) = f(x)^e - x \cdot g(x)^e$. Then, $Pr(P(\alpha) = 0 \bmod n) \geq \mu$.

The factoring algorithm proceeds as follows:
Repeat until the algorithm returns

(1) Choose a monic polynomial $h(x)$ uniformly at random from all monic polynomials of degree d $(= log(e) + L)$ in $\mathbb{Z}_N[x]$.
(2) Compute $h'(x)$, the derivative of $h(x)$ in $\mathbb{Z}_N[x]$.
(3) Then choose a random element $r(x) \in \mathbb{Z}_N[x]/h(x)$.
(4) Compute $z(x) = f(r(x))^e - g(r(x))^e \cdot r(x)$ in $\mathbb{Z}_N[x]/h(x)$.
(5) Run Euclid's algorithm in $\mathbb{Z}_N[x]$ on the pairs $h(x)$ and $z(x)$. If this fails return $gcd(N, H(h(x), z(x)))$.

[3]Euclid's Algorithm could fail since $\mathbb{Z}_N[x]$ is not a Euclidean domain.

(6) Run Euclid's algorithm in $\mathbb{Z}_n[x]$ on the pairs $h(x)$ and $h'(x)$. If this fails return $gcd(N, H(h(x), h'(x)))$.

By Observation 7, if the algorithm terminates, it yields a factor of N.

Now we compute the success probability of one loop of the algorithm. By Lemma 6, the probability that $h(x)$ is irreducible modulo q and has a root modulo p is at least $\frac{1}{2d} \cdot \frac{1}{2} = \frac{1}{4d}$. We assume this for the rest of the proof.

Let the root of $h(x)$ modulo p be s. Therefore $(x - s) \mid h(x)$ in $\mathbb{Z}_p[x]$.

- Case 1: $(x - s)^2 \mid h(x)$ in $\mathbb{Z}_p[x]$.
  This implies $(x - s) \mid gcd_p(h(x), h'(x))$.
  However, since $h(x)$ is irreducible in $\mathbb{Z}_q[x]$, $gcd_q(h(x), h'(x)) \in \mathbb{Z}_q$.
  Therefore $gcd_p(h(x), h'(x))$ and $gcd_q(h(x), h'(x))$ have unequal degree, which implies, by Observation 7, that the Euclid's algorithm on $h(x)$ and $h'(x)$ fails and hence step 6 yields a factor of x.
- Case 2: $(x - s)^2 \nmid h(x)$ in $\mathbb{Z}_p[x]$.
  Let $h(x) = h_1(x) \cdot (x - s) \bmod p$. Then:

$$\mathbb{Z}_N[x]/h(x) \cong \mathbb{Z}_p[x]/h(x) \times \mathbb{Z}_q[x]/h(x) \cong \mathbb{Z}_p[x]/(x - s) \times \mathbb{Z}_p[x]/h_1(x) \times \mathbb{F}_{q^d}. \quad (2)$$

  because $\mathbb{Z}_q[x]/h(x) \cong \mathbb{F}_{q^d}$ as $h(x)$ is irreducible in $\mathbb{Z}_q[x]$ by our assumption.

  Under this isomorphism, let $r(x)$ and $z(x)$ map to the tuples $(r(s)(mod\ p), u(x), r_q(x))$ and $(z(s)(mod\ p), v(x), z_q(x))$ respectively, where $r_q(x)$ and $z_q(x)$ are the reductions of $r(x)$ and $z(x)$ modulo q.
  Since $r(x)$ is uniformly random in $\mathbb{Z}_N[x]/h(x)$, $r(s)$ is uniformly random in $\mathbb{Z}_p[x]/(x - s) \cong \mathbb{Z}_p$.

