Breaking RSA Generically is Equivalent to Factoring

Divesh Aggarwal and Ueli Maurer

Department of Computer Science
ETH Zurich
CH-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

Probably the two most fundamental reduction problems in number-theoretic cryptography are to prove or disprove that breaking the RSA system [22] is as hard as factoring integers and that breaking the Diffie-Hellman protocol [9] is as hard as computing discrete logarithms. While the second problem has been solved to a large extent [15, 18, 2], not much is known about the first for general models of computation. In this paper, we show the equivalence of RSA and factoring for the most general generic model of computation.

1.1 Breaking RSA vs. Factoring

The security of the well-known RSA public-key encryption and signature scheme [22] relies on the assumption that computing e^{th} roots modulo n, which is a product of two primes, is hard. In order to formally state this assumption, we define a pair of random variables, N and E, that are chosen according to a certain joint probability distribution as follows: N is a product of two primes, for example an element chosen uniformly at random from the set of products of two k-bit primes satisfying certain conditions (e.g. [16]), and E is a positive integer[1] such that $\gcd(E, \phi(N)) = 1$. Note that the marginal distribution of N over products of two primes is defined by the joint distribution of (N, E). Throughout this paper, we will assume that the sum of the lengths of N and E is bounded by the security parameter κ and the terms negligible, non-negligible, and polynomial-time are with respect to κ. We state three assumptions below:

[1] In principle E can be much larger than N.

Factoring Assumption: There exists no probabilistic polynomial-time algorithm that, given N, finds a non-trivial factor of N with non-negligible[2] probability.

RSA Assumption: There exists no probabilistic polynomial-time algorithm that, given the pair (N, E) and an element a chosen uniformly at random from $\mathbb{Z}_N^*$, computes $x \in \mathbb{Z}_N^*$ such that $x^E \equiv a$ modulo N with non-negligible probability.

Generic RSA Assumption: There exists no probabilistic polynomial-time *generic* ring algorithm (a class of algorithms that we will introduce later) that, given the pair (N, E) and an element a chosen uniformly at random from $\mathbb{Z}_N^*$, computes $x \in \mathbb{Z}_N^*$ such that $x^E \equiv a$ modulo N with non-negligible probability.

It is easy to see that if the RSA assumption holds then the factoring assumption holds. However it is a long-standing open problem whether the converse is true. Since no progress has been made for general models of computation, it is 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. Shoup [23], based on the work of Nechaev [20], introduced 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 cyclic groups in the context of generic algorithms. Maurer [17] provided a simpler and more general model for analyzing representation-independent algorithms.

1.2 The Generic Model of Computation

We give a brief description of the model of [17]. The model is characterized by a black-box $\mathbf{B}$ which can store values from a certain set T in internal state variables $V_0, V_1, V_2, \cdots$. The initial state (the input of the problem to be solved) consists of the values of $[V_0, \ldots, V_\ell]$ for some positive integer ℓ, 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.* For a set Π of operations of some arities on T, a computation operation consists of selecting $f \in \Pi$ (say t-ary) as well as the indices $i_1, \ldots, i_{t+1}$ of $t+1$ state variables. $\mathbf{B}$ computes $f(V_{i_1}, \ldots, V_{i_t})$ and stores the result in $V_{i_{t+1}}$.
- *Relation Queries.* For a set Σ of relations (of some arities) on T, a query consists of selecting a relation $\rho \in \Sigma$ (say t-ary) as well as the indices $i_1, \ldots, i_t$ of t state variables. The query is replied by the binary output $\rho(V_{i_1}, \ldots, V_{i_t})$ that takes the value 1 if the relation is satisfied, and 0 otherwise.

For this paper, we only consider the case $t = 2$ and the only relation queries we consider are equality queries.

[2] A function $f(\kappa)$ is considered a non-negligible function in κ if there exists $c > 0$ and $k_0 \in \mathbb{N}$ such that for all $\kappa > k_0$, $|f(\kappa)| > \frac{1}{\kappa^c}$.

An algorithm in this model is characterized by its interactions with the black box $\mathbf{B}$. The algorithm inputs operations (computation operations and relation queries) to the black box, and the replies to the relation queries are input to the algorithm. The complexity of an algorithm for solving any problem can be measured by the number of operations it performs on $\mathbf{B}$.

For this paper, the set T is $\mathbb{Z}_N$. Also $\ell = 1$ and V_0 is always set to be the unit element 1 of $\mathbb{Z}_N$ and V_1 is the value a whose E^{th} root is to be computed. A *generic ring algorithm (GRA)* is an algorithm that is just allowed to perform the ring operations, i.e., addition and multiplication as well as the inverse ring operations (subtraction and division), and to test for equality (i.e., make an equality query). In this model, for example, GRAs on $\mathbb{Z}_N$ correspond to $\Pi = \{+, -, \cdot, /\}$ and $\Sigma = \{=\}$. A *straight-line program (SLP)* on $\mathbb{Z}_N$, which is a deterministic algorithm that is just allowed to perform ring operations, corresponds to the case where Σ is the empty set, i.e., no equality tests are possible.

Many results in the literature are restricted in that they exclude the inverse operations, but since these operations are easy[3] to perform in $\mathbb{Z}_N$, they should be included as otherwise the results are of relatively limited interest. Note that division by non-invertible elements of $\mathbb{Z}_N$ is not defined. This can be modeled in the above generic model by having the black-box $\mathbf{B}$ send an "exception" bit b to the algorithm and leaving the corresponding state variable undefined whenever there is a division by a non-invertible element. In order to avoid having to handle these exceptions, we instead describe a black-box $\widetilde{\mathbf{B}}$ that (we will show) is almost as powerful as $\mathbf{B}$, but easier to work with.

$\widetilde{\mathbf{B}}$ stores values from $\mathbb{Z}_N \times \mathbb{Z}_N$. V_0 and V_1 are set to be $(1,1)$ and $(x,1)$ respectively. The operations in $\{+, -, \cdot, /\}$ are defined on $\mathbb{Z}_N \times \mathbb{Z}_N$ as follows (for $\alpha, \beta, \gamma, \delta \in \mathbb{Z}_N$):

$$(\alpha, \beta) \circ (\gamma, \delta) = \begin{cases} (\alpha\delta + \beta\gamma, \beta\delta) & \text{if } \circ \text{ is } + \\ (\alpha\delta - \beta\gamma, \beta\delta) & \text{if } \circ \text{ is } - \\ (\alpha\gamma, \beta\delta) & \text{if } \circ \text{ is } \cdot \\ (\alpha\delta, \beta\gamma) & \text{if } \circ \text{ is } / \end{cases}$$

If (α, β) and (γ, δ) are queried for equality, 1 is returned if $\alpha\delta = \beta\gamma$, and 0 otherwise.

The interpretation of $\widetilde{\mathbf{B}}$ is that if an internal state variable in $\widetilde{\mathbf{B}}$ takes a value (α, β) then, on inputting the same sequence of operations to $\mathbf{B}$, the corresponding internal state variable in $\mathbf{B}$ takes the value α/β if there was no "exception". We will show in Section 2.2 that any GRA (for solving a certain computation problem) interacting with $\mathbf{B}$ can be converted into a GRA interacting with $\widetilde{\mathbf{B}}$ such that both have essentially the same success probability and complexity of the same order.

