Breaking RSA Generically Is Equivalent to Factoring

Divesh Aggarwal and Ueli Maurer

***Abstract*— Let N be a random variable distributed according to some appropriate distribution over the set of products of two primes, such that factoring N is believed to be hard. The RSA assumption states that, given an a chosen uniformly at random from $\mathbb{Z}_N$ and an $e \in \mathbb{N} \setminus \{1\}$ such that $\gcd(e, \phi(N)) = 1$, it is computationally hard to find an $x \in \mathbb{Z}_N$ such that $x^e - a \equiv 0 \pmod N$. When complexity-theoretic (relative) lower bounds for certain cryptographic problems in a general model of computation seem to elude discovery, a common practice in cryptography is to give proofs of computational security in meaningful restricted models of computation. An example of such a restricted model that is interesting in cryptography is the generic group model that has been used for proving lower bounds for the discrete logarithm problem and other related problems. A generic model captures that an algorithm does not exploit the bit representation of the elements other than for testing equality. In this paper, we prove that the problem of factoring N can be efficiently reduced to solving the RSA problem on $\mathbb{Z}_N$ in the generic ring model of computation, where an algorithm can perform ring operations, inverse ring operations, and test equality. This provides evidence toward the soundness of the RSA encryption and digital signature scheme, in particular showing that under the factoring assumption, they are not vulnerable to certain kinds of cryptanalytic attacks.**

***Index Terms*— Generic algorithms, reductions, factoring, RSA.**

I. Introduction

THE two most fundamental reduction problems in number-theoretic cryptography are to prove or disprove that breaking the RSA system [18] is as hard as factoring integers and that breaking the Diffie-Hellman protocol [7] is as hard as computing discrete logarithms. While the second problem has been solved to a large extent [3], [13], [16], not much is known about the first for general models of computation. In this paper, we show that breaking RSA generically is as hard as factoring the RSA modulus.

A. RSA and Factoring

In the plain RSA public key encryption scheme [18], the message $x \in \mathbb{Z}_n$, where n is a product of two primes, is encrypted as $x^e \pmod n$, where $e > 1$ is an integer. The security of this scheme relies on the assumption that, given a, it is hard to find x such that $x^e - a \equiv 0 \pmod n$.

Before we state the RSA assumption formally, we introduce the following notation. Let κ denote the security parameter. Let $(N_\kappa, E_\kappa) \in \mathbb{N} \times \mathbb{N}$ be a pair of jointly distributed random variables such that N_κ is a κ-bit integer that is a product of two primes, $E_\kappa > 1$ is such that $\gcd(E_\kappa, \phi(N_\kappa)) = 1$, and $\log(E_\kappa)$ is bounded from above by a polynomial in κ. Our main result is a polynomial-time reduction that works for every fixed $n = pq$ and every fixed e with $\gcd(e, \phi(n)) = 1$. The reduction therefore works for any distribution (N_κ, E_κ). Stating the results for fixed n and e yields a strictly stronger results than only stating it for distributions for which a certain assumption holds. When we consider N_κ (for example in the factoring assumption below), its distribution is understood to be the corresponding marginal distribution.

We state the two assumptions whose relations we study in this paper:

- *Factoring Assumption for $(N_\kappa)_{\kappa \in \mathbb{N}}$:* For every probabilistic polynomial-time algorithm that takes as input N_κ, the probability of finding a non-trivial factor of N_κ is negligible.[1]
- *RSA Assumption for $(N_\kappa, E_\kappa)_{\kappa \in \mathbb{N}}$:* For every probabilistic polynomial-time algorithm that takes as input the pair (N_κ, E_κ) and an element a chosen uniformly at random from $\mathbb{Z}_{N_\kappa}$, the probability that it computes $x \in \mathbb{Z}_{N_\kappa}$ such that $x^{E_\kappa} - a \equiv 0 \pmod{N_\kappa}$ is negligible.

The probability of success of the algorithms in the above definitions is taken over N_κ, E_κ, a, and the randomness of the algorithm.

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 [19], based on the work of Nechaev [17], introduced the concept of generic algorithms which are algorithms that do not exploit any property of the representation of the

Manuscript received April 28, 2013; revised July 28, 2015; accepted February 28, 2016. Date of publication July 27, 2016; date of current version October 18, 2016.

D. Aggarwal is with the Centre of Quantum Technologies, Department of Computer Science, National University of Singapore, Singapore, 117543 (e-mail: divesh@comp.nus.edu.sg).