  This implies $Pr(z(s) = 0 \bmod p) = Pr(f(r(s))^e - g(r(s))^e \cdot r(s) = 0 \bmod p) \geq Pr(f(r(s))^e - g(r(s))^e \cdot r(s) = 0 \bmod n) \geq \mu$.
  Therefore, with probability at least μ, $(x - s)$ divides $z(x)$ in $\mathbb{Z}_p[x]$, which implies $Pr((x - s)$ divides $gcd_p(z(x), h(x))) \geq \mu$.

  Since $r(x)$ is uniformly random in $\mathbb{Z}_N[x]/h(x)$, $r_q(x)$ will be uniformly random in $\mathbb{Z}_q[x]/h(x) \cong \mathbb{F}_{q^d}$.
  This implies $Pr(z_q(x) = 0) = Pr(P(r_q(x)) = 0) \leq \frac{deg(P)}{q^d}$ (because a polynomial over a finite field can have at most as many roots as the degree of the polynomial).
  So, by Lemma 2, $Pr(z_q(x) \neq 0$ in $\mathbb{Z}_q[x]) \geq 1 - \frac{2^L \cdot (e+1)}{q^d} \geq \frac{1}{2}$ (because $d = L + log(e)$).
  $z(x) \neq 0$ in $\mathbb{Z}_q[x]$ implies $gcd_q(z(x), h(x))$ has degree 0 because $h(x)$ is irreducible modulo q.
  Therefore the probability that the Euclid's algorithm on $h(x)$ and $z(x)$ fails is at least $\frac{1}{4d} \cdot \mu \cdot \frac{1}{2} = \frac{\mu}{8d}$.

Now we compute the time complexity of one run of the loop.
Generating $h(x)$ and $r(x)$ can be done by choosing d values uniformly at random from $\mathbb{Z}_N$ and can be done in time $\Theta(d)$. Computing the derivative requires $\Theta(d)$ operations in $\mathbb{Z}_N$.

By Lemma 3, there exists a SLP on $\mathbb{Z}_N$ with $4L$ steps that contains $f(\alpha)$ and $g(\alpha)$ that uses only the operations $\{+, -, \cdot\}$. e^{th} power can be computed by such a SLP using $log(e)$ steps. So, we can find an $O(L + log(e))$ step SLP on $\mathbb{Z}_N$ that computes $P(x)$ for a given $x \in \mathbb{Z}_N$.

Each multiplication operation in $\mathbb{Z}_N[x]/h(x)$ can be implemented by at most d^2 multiplication operations and at most d^2 additions operations in $\mathbb{Z}_N$. Each addition operation can be performed by $O(d)$ operations in $\mathbb{Z}_N$. Therefore, $P(r(x)) = z(x)$ can be computed in time $O(d^2 \cdot L + d^2 \cdot log(e)) = O(d^3)$. Euclid's algorithm on $z(x)$ and $h(x)$ can be performed by $O(d(log(N) + d))$ operations. So, the running time of one loop of the algorithm is $O(d^3 + log(N)d)$.

The expected number of times we need to run this loop is $\Theta(\frac{d}{\mu})$. So the expected running time of the algorithm is $O(d^3+log(N)d)\cdot\Theta(\frac{d}{\mu}) = O(\frac{d^4+log(N)d^2}{\mu}) = O(\frac{d^4+log(N)^2}{\mu})$. Since $d = log(e) + L$, we get an upper bound on the expected running time of the factoring algorithm as $O(\frac{L^4+log^4(e)+log^2(N)}{\mu})$.

□

Using Lemma 5 and Lemma 8, we get the following result.