[3] Division of an element a by b for $a, b \in \mathbb{Z}_N$ can be performed easily by first computing b^{-1} using Euclid's algorithm and then computing $a \cdot b^{-1}$ in $\mathbb{Z}_N$.

1.3 Related Work and Contributions of this Paper

Research on the relation between RSA and factoring comes in two flavours. There have been results giving evidence against (e.g. [3, 12]) and in favour of (e.g. [4, 13]) the assumption that breaking RSA is equivalent to factoring.

Boneh and Venkatesan [3] showed that any SLP that factors N by making at most a logarithmic number of queries to an oracle solving the Low-Exponent RSA (LE-RSA) problem (the 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 that makes a small number of queries to the LE-RSA oracle. Joux et al [12] showed that computing e-th roots modulo N is easier than factoring N with currently known methods, given sub-exponential access to an oracle outputting the roots of numbers of the form $x_i + c$.

Brown [4] showed that if factoring is hard then the LE-RSA problem is intractable for SLPs with $\Pi = \{+, -, \cdot\}$. More precisely, he proved that an efficient SLP for breaking LE-RSA can be transformed into an efficient factoring algorithm. Leander and Rupp [13] generalized the result of [4] to GRAs which, as explained above, can test the equality of elements. Again, division is excluded ($\Pi = \{+, -, \cdot\}$).

Another theoretical result about the hardness of the RSA problem is due to Damgård and Koprowski [8]. 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, subtraction and division from the set of operations ($\Pi = \{\cdot\}$).

Our results generalize the previous results in several ways. (Actually, Theorem 1 appears to be the most general statement about the equivalence of factoring and breaking RSA in a generic model.)

- First, compared to [4, 13] 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, we allow for *randomized* GRAs, an extension less trivial than it might appear.
- Third, compared to [8, 4, 13] we consider the unrestricted set of ring operations, including division. 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 division[4].

  Actually, as has been pointed out in [4], computing the multiplicative inverse of a random element in $\mathbb{Z}_N$ generically is hard if $\Pi = \{+, -, \cdot\}$.
- Fourth, compared to [13] we give an explicit algorithm that factors N given a GRA computing E^{th} roots. In [13], the reduction from GRAs to SLPs is only non-uniform.

[4] In [4], the author has mentioned and given justification for the fact that most results of his paper will extend to SLPs with division.

The problem we solve has been stated as an open problem in [8] and [4].

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 prove our main result. Section 4 provides some conclusions and lists some open problems.

2 Preliminaries

2.1 Straight-Line Programs

Straight-line programs for a ring are deterministic algorithms that perform a sequence of ring operations. Thus an SLP corresponds to $\Pi = \{+, -, \cdot, /\}$ and $\Sigma = \{\}$ in the model of [17]. More concretely:

Definition 1. An L-step *straight-line program (SLP)* S is a sequence of triples $(i_2, j_2, \circ_2), \ldots, (i_L, j_L, \circ_L)$ such that $0 \leq i_k, j_k < k$ and $\circ_k \in \{+, -, \cdot, /\}$.

The interpretation for this is that if x is considered a variable, then the SLP S computes a sequence of rational functions $f_2, \ldots, f_L (= f^S)$, where, $f_0 = 1$, $f_1 = x$ and $f_k = f_{i_k} \circ_k f_{j_k}$ for $2 \leq k \leq L$. Thus, an SLP can be interpreted as the evaluation of a rational function f^S in x given by a pair of polynomials (P^S, Q^S) representing the numerator and the denominator of f^S, respectively, as follows.

1. Let $P_0^S = 1, Q_0^S = 1, P_1^S = x, Q_1^S = 1$.
2. For $2 \leq k \leq L$, $(P_k^S, Q_k^S) = (P_{i_k}^S, Q_{i_k}^S) \circ_k (P_{j_k}^S, Q_{j_k}^S)$ is given by:

$$(P_k^S, Q_k^S) = \begin{cases} (P_{i_k}^S \cdot Q_{j_k}^S + P_{j_k}^S \cdot Q_{i_k}^S, Q_{i_k}^S \cdot Q_{j_k}^S) & \text{if } \circ_k \text{ is } + \\ (P_{i_k}^S \cdot Q_{j_k}^S - P_{j_k}^S \cdot Q_{i_k}^S, Q_{i_k}^S \cdot Q_{j_k}^S) & \text{if } \circ_k \text{ is } - \\ (P_{i_k}^S \cdot P_{j_k}^S, Q_{i_k}^S \cdot Q_{j_k}^S) & \text{if } \circ_k \text{ is } \cdot \\ (P_{i_k}^S \cdot Q_{j_k}^S, Q_{i_k}^S \cdot P_{j_k}^S) & \text{if } \circ_k \text{ is } / \end{cases}$$

3. Output $(P^S, Q^S) = (P_L^S, Q_L^S)$.

Therefore, from now on, we identify the SLP S with the pair of polynomials (P^S, Q^S).

Lemma 1. *P_k^S and Q_k^S are polynomials of degree at most 2^k.*

Proof. The proof is by induction on k. The claim is trivially true for $k = 0$. We assume that P_r^S and Q_r^S are polynomials of degree at most 2^r for all $r < k$. So, since $i_k, j_k \leq k - 1$, $deg(P_{i_k}^S)$, $deg(Q_{i_k}^S)$, $deg(P_{j_k}^S)$ and $deg(Q_{j_k}^S)$ are all at most 2^{k-1}. This implies, by the definition of (P_k^S, Q_k^S), that the degree of P_k^S and Q_k^S is at most $2^{k-1} + 2^{k-1} = 2^k$. $\square$

Next, we show that any SLP can be "converted" into an SLP that does not require the division operation, i.e., for which $\Pi = \{+, -, \cdot\}$, without increasing the complexity by more than a constant factor. A result similar to the following but with a factor of 6 instead of 4 has been proven independently in [11].

Lemma 2. *For any L-step SLP S with $\Pi = \{+, -, \cdot, /\}$ that computes the rational function f^S given by (P^S, Q^S), there exists a $4L$-step SLP with $\Pi = \{+, -, \cdot\}$ that computes P^S and Q^S.*

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 an SLP S' of length $\ell \leq 4L'$ that uses only the operations $\{+, -, \cdot\}$ and computes P^S_k and Q^S_k for $1 \leq k \leq L'$. Let this program compute the polynomials R_i for $1 \leq i \leq \ell$. Let $L = L' + 1$. Now consider the following cases:

– Case (i): $\circ_L$ is $+$.
Let $R_{\ell+1} = P^S_{i_L} \cdot Q^S_{j_L}$, $R_{\ell+2} = P^S_{j_L} \cdot Q^S_{i_L}$, $R_{\ell+3} = R_{\ell+1} + R_{\ell+2} = P^S_L$ and $R_{\ell+4} = Q^S_{i_L} \cdot Q^S_{j_L} = Q^S_L$.
– Case (ii): $\circ_L$ is $-$.
Let $R_{\ell+1} = P^S_{i_L} \cdot Q^S_{j_L}$, $R_{\ell+2} = P^S_{j_L} \cdot Q^S_{i_L}$, $R_{\ell+3} = R_{\ell+1} - R_{\ell+2} = P^S_L$ and $R_{\ell+4} = Q^S_{i_L} \cdot Q^S_{j_L} = Q^S_L$.
– Case (iii): $\circ_L$ is $\cdot$.
Let $R_\ell = P^S_{i_L} \cdot P^S_{j_L} = P^S_L$ and $R_{\ell+2} = Q^S_{i_L} \cdot Q^S_{j_L} = Q^S_L$.
– Case (iv): $\circ_L$ is $/$.
Let $R_{\ell+1} = P^S_{i_L} \cdot Q^S_{j_L} = P^S_L$ and $R_{\ell+2} = Q^S_{i_L} \cdot P^S_{j_L} = Q^S_L$.