U. Maurer is with the Department of Computer Science, ETH Zürich, Zürich CH-8092, Switzerland (e-mail: maurer@inf.ethz.ch).

Communicated by R. Cramer, Associate Editor for Complexity and Cryptography.

Digital Object Identifier 10.1109/TIT.2016.2594197

[1] A function $f(\kappa)$ is defined to be negligible if it vanishes faster than the inverse of any polynomial (in κ), i.e., if for every $c > 0$ there exists $k_0 \in \mathbb{N}$ such that for all $\kappa > k_0$, $|f(\kappa)| < \frac{1}{\kappa^c}$. The terms polynomial-time and negligible in the definitions of these assumptions are with respect to the security parameter κ.

elements. They proved lower bounds on the complexity of computing discrete logarithms in cyclic groups in the context of generic algorithms. Maurer [14] provided a simpler and more general model for analyzing representation-independent algorithms. In this work, we consider the assumption that breaking RSA is hard in the generic ring model, which we formally define in Section III after introducing generic ring algorithms.

B. The Generic Model of Computation

We give a brief description of the model of [14]. The model is characterized by a black-box **B** which can store values from a certain set $\mathcal{Z}$ 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 **B** allows two types of operations:

- *Computation Operations.* For a set Π of operations of some arities on $\mathcal{Z}$, 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. **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 $\mathcal{Z}$, 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.

An algorithm in this model is characterized by its interactions with the black box **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 **B**.

For this paper, the set $\mathcal{Z}$ is $\mathbb{Z}_n$, where n is an integer that is a product of two primes. Moreover, $\ell = 1$, and V_0 is always set to be the unit element 1 of $\mathbb{Z}_n$, and V_1 is the value a. 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 $\Sigma = \{eq\}$, where eq stands for equality queries. 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.

Some results like [11] in the literature are restricted in that they exclude the inverse operations, but since these operations are easy[2] to perform in $\mathbb{Z}_n$, they should be included as otherwise the results are of relatively limited interest.

[2]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 $\mathbb{Z}_n$.

C. Discussion and Relevance of the Generic Model of Computation

Since Shoup's result [19] on proving lower bounds for computing discrete logarithms and some related problems in the generic group model (where the only allowed operations are the group operations and equality queries), it has been a well accepted approach to give proofs of hardness of problems relevant in cryptography in the generic group/ring model.

Three kinds of problems can be considered in the generic group/ring model: *extraction problems*, *computation problems* and *decision problems*.

Extraction problems are problems where the task of the algorithm is to extract the input x from the black-box. An example of this is the discrete logarithm problem, as has been explained in [14]. Other examples of the extraction problem in related models have been considered in [2], [3], and [15].

Decision problems are problems of computing some predicate (function) of the input which evaluates to either 0 or 1. In this case, a generic algorithm tries to guess the bit based on its interactions with the blackbox. An example of this in the generic group model is the Decisional Diffie Hellman problem.

The only possible way to guess the output of a decision problem is based on the result of the relation queries. In the generic ring model described in Section 1.2, the only relation queries allowed are equality queries. Thus, for the ring $\mathbb{Z}_n$, as we show in Lemma 5, if the input is chosen uniformly at random from $\mathbb{Z}_n$, then if we can obtain any non-trivial information from an equality query with non-negligible probability, then we can use this to factor n. Therefore, under the assumption that factoring n is hard, there does not exist any GRA for solving any decision problem on the ring $\mathbb{Z}_n$ for an input chosen uniformly at random from $\mathbb{Z}_n$. Thus, even problems as simple as computing the least significant bit of a random input in $\mathbb{Z}_n$ is hard with respect to generic ring algorithms. In particular, computing the Jacobi symbol is an example of a problem that is easy to solve in general, but is hard in the generic ring model as has also been pointed out in [9].

This may at first seem to suggest that a result showing the hardness of a certain problem with respect to generic ring algorithms is of no significance. However, computation problems still remain an interesting class of problems in the generic ring model of computation. The computation problems are problems where the algorithm is required to compute a function of the input that results in a set of elements of the underlying group/ring structure. The algorithm is successful if it is able to compute this set of elements inside the black-box using the allowed operations and relation queries. An example of a computation problem in the generic group model is the Computational Diffie Hellman problem.