Theorem 9. *The Generic RSA assumption on $\mathbb{Z}_N$ holds if and only if factoring N is hard.*

4. Conclusions and open problems

In this paper we showed that if factoring is hard, then no generic algorithm can solve the RSA problem efficiently. This solves, in the generic model, the long-standing open problem of the equivalence of factoring and breaking RSA. There are yet other problems that can be looked at in this model. For instance, the Cramer-Shoup cryptosystem and signature scheme relies on the "Strong RSA Assumption" [8, 1], which allows the adversary to himself choose an exponent $e > 1$. A natural question would be whether we can show that factoring is equivalent to solving strong RSA using a generic algorithm. It is not clear whether this statement is true. The proof of Lemma 8, however, does not work for this case because here e will depend on the input α. As a result, in the proof of Lemma 8, $P(\alpha) = f(\alpha)^e - \alpha \cdot g(\alpha)^e$ is not a polynomial in α (because the exponent is not independent of α).

References

[1] N. Baric and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In Jr. Burton S. Kaliski, editor, *EUROCRYPT 1997* volume 1233 of *Lecture Notes in Computer Science*, pages 480-494. Springer-Verlag 1997.

[2] D. Boneh and R. Venkatesan. Breaking RSA may be easier than factoring. In K. Nyberg, editor, *Advances in Cryptology - EUROCRYPT 1998*, volume 1403 of *Lecture Notes in Computer Science*, pages 59-71. IACR, Springer, 1998.

[3] D. R. L. Brown. Breaking RSA may be as difficult as factoring. *Cryptology ePrint Archive*, Report 205/380, 2006.

[4] L. Childs. *A concrete introduction to higher algebra*. New York: Springer-Verlag, 1992.

[5] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In Hugo Krawczyk, editor, *CRYPTO 1998*, volume 1462 of *Lecture Notes in Computer Science*, pages 13-25. Springer-Verlag 1998.

[6] R. Cramer and V. Shoup. Signature schemes based on the strong RSA assumption. In *6th ACM Conference on Computer and Communications Security*, pages 46-52. ACM, Nov 1999.

[7] I. Damgard and M. Koprowski. Generic lower bounds for root extraction and signature schemes in general groups. in *Advances in Cryptology - EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 256-271. Springer-Verlag, 2002.

[8] E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In Jr. Burton S. Kaliski, editor, *CRYPTO 1997* volume 1294 of *Lecture Notes in Computer Science*, pages 16-30. Springer-Verlag 1997.

[9] S. Hohenberger. The cryptographic impact of groups with infeasible inversion. Master's thesis, Massachussetts Institute of Technology, EECS Dept., Cambridge, MA, June 2003.

[10] G. Leander and A. Rupp. On the Equivalence of RSA and Factoring Regarding Generic Ring Algorithms. In *Asiacrypt 2006* volume 4284 of *Lecture Notes in Computer Science*, pages 241-251. Springer-Verlag 2006.

[11] R. Lidl and H. Niederreiter. *Introduction to finite fields and their applications*. Cambridge University Press, 1994.

[12] U. Maurer. Abstract models of computation in cryptoraphy. In Nigel P. Smart, editor, *Cryptography and Coding 2005*, volume 3796 of *Lecture Notes in Computer Science*, pages 1-12. Springer-Verlag, 2005.

[13] D. Micciancio. The RSA group is pseudo-free. In R. Cramer, editor, *EUROCRYPT 2005* volume 3494 of *Lecture Notes in Computer Science*, pages 387-403. Springer-Verlag 2005.

[14] V. I. Nechaev. Complexity of a deterministic algorithm for the discrete logarithm. *Mathematical Notes*, volume 55, no. 2, pages 91-101, 1994.

[15] R.L. Rivest. On the notion of pseudo-free groups. In M. Naor, editior, *Theory of Cryptography conference - TCC 2004*, volume 2951 of *Lecture Notes in Computer Science*, pages 505-521, Cambridge, MA, USA, Feb. 2003. Springer.

[16] R. Rivest, A. Shamir and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, volume 21: pages 120-126, 1978.

[17] V. Shoup. Lower bounds for discrete logarithms and related problems. In Jr. Burton S. Kaliski, editor, *EUROCRYPT 1997* volume 1233 of *Lecture Notes in Computer Science*, pages 256-266. Springer-Verlag 1997.