By induction hypothesis, SLP S' computes P^S_k and Q^S_k for $1 \leq k \leq L'$. We can extract from S', the indices corresponding to each of P^S_k and Q^S_k for $1 \leq k \leq L'$. Therefore, in each of the cases mentioned above, we get an SLP of length at most $\ell + 4 \leq 4(L' + 1) = 4L$ that computes P^S_k and Q^S_k for $1 \leq k \leq L' + 1$. □

Note that the SLP described in the above lemma does not compute the rational function f^S but only computes the two polynomials P^S and Q^S. This, however, is sufficient for our purpose.

2.2 Generic Ring Algorithms

A deterministic generic ring algorithm can be seen as a generalized SLP that also allows equality queries. More concretely:

Definition 2. An L-step *deterministic generic ring algorithm (deterministic GRA)* G is an algorithm that, in the k^{th} step, for $2 \leq k \leq L$, outputs an operation of the form $(i_k, j_k, \circ_k)$, where $0 \leq i_k, j_k < k$ are some positive integers and $\circ_k \in \{+, -, \cdot, /, eq\}$. It takes as input an "exception" bit b which is 1 if the operation is not defined, and 0 otherwise. If b is 0 and $\circ_k$ is eq, it takes another input bit (as the result of the equality query).

The interpretation for this definition is that if x is considered a variable, then G computes a sequence of rational functions $f_2, \ldots, f_L$, where $f_0 = 1$, $f_1 = x$, and $f_k = f_{i_k} \circ_k f_{j_k}$ if $\circ_k \in \{+, -, \cdot, /\}$ and $f_k = f_{k-1}$ if $\circ_k$ is eq for $2 \leq k \leq L$. Also, $f_k = \perp$ if f_{j_k} is not invertible and $\circ_k$ is $/$ or one of f_{i_k} and f_{j_k} is $\perp$.

If G is to be executed on an input $a \in \mathbb{Z}_N$, this is modeled, as mentioned in Section 1.2, by G interacting with the black-box $\mathbf{B}$ where an operation of the form (i, j, eq) is an equality query on the i^{th} and j^{th} elements in $\mathbf{B}$. We now show that if a GRA G is successful in solving a certain computation problem for $\mathbf{B}$ with probability γ, then the sum of the probabilities that there is a non-trivial non-invertible element computed by G in $\mathbf{B}$ (in which case this element can be used to factorize N) and that there is a GRA (of double the length of G) that is successful in solving the same computational problem for $\widetilde{\mathbf{B}}$, where $\widetilde{\mathbf{B}}$ is as defined in Section 1.2, is at least γ. Since our aim is to reduce the problem of factoring N to computing the E-th roots modulo N in the generic model of Section 1.2, it is therefore sufficient to prove the result in the generic model replacing $\mathbf{B}$ by $\widetilde{\mathbf{B}}$.

Consider a black box $\mathbf{B}'$ that behaves exactly like $\mathbf{B}$ except that it does not output an exception bit. Any GRA G interacting with $\mathbf{B}$ can be converted into a GRA G' interacting with $\mathbf{B}'$ as follows, such that $\mathbf{B}$ and $\mathbf{B}'$ have identical behaviour internally. For each new value computed inside $\mathbf{B}'$, G' makes an equality query of the computed value and 0 and stores the result internally. Also, it maintains internally a list of indices of state variables which are undefined. Then, G' can internally simulate the exception bit that G receives from $\mathbf{B}$ by setting the exception bit as 1 if and only if there is a division by 0 in $\mathbf{B}'$ or an operation is performed on two values, one of which is undefined. Thus G' performs at most twice the number of steps as G and gets the same input as G if there is no division by a non-trivial non-invertible element in $\mathbf{B}$.

By definition of the operations $\{+, -, \cdot, /, eq\}$ in $\widetilde{\mathbf{B}}$, if a GRA performing a sequence of operations computes a value α in $\mathbf{B}'$ and the same sequence of operations computes a pair (α_1, α_2) in $\widetilde{\mathbf{B}}$, then $\alpha = \frac{\alpha_1}{\alpha_2}$ if α is not undefined.

Thus any GRA G that computes a certain value in $\mathbf{B}$ (without any division by a non-trivial non-invertible element of $\mathbb{Z}_N$) can be converted into a GRA G' that computes a pair of values such that if the first element of this pair is divided by the second, we get the value computed in $\mathbf{B}$ (unless, this value is undefined). Hence, from now on, we will consider only the black-box $\widetilde{\mathbf{B}}$ and therefore assume that the corresponding GRA is never returned 1 as the "exception" bit. As a result, we can ignore the "exception" bit.

Note that for a given sequence of bits input to it, a deterministic GRA G behaves like an SLP (that performs a trivial operation of copying the previous value whenever $\circ_k$ is eq). A deterministic GRA can be interpreted as a binary tree T^G with each vertex corresponding to an operation from the set $\{+, -, \cdot, /, eq\}$. The vertices corresponding to an eq operation have two children and all other vertices have one child. The edges between a vertex corresponding to an eq operation and its left and right child are labeled 0 and 1, respectively, while all the other edges do not have a label. An execution of G corresponds to a path from the root to a leaf of T^G. The choice of the subsequent vertex at each vertex corresponding to an operation eq is made by choosing the edge that has as label the bit input to G at the corresponding step. The sequence of operations corresponding to vertices along each path from the root to a vertex of T^G is an SLP, and so we can associate a pair of polynomials (using the definition of

SLP) with each vertex of T^G. The pair of polynomials associated with vertex v is denoted by (P^v, Q^v).

Definition 3. A *randomized generic ring algorithm* $\mathcal{G}$ is a GRA where the choice of the operation at each step is randomized. A randomized GRA can be understood as a random variable whose values are deterministic GRAs.

2.3 Mathematical Preliminaries

In this section we introduce some notations and prove some results about the mathematical structures used in this paper.

For integers a, b, c, we denote by $a \equiv_c b$, that a is congruent to b modulo c. For any event E, we denote the probability of E by $\mathsf{P}(\mathsf{E})$.

For the rest of the paper, we assume that the random variable N takes values from some set $\mathcal{N}$. Furthermore, (n, e), where $n = pq$, denotes a value taken by the pair (N, E). By $\mathbb{Z}_n[x]$, we denote the ring of polynomials in x with coefficients in $\mathbb{Z}_n$ and for $h(x) \in \mathbb{Z}_n[x]$ by $\mathbb{Z}_n[x]/h(x)$ quotient of the ring $\mathbb{Z}_n[x]$ by a principal ideal generated by an irreducible polynomial $h(x)$. For $P(x) \in \mathbb{Z}_n[x]$, we define the following.