There exist (maybe inefficient) generic algorithms for solving almost all computation problems. In particular, consider the RSA problem in the ring of integers modulo $n = pq$. There is a trivial GRA that solves the RSA problem by computing x^e for different values of x from $2, 3, \ldots$ until an x is obtained such that $x^e = a$. Thus it is interesting to obtain a result that shows the impossibility of getting an efficient GRA under a plausible assumption like the assumption that factoring is hard.

A result in the generic model makes more sense than, for example, results in the monotone circuit model in complexity theory where one is allowed only AND and OR but no NOT gates, since there are instances where generic algorithms are the best known algorithms e.g., discrete logarithms problem over elliptic curves, while such instances do not exist for the monotone circuit model. In particular, a result in this model rules out certain classes of cryptanalytic attacks for breaking the corresponding cryptosystem.

Motivated by the fact that decision problems like the Jacobi symbol are hard in the generic model and easy in general, one might want to consider a more general model than the generic ring model of computation where one is, for example, given oracle access to the Jacobi symbol. This can be modeled nicely by allowing another relation query corresponding to whether the Jacobi symbol is 0 or 1.

D. Comparison With Related Work

Research on the relation between RSA and factoring comes in two flavours. There have been results giving evidence against (e.g. [4], [10]) and in favour of (e.g. [5], [11]) the assumption that breaking RSA is equivalent to factoring.

Boneh and Venkatesan [4] 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. [10] showed that given access to an oracle that computes e-th roots of numbers of the form $x+c$ for a fixed c and any x, computing arbitrary e-th roots modulo n is easier than factoring n.

Brown [5] 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 [11] generalized the result of [5] 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 [6]. 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. In fact, Theorem 1 appears to be the most general statement about the equivalence of breaking RSA in a generic ring model and factoring.

- First, compared to [5] and [11], 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 [5], [6], and [11], 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.[3] Actually, as has been pointed out in [5], computing the multiplicative inverse of a random element in $\mathbb{Z}_n$ generically is hard if $\Pi = \{+, -, \cdot\}$.

The problem we solve has been stated as an open problem in [5] and [6].

II. Preliminaries

A. Straight-Line Programs and Generic Ring Algorithms

In this section we define GRAs and SLPs (as a special case) as (syntactic) mathematical objects. Then we define the semantics of the GRA (SLP) in terms of what is computed at each step, and finally we define the partial function $\mathbb{Z}_n \to \mathbb{Z}_n$ computed by the GRA (SLP) when it is run for the ring $\mathbb{Z}_n$.

An SLP (for rings) is a deterministic algorithm that performs a sequence of ring operations. Thus an SLP corresponds to $\Pi = \{+, -, \cdot, /\}$ and $\Sigma = \{\}$ in the model of [14]. More precisely, an SLP is a sequence of operations, where the k-th operation is of the form $(i, j, \circ)$ for $0 \leq i, j < k$ and $\circ \in \{+, -, \cdot, /\}$. The result of the k-th step is defined as the result of applying $\circ$ to the i-th and the j-th intermediate results. To be compatible with the definition of a GRA, we define an SLP as a labeled path (a graph) rather than a sequence:

Definition 1: An L-step SLP *S is a partially labeled path $v_0, \ldots, v_L$ where v_0, v_1 are unlabeled and v_k for $k \in [2, L]$ carries a label of the form $(v_i, v_j, \circ)$ for $0 \leq i, j < k$ and $\circ \in \{+, -, \cdot, /\}$.*

A deterministic generic ring algorithm (GRA) is a generalized SLP that also allows equality queries, i.e., it corresponds to $\Pi = \{+, -, \cdot, /\}$ and $\Sigma = \{=\}$. Hence a GRA corresponds to a labeled tree, where branches in the tree correspond to equality queries, where the left [right] child of a branching vertex corresponds to the equality test being [not] satisfied. An equality query must specify which two intermediate results (i.e., which vertices in the graph) are to be compared. Formally:

Definition 2: An L-step deterministic GRA *G is a depth-L partially vertex-labeled and edge-labeled binary tree where the root vertex and its child are unlabeled and each has one outgoing edge, and the remaining tree has two kinds of vertices:* branching vertices *and* non-branching vertices*. A branching vertex v has two outgoing edges labeled* 0 *(for left edge) and* 1 *(for right edge), and v carries a label of the form (u, w), where u, w are some non-branching vertices in the path from the root to v. A non-branching vertex v has one outgoing edge and v has a label of the form $(u, w, \circ)$, where u, w are some non-branching vertices in the path from the root to v, and $\circ \in \{+, -, \cdot, /\}$.*

Note that an SLP can be seen as a special case of a GRA with no branching vertices. Next, we define a randomized GRA as a further generalization of a deterministic GRA, where the choice of the operation at each step is randomized.