– Let $\nu_n(P)$ be the fraction of roots of P in $\mathbb{Z}_n$, i.e.,

$$\nu_n(P) = \frac{|\{x \in \mathbb{Z}_n | P(x) \equiv_n 0\}|}{n} .$$

Similarly we define $\nu_p(P)$ and $\nu_q(P)$.

– The fraction of elements a in $\mathbb{Z}_n$ such that $P(a)$ has a non-trivial greatest common divisor with n is defined as $\eta_n(P)$,

$$\eta_n(P) = \frac{|\{x \in \mathbb{Z}_n | \gcd(P(a), n) \notin \{1, n\}\}|}{n} .$$

We prove three lemmas that we will need later. The reader may skip to Section 3 and return to the following lemmas when they are referenced.

Lemma 3. *For any $P(x) \in \mathbb{Z}_n[x]$, if $\nu_n(P) \in [\delta, 1-\delta]$, then $\eta_n(P) \geq \delta^{\frac{3}{2}}$.*

Proof. We denote $\nu_p(P)$ and $\nu_q(P)$ by ν_p and ν_q, respectively. By the Chinese remainder theorem, $\nu_n(P) = \nu_p \cdot \nu_q$ and $\eta_n(P) = \nu_p(1-\nu_q) + \nu_q(1-\nu_p)$. Using $\delta \leq \nu_p \cdot \nu_q \leq 1 - \delta$, we obtain

$$\begin{aligned}
\eta_n(P) &= \nu_p(1-\nu_q) + \nu_q(1-\nu_p) = \nu_p + \nu_q - 2\nu_p \cdot \nu_q \\
&= (\sqrt{\nu_p} - \sqrt{\nu_q})^2 + 2\sqrt{\nu_p \cdot \nu_q} - 2\nu_p \cdot \nu_q \\
&\geq 2\sqrt{\nu_p \cdot \nu_q} - 2\nu_p \cdot \nu_q = 2\sqrt{\nu_p \cdot \nu_q}(1 - \sqrt{\nu_p \cdot \nu_q}) \\
&\geq 2\sqrt{\delta}(1 - \sqrt{1-\delta}) \\
&\geq 2\sqrt{\delta}(1 - (1 - \tfrac{\delta}{2})) = \delta^{\frac{3}{2}} .
\end{aligned}$$

□

Lemma 4. *Let p be a prime. A random monic polynomial $f(x) \in \mathbb{Z}_p[x]$ of degree d 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., [14]) it follows that the number of monic irreducible polynomials of degree d over F_p is at least $\frac{p^d}{2d}$. Therefore $f(x)$ is 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:

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

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 $1/2$.[5] □

Lemma 5. *For any discrete random variable X that takes values in [0,1] and for any $\tau > 0$, $\mathsf{P}(X \geq \tau) \geq E[X] - \tau$.*

Proof.

$$\begin{aligned} \mathrm{E}[X] &= \sum_{x \geq \tau} x \cdot \mathsf{P}(X = x) + \sum_{x < \tau} x \cdot \mathsf{P}(X = x) \\ &\leq \sum_{x \geq \tau} \mathsf{P}(X = x) + \sum_{x < \tau} \tau \cdot \mathsf{P}(X = x) \\ &\leq \mathsf{P}(X \geq \tau) + \tau \ , \end{aligned}$$

which implies the result. □

3 The Main Theorem

3.1 Statement of the Theorem

As mentioned earlier, in this paper we restrict our attention to the case where the adversary is only allowed to use a GRA to solve the RSA problem. We refer to the RSA assumption in this case as the generic RSA assumption for the random variables (N, E) that was introduced in Section 1.1.

We state the main result of the paper.

[5] Note that, by a careful analysis, it is possible to prove a better lower bound on the probability that $f(x)$ has a root in $\mathbb{Z}_p$ but a lower bound of $1/2$ is sufficient for our purpose.

Theorem 1. *For any pair (N, E), the factoring assumption holds for N implies that the generic RSA assumption holds for (N, E).*

Remark: In the proof of this theorem, we give an algorithm that, for every $n \in \mathcal{N}$ for which there is a GRA that computes the e^{th} root of a uniformly random element chosen from $\mathbb{Z}_n$, factors n with overwhelming probability. The factoring assumption and the generic RSA assumption are, however, stated for the random variable N and not for a fixed n, as the factoring problem would otherwise not be defined reasonably, and also the terms polynomial-time and non-negligible would not make sense for a fixed n. Hence, our proof actually proves a stronger statement than Theorem 1.

3.2 Proof of the Theorem

3.2.1 Overview of the Proof

In Section 3.2.2, we show that an SLP that computes e^{th} roots can be used to factor n. Then, in Section 3.2.3, we show that from a deterministic GRA that computes e^{th} roots we can either obtain an SLP that computes e^{th} roots or a factor of n. In Section 3.2.4, we generalize the results of Section 3.2.3 from deterministic GRAs to randomized GRAs. In Section 3.2.5, we combine the results of Section 3.2.2 and 3.2.4 to show that a randomized GRA that computes e^{th} roots can be used to give an algorithm for factoring n.

3.2.2 The Proof for Straight-Line Programs

In this section we give an algorithm that, with non-negligible probability, factors n given access to an SLP that, with non-negligible probability, computes the e^{th} root of an element chosen uniformly at random from $\mathbb{Z}_n$.

For polynomials $b(x), c(x) \in \mathbb{Z}_n[x]$, let $\gcd_p(b(x), c(x))$ and $\gcd_q(b(x), c(x))$ be the greatest common divisor of the polynomials modulo p and q, respectively. The following proposition is easy to see.

Proposition 1. *Let $b(x), c(x) \in \mathbb{Z}_n[x]$. Then:*

- *If Euclid's algorithm, when run on $b(x)$ and $c(x)$, fails*[6]*, some step of the algorithm yields a non-trivial non-invertible element of $\mathbb{Z}_n$. We denote this element as $H(b(x), c(x))$.*
- *If $deg(gcd_p(b(x), c(x))) \neq deg(gcd_q(b(x), c(x)))$, then Euclid's algorithm, when run on $b(x)$ and $c(x)$, fails.*

Lemma 6. *For all $\epsilon > 0$, $\mu > 0$, and $L \in \mathbb{N}$, there exists an algorithm of time complexity $O(L^3 + log^3(e))$ that, for every SLP S such that $\nu_n((P^S)^e - x(Q^S)^e) \geq \mu$ and $Q^S(x)$ is not the zero polynomial, returns a factor of n with probability $\frac{\mu}{8(L+log(e))}$.*

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

Proof. Let $f(x) = P^S(x)^e - x \cdot Q^S(x)^e$. Then $\nu_n(f) \geq \mu$. By Lemma 1, $deg(f) \leq 2^L e + 1$. By Lemma 2, there is a $4L$-step SLP that uses only the operations $\{+, -, \cdot\}$ and generates the polynomials $P^S(x)$ and $Q^S(x)$. Given $P^S(x)$ and $Q^S(x)$, $P^S(x)^e$ and $Q^S(x)^e$ can be computed in $2\lceil log(e) \rceil$ steps each. Therefore there is an SLP S_1 with at most $4L + 4\lceil log(e) \rceil + 2$ steps that computes $f^{S_1}(x) = f(x)$. For the factoring algorithm, we use this SLP. Let $d = L + \lceil log(e) \rceil$. The factoring algorithm proceeds as follows:

```
Algorithm 1: Factoring Algorithm
  Input: $n$, SLP $S_1$
  Output: A factor of $n$
1 Choose a monic polynomial $h(x)$ uniformly at random from all monic
  polynomials of degree $d$ in $\mathbb{Z}_n[x]$;
2 Compute $h'(x)$, the derivative of $h(x)$ in $\mathbb{Z}_n[x]$;
3 Choose a random element $r(x) \in \mathbb{Z}_n[x]/h(x)$;
4 Compute $z(x) = f(r(x))$ in $\mathbb{Z}_n[x]/h(x)$ using SLP $S_1$;
5 Run Euclid's algorithm in $\mathbb{Z}_n[x]$ on $h(x)$ and $z(x)$. If this fails return
  $\gcd(n, H(h(x), z(x)))$;
6 Run Euclid's algorithm in $\mathbb{Z}_n[x]$ on $h(x)$ and $h'(x)$. If this fails return
  $\gcd(n, H(h(x), h'(x)))$;
```

By Proposition 1, if Euclid's algorithm fails in step 5 or step 6, then we get a factor of n.

Now we compute the success probability of the algorithm. By Lemma 4, 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 that this is the case for the rest of the proof.

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

- 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 $\mathbb{Z}_q[x]$, $\gcd_q(h(x), h'(x))$ has degree 0. Therefore $\gcd_p(h(x), h'(x))$ and $\gcd_q(h(x), h'(x))$ have different degree, which implies, by Proposition 1, that Euclid's algorithm on $h(x)$ and $h'(x)$ fails and hence step 6 yields a factor of n.
- CASE 2: $(x - s)^2 \nmid h(x)$ in $\mathbb{Z}_p[x]$.
  Let $h(x) = h_1(x) \cdot (x - s)$ in $\mathbb{Z}_p[x]$. Then:

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

  because $\mathbb{Z}_q[x]/h(x) \cong \mathbb{F}_{q^d}$ (the finite field containing q^d elements) as $h(x)$ is irreducible in $\mathbb{Z}_q[x]$ by our assumption.
  Under this isomorphism, let $r(x)$ maps to the triple

  $$(r(s) \bmod p, \ u(x), \ r_q(x)) \ ,$$

and $z(x)$ to the triple

$$(z(s) \bmod p,\ v(x),\ z_q(x)) ,$$

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

$$\mathsf{P}\big(z(s) \equiv_p 0\big) \ = \ \mathsf{P}\big(f(r(s)) \ \equiv_p \ 0\big) \ \geq \ \mathsf{P}\big(f(r(s)) \equiv_n 0\big) \ \geq \ \mu \ .$$

Therefore, with probability at least μ, $(x-s)$ divides $z(x)$ in $\mathbb{Z}_p[x]$, which implies $\mathsf{P}((x-s) \mid \gcd_p(z(x), h(x))) \geq \mu$. Since $r(x)$ is uniformly random in $\mathbb{Z}_n[x]/h(x)$, $r_q(x)$ is uniformly random in $\mathbb{Z}_q[x]/h(x) \cong \mathbb{F}_{q^d}$. A polynomial over a finite field can have at most as many roots as the degree of the polynomial. Therefore, for random x,

$$\mathsf{P}\big(z_q(x) = 0\big) \ = \ \mathsf{P}\big(f(r_q(x)) = 0\big) \ \leq \ \frac{deg(f)}{q^d} \ \leq \ \frac{2^L e + 1}{q^d} \ \leq \ \frac{1}{2} \ ,$$

using the fact that $d = L + \lceil log(e) \rceil$ and $deg(f) \leq 2^L e + 1$. Hence, $\mathsf{P}(z_q(x) \neq 0) \geq \frac{1}{2}$. The condition $z_q(x) \neq 0$ implies that $\gcd_q(z(x), h(x))$ has degree 0 because $h(x)$ is irreducible modulo q.

Therefore the probability that Euclid's algorithm run 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 random $h(x)$ and $r(x)$ and computing the derivative requires $O(d)$ operations in $\mathbb{Z}_n$. Each operation in $\mathbb{Z}_n[x]/h(x)$ can be implemented by at most d^2 operations in $\mathbb{Z}_n$. The function f is computed in $\mathbb{Z}_n$ using an at most $(4L + 4\lceil log(e) \rceil + 2)$-step SLP S_1. Therefore, $f(r(x)) = z(x)$ 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 $h(x)$ and on $h(x)$ and $h'(x)$ can be performed by $O(d^2)$ operations. Thus, the running time of the algorithm is $O(d^3)$. $\square$

3.2.3 From Deterministic GRAs to SLPs

In this section we give an algorithm that, given access to a deterministic GRA that computes e^{th} roots in $\mathbb{Z}_n$, outputs either a factor of n or an SLP that computes e^{th} roots in $\mathbb{Z}_n$.

Definition 4. For a deterministic GRA G, let $\lambda_n(G)$ denote the probability that G, when run on an input a chosen uniformly at random from $\mathbb{Z}_n$, is successful in computing the e^{th} root of a.

Lemma 7. *For all $\epsilon > 0$ and $L \in \mathbb{N}$, there exists an algorithm (Algorithm 2) of time complexity $O((\frac{L}{\epsilon})^{5/2}$ that, given access to an L-step deterministic GRA G, with probability $1 - \epsilon$, either outputs a factor of n or an L-step SLP S such that $\nu_n((P^S)^e - x(Q^S)^e) \geq \lambda_n(G) - \frac{\epsilon}{2}$.*

Proof. Consider the tree T^G(see Section 2.2). With each vertex v of T^G, we can associate an SLP, and hence a sequence $(P_1^v, Q_1^v), \ldots, (P_k^v, Q_k^v)$ given by the sequence of operations along the path from the root of T^G to that vertex.

Let $\delta = \frac{\epsilon}{2L}$. We classify the vertices corresponding to equality queries in T^G into two kinds of vertices-*extreme* and *non-extreme* vertices. For a vertex v corresponding to an equality query (i_k, j_k, eq), if

$$\nu_n(P_{i_k}^v \cdot Q_{j_k}^v - P_{j_k}^v \cdot Q_{i_k}^v) \in [\delta, 1 - \delta] \ ,$$

then we call v a *non-extreme* vertex and otherwise we call v an *extreme* vertex.

Let T_{ex}^G be the tree obtained from T^G by truncating the sub-tree rooted at v for all non-extreme vertices v. Therefore all non-extreme vertices present in T_{ex}^G are at the leaves. Also, we can assume, without loss of generality, that the last step of G is not an equality query since that would be of no use. Hence there cannot be an extreme vertex at a leaf vertex of T_{ex}^G.

Let $v^\star(T_{ex}^G)$ be the unique leaf vertex of T_{ex}^G reached by starting from the root and inputting, for all extreme vertices v, the bit 1 to G if $\nu_n(P_{i_k}^v \cdot Q_{j_k}^v - P_{j_k}^v \cdot Q_{i_k}^v) \in [0, \delta)$ and the bit 0 if $\nu_n(P_{i_k}^v \cdot Q_{j_k}^v - P_{j_k}^v \cdot Q_{i_k}^v) \in (1 - \delta, 1]$. Note that $v^\star(T_{ex}^G)$ is either a non-extreme vertex or corresponds to a computation operation. We call the path in T_{ex}^G from the root to the vertex $v^\star(T_{ex}^G)$ the *dominating path* because this is the path that is most likely to be taken if G is run on a random input from $\mathbb{Z}_n$ as we make the most likely choice at each equality test (assuming δ to be small).

Let $M = \lceil \frac{L^{3/2}}{(\epsilon/2)^{5/2}} \rceil$. Consider algorithm 2, as given on the next page.

The intuition is that when executing the GRA for a random element a, either all the equality test one encounters are irrelevant in the sense that the probability that the outcome depends on a is very small, and hence the execution corresponds to an SLP, or the relevant equality test encountered during the execution can be used to factor.

At each equality query, it tries to find a factor of n using the two pairs of polynomials that are compared. If it fails, it outputs the SLP S as the sequence of computation operations along one of the paths from the root to a leaf of T_G. This path corresponds to a unique path in T_{ex}^G. This path is chosen by generating, for each equality query, a uniformly random element in $\mathbb{Z}_n$ and then testing the equality of the two polynomials on this element, and choosing the subsequent vertex based on the result of this equality test. Algorithm 2 is successful with high probability (as shown below) if this path is the dominating path, i.e., if it reaches the vertex $v^\star(T_{ex}^G)$.

Line 7 of the algorithm is a trivial operation and could be avoided but is there so that we do not have to manipulate the indices of the straight line program output by the algorithm.

The SLP S output by Algorithm 2 corresponds to one of the paths from the root to a leaf of T^G which defines a unique path from the root to a leaf of T_{ex}^G.

Let the leaf vertex of T_{ex}^G in which this path terminates be v_S. Note that v_S might not be a leaf vertex of T_G.

Algorithm 2:

```
Input: GRA $G$, $n$
Output: A factor of $n$ or an SLP $S$
1  Initialize $S$ to be the empty sequence;
2  for $k \leftarrow 2$ to $L$ do
3      Get the operation $\{i_k, j_k, \circ_k\}$ from $G$;
4      if $\circ_k \in \{+, -, \cdot, /\}$ then
5          Append $\{i_k, j_k, \circ_k\}$ to $S$;
6      else                                        /* Here, $\circ_k$ is $eq$ */
7          Append $\{k-1, 0, \cdot\}$ to $S$;
8          for $i \leftarrow 1$ to $M$ do
9              Generate a random element $x \in_R \mathbb{Z}_n$;
10             Compute $g = \gcd\left(P_{i_k}^S(x) \cdot Q_{j_k}^S(x) - P_{j_k}^S(x) \cdot Q_{i_k}^S(x), n\right)$;
11             if $g \notin \{1, n\}$ then return $g$;
12         end
13         Generate a random element $x' \in_R \mathbb{Z}_n$;
14         if $P_{i_k}^S(x') \cdot Q_{j_k}^S(x') - P_{j_k}^S(x') \cdot Q_{i_k}^S(x') = 0$ then return the bit 0 to
           $G$ else return the bit 1 to $G$;
15     end
16 end
17 Return $S$;
```

If Algorithm 2 outputs S, then let (P^S, Q^S) denote the pair of polynomials corresponding to S.

Let E be the event that $v_S = v^\star(T_{ex}^G)$, i.e., that the dominating path is found by Algorithm 2. The event E does not occur if there exists an extreme vertex v in the path from the root of T_{ex}^G to v_S corresponding to (i_k, j_k, eq) such that Algorithm 2 inputs 0 to G and $\nu_n(P_{i_k}^v \cdot Q_{j_k}^v - P_{j_k}^v \cdot Q_{i_k}^v) \in [0, \delta)$ or Algorithm 2 inputs 1 to G and $\nu_n(P_{i_k}^v \cdot Q_{j_k}^v - P_{j_k}^v \cdot Q_{i_k}^v) \in (1-\delta, 1]$. Note that this can happen with probability at most δ at each extreme vertex v and there can be at most L such extreme vertices in the path from the root of T_{ex}^G to v_S. Therefore,

$$\mathsf{P}(\mathsf{E}) = 1 - \mathsf{P}(\overline{\mathsf{E}}) \geq 1 - \delta \cdot L = 1 - \tfrac{\epsilon}{2} \ .$$

Now we compute the success probability of the algorithm. There are two possible cases depending on whether $v^\star(T_{ex}^G)$ is a non-extreme vertex or corresponds to a computation operation.

– CASE 1: $v^\star(T_{ex}^G)$ is a non-extreme vertex.
  In this case we show that the factoring algorithm is successful with probability at least $1 - \epsilon$.
  Let F be the event that Algorithm 2 returns a factor of n. We compute $\mathsf{P}(\mathsf{F}|\mathsf{E})$.
  If E holds, then v_S is a non-extreme vertex. Therefore, by Lemma 3, a factor

of n is returned in one test in Step 11 at the equality query corresponding to v_S with probability at least $\delta^{3/2}$. The total number of times step 11 is repeated for this equality query is M. Therefore[7],

$$\mathsf{P}(\mathsf{F}|\mathsf{E}) \geq 1 - (1-\delta^{3/2})^M \geq 1 - \exp(-(\delta^{3/2})M) = 1 - \exp(-\tfrac{2}{\epsilon}) \geq 1 - \tfrac{\epsilon}{2} \ .$$

This implies

$$\mathsf{P}(\mathsf{F}) \geq \mathsf{P}(\mathsf{F}|\mathsf{E}) \cdot \mathsf{P}(\mathsf{E}) \geq (1-\tfrac{\epsilon}{2})^2 \geq 1-\epsilon \ .$$

– CASE 2: $v^\star(T^G_{ex})$ corresponds to a computation operation.
In this case, we show that if the factoring algorithm is not successful, then, with probability $1-\frac{\epsilon}{2}$, we have $\nu_n((P^S)^e - x(Q^S)^e) \geq \lambda_n(G) - \frac{\epsilon}{2}$.
The fraction of inputs $a \in \mathbb{Z}_n$ such that when G is run on a, the corresponding path taken on T^G_{ex} does not terminate in $v^\star(T^G_{ex})$ is at most $\delta \cdot L = \frac{\epsilon}{2}$ (because the number of extreme vertices in any path from root to a leaf is at most L). This implies,

$$\mathsf{P}_{a \in_R \mathbb{Z}_n}\left(P^{v^\star(T^G_{ex})}(a)^e - a \cdot Q^{v^\star(T^G_{ex})}(a)^e \equiv_n 0\right) \ \geq \ \lambda_n(G) - \tfrac{\epsilon}{2} \ .$$

Therefore, if E occurs, then

$$\begin{aligned}\nu_n((P^S)^e - x(Q^S)^e) &= \mathsf{P}_{a \in_R \mathbb{Z}_n}\left(P^{v^\star(T^G_{ex})}(a)^e - a \cdot Q^{v^\star(T^G_{ex})}(a)^e \equiv_n 0\right) \\ &\geq \lambda_n(G) - \tfrac{\epsilon}{2} \ .\end{aligned}$$

Hence,

$$\mathsf{P}\left(\nu_n((P^S)^e - x(Q^S)^e) \geq \lambda_n(G) - \tfrac{\epsilon}{2}\right) \ \geq \ \mathsf{P}(\mathsf{E}) \ \geq \ 1 - \tfrac{\epsilon}{2} \ .$$

Now, we compute the time complexity of Algorithm 2. The loop in steps 8-12 of the algorithm and steps 5,13 and 14, which are the steps in which computation is being performed, are each executed at most L times. Therefore the time complexity of the algorithm is $O(L \cdot M) = O((\frac{L}{\epsilon})^{5/2})$. □

3.2.4 Handling Randomized GRAs

Now we show that a result similar to Lemma 7 also holds for *randomized* GRAs. Recall that a randomized GRA $\mathcal{G}$ is understood as a random variable whose values are deterministic GRAs. Let $\mathsf{P}_\mathcal{G}$ denote the probability distribution of $\mathcal{G}$. $\lambda_n(\mathcal{G})$ is a random variable. Hence, $\mathrm{E}[\lambda_n(\mathcal{G})]$ is the probability of success of $\mathcal{G}$ in computing the e^{th} root of an element chosen uniformly from $\mathbb{Z}_n$.

Lemma 8. *For all $\epsilon' > 0$ and for every L-step randomized GRA $\mathcal{G}$ such that $E[\lambda_n(\mathcal{G})] \geq \mu$, with probability $\frac{\mu}{2} - \epsilon'$, Algorithm 2 from Lemma 7 either outputs a factor of n or an L-step SLP S such that $\nu_n((P^S)^e - x(Q^S)^e) \geq \frac{\mu}{2}$.*

[7] We use the notation $exp(\cdot)$ to denote exponentiation to the natural base in order to avoid confusion with the public exponent e.

Proof. We use the same notation as that used in the proof of Lemma 7. Let $\epsilon = \frac{\epsilon'}{2}$.

The tree $T_{ex}^{\mathcal{G}}$ is a random variable. The vertex $v^\star(T_{ex}^{\mathcal{G}})$ is either an extreme vertex or corresponds to a computation operation. Let $\mathcal{C}$ be the set of deterministic GRAs G such that $v^\star(T_{ex}^{G})$ corresponds to a computation operation. Let C be the event that $\mathcal{G} \in \mathcal{C}$, i.e, that $v^\star(T_{ex}^{\mathcal{G}})$ corresponds to a computation operation. Let $\mathsf{P}(\overline{\mathsf{C}}) = \beta$. Using CASE 1 of the proof of Lemma 7, $\mathsf{P}(\mathsf{F}|\overline{\mathsf{C}}) \geq 1 - \epsilon$. Hence,

$$\mathsf{P}(\mathsf{F} \cap \overline{\mathsf{C}}) = \mathsf{P}(\mathsf{F}|\overline{\mathsf{C}}) \cdot \mathsf{P}(\overline{\mathsf{C}}) \geq (1-\epsilon) \cdot \beta \geq \beta - \epsilon \; . \tag{1}$$

Let D be the event that Algorithm 2 outputs an SLP S and $\nu_n((P^S)^e - x(Q^S)^e) \geq \frac{\mu}{2}$. We define a random variable Y that takes the value $\nu_n((P^S)^e - x(Q^S)^e)$ (where S is the SLP that would be output by Algorithm 2 when run on G if Algorithm 2 does not yield a factor of n) if C occurs, and $Y = 0$ otherwise. Since $Y \geq \frac{\mu}{2}$ implies C occurs and one of F or D occurs

$$\mathsf{P}((\mathsf{F} \cup \mathsf{D}) \cap \mathsf{C}) \geq \mathsf{P}(Y \geq \tfrac{\mu}{2}) \geq \mathrm{E}[Y] - \tfrac{\mu}{2} \; , \tag{2}$$

where the second inequality follows from Lemma 5. Now, we compute $\mathrm{E}[Y]$. Using $\mathrm{E}[\lambda_n(\mathcal{G})] \geq \mu$, we obtain

$$\begin{aligned}\mu \leq \mathrm{E}[\lambda_n(\mathcal{G})] &= \sum_{G} \mathsf{P}_{\mathcal{G}}(G) \cdot \lambda_n(G) \leq \sum_{G \in \mathcal{C}} \mathsf{P}_{\mathcal{G}}(G) \cdot \lambda_n(G) + \sum_{G \notin \mathcal{C}} \mathsf{P}_{\mathcal{G}}(G) \\ &= \sum_{G \in \mathcal{C}} \mathsf{P}_{\mathcal{G}}(G) \cdot \lambda_n(G) + \beta \end{aligned} \tag{3}$$

In the proof of Lemma 7, (CASE 2) we showed that for $G \in \mathcal{C}$, if Algorithm 2, when executed on G, outputs an SLP S, then $\nu_n((P^S)^e - x(Q^S)^e) \geq \lambda_n(G) - \frac{\epsilon}{2}$ with probability at least $1 - \frac{\epsilon}{2}$. Therefore,

$$\begin{aligned}\mathrm{E}[Y] &\geq \sum_{G \in \mathcal{C}} \mathsf{P}_{\mathcal{G}}(G) \cdot (\lambda_n(G) - \tfrac{\epsilon}{2}) \cdot (1 - \tfrac{\epsilon}{2}) \\ &= \sum_{G \in \mathcal{C}} \Big(\mathsf{P}_{\mathcal{G}}(G) \cdot \lambda_n(G) - \tfrac{\epsilon}{2} \cdot \mathsf{P}_{\mathcal{G}}(G) \cdot \lambda_n(G) - \tfrac{\epsilon}{2} \cdot \mathsf{P}_{\mathcal{G}}(G) + \tfrac{\epsilon^2}{4} \cdot \mathsf{P}_{\mathcal{G}}(G) \Big) \\ &\geq \sum_{G \in \mathcal{C}} \mathsf{P}_{\mathcal{G}}(G) \cdot \lambda_n(G) - \tfrac{\epsilon}{2} - \tfrac{\epsilon}{2} + 0 \\ &\geq \mu - \beta - \epsilon, \end{aligned} \tag{4}$$

where the last inequality is obtained by using equation (3). Now, combining equations (1), (2) and (4) we get

$$\begin{aligned}\mathsf{P}(\mathsf{F} \cup \mathsf{D}) &= \mathsf{P}((\mathsf{F} \cup \mathsf{D}) \cap \overline{\mathsf{C}}) + \mathsf{P}((\mathsf{F} \cup \mathsf{D}) \cap \mathsf{C}) \\ &\geq \mathsf{P}(\mathsf{F} \cap \overline{\mathsf{C}}) + \mathsf{P}((\mathsf{F} \cup \mathsf{D}) \cap \mathsf{C}) \\ &\geq \mathsf{P}(\mathsf{F} \cap \overline{\mathsf{C}}) + \mathrm{E}[Y] - \tfrac{\mu}{2} \\ &\geq \beta - \epsilon + \mu - \beta - \epsilon - \tfrac{\mu}{2} = \tfrac{\mu}{2} - 2\epsilon = \tfrac{\mu}{2} - \epsilon' \end{aligned}$$

which completes the proof. □

3.2.5 Putting Things Together

In this section, we complete the proof of Theorem 1.

Proof. Suppose there exists a randomized GRA $\mathcal{G}$ that succeeds in breaking RSA with probability μ_n on $\mathbb{Z}_n$ for some $e > 1$. Then, by Lemma 8, with probability $\mu_n - \epsilon'$, Algorithm 2 either returns a factor of n or an SLP S that succeeds in breaking RSA with probability at least $\frac{\mu_n}{2}$, which can be converted into an algorithm that factors n (Algorithm 1) with probability $\frac{\mu_n}{16(L+log(e))}$.

Since the result is true for all $\epsilon' > 0$, let $\epsilon' = \frac{\mu_n}{2}$. Thus, one execution of Algorithm 2 followed by Algorithm 1 (if needed, i.e., if Algorithm 2 does not return a factor of n) runs in time $O(L^3 + log^3(e) + (\frac{L}{\mu_n})^{5/2})$ and returns a factor of n with probability at least $\frac{\mu_n^2}{32(L+log(e))}$. Therefore, the expected number of times the two algorithms need to be repeated in order to get a factor of n is $\frac{32(L+log(e))}{\mu_n^2}$. Hence the expected time complexity of the factoring algorithm is $O((L^3 + log^3(e) + (\frac{L}{\mu_n})^{5/2}) \cdot (\frac{L+log(e)}{\mu_n^2}))$ which is polynomial in L, $log(e)$ and $\frac{1}{\mu_n}$.

If, for a random N, $\mathcal{G}$ succeeds in breaking RSA with probability μ on $\mathbb{Z}_N$ (i.e., if $\mathrm{E}[\mu_N] = \mu$), then by Lemma 5, $\mathsf{P}(\mu_n \geq \frac{\mu}{2}) \geq \mu - \frac{\mu}{2} = \frac{\mu}{2}$. For all n such that $\mu_n \geq \frac{\mu}{2}$, our factoring algorithm runs in time polynomial in L, $log(e)$ and $\frac{1}{\mu}$, which is polynomial if μ is non-negligible. Therefore the factoring algorithm is a probabilistic polynomial time algorithm that succeeds in factoring N with non-negligible probability. □

4 Conclusions and Open Problems

In this paper we showed that if factoring is hard, then no generic ring algorithm can solve the RSA problem efficiently. Also, if there exists an efficient algorithm that can factor N, then we can compute d such that $e \cdot d \equiv_{\phi(N)} 1$, and then the e^{th} root can be computed by computing the d^{th} power, i.e., *generically* in $O(log(d))$ steps. Thus, this proves, in the generic model, the equivalence of factoring and breaking RSA.

It is interesting to note that all arguments in the paper work not just for the RSA equation $x^e = a$ but for any non-trivial polynomial equation in x and a. More concretely, this means the following. We say that a polynomial $M(x, a)$ is trivial if there exists a rational function g such that $M(g(a), a) \equiv 0$ in $\mathbb{Z}[a]$. Then, if there exists a GRA that, given any non-trivial polynomial $M(x, a)$, computes, with non-negligible probability, an x such that $M(x, a) \equiv_N 0$ for a chosen uniformly at random from $\mathbb{Z}_N$, then there exists an algorithm for factoring N.

There are 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" [10, 1], which allows the adversary to himself choose an exponent $e > 1$. A natural question would be whether factoring is equivalent to solving strong RSA using a GRA. It is not clear whether this statement is true. The proof of Lemma 6, however, does not work for this case because here e will depend on the input a. As a result, in the proof of Lemma 6, $f(a)$ is not a polynomial in a (because the exponent is not independent of a).

References

1. N. Baric and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *EUROCRYPT 1997*, volume 1233 of *Lecture Notes in Computer Science*, pages 480-494.
2. D. Boneh and R. Lipton. Black box fields and their application to cryptography. In *CRYPTO 1996*, volume 1109 of *Lecture Notes in Computer Science*, pages 283-297.
3. D. Boneh and R. Venkatesan. Breaking RSA may be easier than factoring. In *EUROCRYPT 1998*, volume 1403 of *Lecture Notes in Computer Science*, pages 59-71.
4. D. R. L. Brown. Breaking RSA may be as difficult as factoring. In *Cryptology ePrint Archive*, Report 205/380, 2006.
5. L. Childs. *A concrete introduction to higher algebra.* New York: Springer-Verlag, 1992.
6. R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO 1998*, volume 1462 of *Lecture Notes in Computer Science*, pages 13-25.
7. 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, 1999.
8. I. Damgård and M. Koprowski. Generic lower bounds for root extraction and signature schemes in general groups. In *EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 256-271.
9. W. Diffie and M. Hellman. New directions in cryptography. In *IEEE Transactions on Information Theory*, volume 22, no. 6, pages 644-654, 1976.
10. E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *CRYPTO 1997*, volume 1294 of *Lecture Notes in Computer Science*, pages 16-30.
11. T. Jager. Generic group algorithms. Master's thesis, Ruhr Universität Bochum, 2007.
12. A. Joux, D. Naccache and E. Thomé. When e-th roots become easier than factoring. In *ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 13-28.
13. 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.
14. R. Lidl and H. Niederreiter. In *Introduction to finite fields and their applications.* Cambridge University Press, 1994.
15. U. Maurer. Towards proving the equivalence of breaking the Diffie-Hellman protocol and computing discrete logarithms. In *CRYPTO 1994*, volume 839 of *Lecture Notes in Computer Science*, pages 271-281.

16. U. Maurer. Fast generation of prime numbers and secure public-key cryptographic parameters. In *Journal of Cryptology*, volume 8, no. 3, pages 123-155, 1995.
17. U. Maurer. Abstract models of computation in cryptography. In *Cryptography and Coding 2005*, volume 3796 of *Lecture Notes in Computer Science*, pages 1-12.
18. U. Maurer and S. Wolf. The relationship between breaking the Diffie-Hellman protocol and computing discrete logarithms. In *SIAM Journal of Computing*, volume 28, no. 5, pages 1689-1721, 1999.
19. D. Micciancio. The RSA group is pseudo-free. In *EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 387-403.
20. V. I. Nechaev. Complexity of a deterministic algorithm for the discrete logarithm. In *Mathematical Notes*, volume 55, no. 2, pages 91-101, 1994.
21. R.L. Rivest. On the notion of pseudo-free groups. In *TCC 2004*, volume 2951 of *Lecture Notes in Computer Science*, pages 505-521.
22. R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. In *Communications of the ACM*, volume 21, pages 120-126, 1978.
23. V. Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT 1997* volume 1233 of *Lecture Notes in Computer Science*, pages 256-266.