[3] In [5], the author has mentioned and given justification for the fact that most results of his paper will extend to SLPs with division.

Definition 3: A randomized GRA *is a random variable whose values are deterministic GRAs.*

In our main reduction (Theorem 2), we will talk about an algorithm having access to a GRA $\mathcal{G}$. By this, we mean that the algorithm has a pointer to the root of the tree (denoted $\mathcal{G}.root$), and each vertex v of the tree has a pointer to its left and the right child (denoted $v.left$ and $v.right$, respectively). If the vertex is a non-branching vertex, and hence has only one child, then $v.right = \perp$.

A GRA (or SLP) without division operation can naturally be interpreted as computing a polynomial in $\mathbb{Z}[x]$ at each non-branching vertex, where by definition the root and the child of the root are labeled by polynimials 1 and x, respectively. (Here x stands for the indeterminate of the polynomial. For a concrete argument a in the ring, x will be thought of being set to a.) Note that these polynomials are defined over $\mathbb{Z}[x]$, no matter to which particular ring R (e.g. $R = \mathbb{Z}_n$) it is thought of as being applied. If the GRA contains division operations, then each non-branching vertex corresponds naturally to a pair of polynomials in $\mathbb{Z}[x]$, the numerator and the denominator polynomial. Note that we do not yet interpret this pair as a (rational) function and hence the possible problem of a division by 0 does not arise at this point.

Definition 4: For a GRA G (or SLP S) and non-branching vertex v, the pair $(P_v^G(x), Q_v^G(x))$ of polynomials in $\mathbb{Z}[x]$ associated with v *is defined inductively, as follows:*

1) *The root has associated the pair $(1, 1)$, and the child of the root the pair $(x, 1)$.*
2) *For each non-branching vertex v, labeled with operation $(u, w, \circ)$, we have*
   - *If $\circ$ is $+$, then*
   $$P_v^G(x) = P_u^G(x) \cdot Q_w^G(x) + P_w^G(x) \cdot Q_u^G(x),$$
   $$Q_v^G(x) = Q_u^G(x) \cdot Q_w^G(x).$$
   - *If $\circ$ is $-$, then*
   $$P_v^G(x) = P_u^G(x) \cdot Q_w^G(x) - P_w^G(x) \cdot Q_u^G(x),$$
   $$Q_v^G(x) = Q_u^G(x) \cdot Q_w^G(x).$$
   - *If $\circ$ is $\cdot$, then*
   $$P_v^G(x) = P_u^G(x) \cdot P_w^G(x),$$
   $$Q_v^G(x) = Q_u^G(x) \cdot Q_w^G(x).$$
   - *If $\circ$ is $/$, then*
   $$P_v^G(x) = P_u^G(x) \cdot Q_w^G(x),$$
   $$Q_v^G(x) = Q_u^G(x) \cdot P_w^G(x).$$

If the GRA is an SLP S, then the final pair of polynomials (i.e., one that corresponding to the last vertex) is unique and will be denoted as (P^S, Q^S).

It follows from the definitions that any SLP S of length L can be "converted" into an SLP S' of length at most $4L$ that computes both $(P^S, 1)$ and $(Q^S, 1)$. A result similar to this but with a factor of 6 instead of 4 has been proven independently in [8].

Lemma 1: For any L-step SLP S, there exists a $4L$-step SLP S' and some indices $r, t \leq 4L$ such that $(P_{v_r}^{S'}, Q_{v_r}^{S'}) = (P^S, 1)$ and $(P_{v_t}^{S'}, Q_{v_t}^{S'}) = (Q^S, 1)$.

For a fixed (partially invertible) ring R, one can associate to every vertex v of a GRA G the partial function f_v^G computed at that vertex, as explained in the definition below. Moreover, executing a GRA for a given ring R and a given argument $a \in R$ means to compute along a path, starting at the root, where each equality test (comparing the values computed at two previous vertices) determines which branch is taken in the corresponding branching vertex. Such a computation ends in a leaf ℓ of the tree, where the leaf ℓ that is reached depends on the argument a. For argument a, the value computed at the reached leaf is defined as the function value $f^G(a)$ computed by the GRA. More formally:

Definition 5: For each non-branching vertex v in a GRA G with corresponding pair of polynomials $(P_v^G(x), Q_v^G(x))$, we associate the function

$$f_v^G : R \to R \cup \{\perp\} : a \mapsto P_v^G(a) \cdot (Q_v^G(a))^{-1},$$

where the function is undefined if $Q_v^G(a)$ is not invertible in R, which is denoted as $f_v^G(a) = \perp$, and where $P_v^G(a)$ and $Q_v^G(a)$ are evaluated over R. Moreover, for an argument $a \in R$, the computation path $v_0, v_1, \ldots, v_\ell$ from the root $v_0(a)$ to a leaf $v_\ell(a) =: \Lambda(a)$ is defined by taking, for each equality test of the form (u, w), the edge labeled 0 if $f_u^G(a) = f_w^G(a)$, and the edge labeled 1 if $f_u^G(a) \neq f_w^G(a)$. The partial function f^G computed by G is defined as

$$f^G : R \to R \cup \{\perp\} : a \mapsto f_{\Lambda(a)}^G.$$

We only consider rings of the form $R = \mathbb{Z}_n$, and for this case the function computed by GRA G will be denoted as $f^{G,n} : \mathbb{Z}_n \to \mathbb{Z}_n \cup \{\perp\}$.

B. Mathematical Preliminaries

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

For any event E, we denote the probability of E by $\Pr(\mathsf{E})$. For any $n \in \mathbb{N}$, $\mathbb{Z}_n[x]$ denotes the ring of polynomials in x with coefficients in $\mathbb{Z}_n$. For $h(x) \in \mathbb{Z}_n[x]$, $\mathbb{Z}_n[x]/(h(x))$ denotes the quotient ring $\mathbb{Z}_n[x]$ by a principal ideal generated by $h(x)$.

Definition 6: For any integer $n \in \mathbb{N}$ and any partial function $f : \mathbb{Z}_n \mapsto \mathbb{Z}_n \cup \{\perp\}$, we define the following.

- Let $\eta_n(f)$ denote the fraction of elements a in $\mathbb{Z}_n$ such that $f(a)$ has a non-trivial greatest common divisor with n. Formally, $\eta_n(f)$ is equal to
$$\frac{\left|\left\{a \in \mathbb{Z}_n | f(a) \neq \perp \ \wedge \ \gcd(f(a), n) \notin \{1, n\}\right\}\right|}{n}.$$
- Let $\nu_n(f)$ denote the fraction of roots of f in $\mathbb{Z}_n$, i.e.,
$$\nu_n(f) := \frac{\left|\left\{a \in \mathbb{Z}_n | f(a) = 0\right\}\right|}{n}.$$
- For any deterministic GRA G, and any function $g : \mathbb{Z}_n \mapsto \mathbb{Z}_n$, let $\lambda_n(G, g)$ be the fraction of $a \in \mathbb{Z}_n$ on which G computes g, when executed on $\mathbb{Z}_n$. Formally,
$$\lambda_n(G, g) := \nu_n\left(f^{G,n}(x) - g(x)\right),$$

where for any $a \in \mathbb{Z}_n$, $f^{G,n}(a) - g(a) = \perp$ if $f^{G,n}(a) = \perp$.

For $n \in \mathbb{N}$ and $e > 1$ such that $\gcd(e, \phi(n)) = 1$, we denote by $x^{1/e} : \mathbb{Z}_n \mapsto \mathbb{Z}_n$, the function that maps $a \in \mathbb{Z}_n$ to $b \in \mathbb{Z}_n$ such that $b^e = a$.

We prove a few results that we will need later. These are similar to what was used in [5] and [11].

Lemma 2: Let $n = pq$ be a product of two primes. For any $P(x) \in \mathbb{Z}_n[x]$ and for any $\delta > 0$, 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

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

□

Lemma 3: Let p be a prime and d be a positive integer. The fraction of monic polynomials $f(x) \in \mathbb{Z}_p[x]$ of degree d that are irreducible in $\mathbb{Z}_p[x]$ is at least $\frac{1}{2d}$ and the fraction that has a root in $\mathbb{Z}_p$ is at least $1/2$.

Proof: From the distribution theorem of monic polynomials (see [12]) it follows that the number of monic irreducible polynomials of degree d over F_p is at least $\frac{p^d}{2d}$, which implies the first claim.

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 $\frac{p^d}{2}$, thus showing the second claim.[4] □

III. The Main Result

Now, we formally define the Generic RSA Assumption.

- *Generic RSA Assumption for $(N_\kappa, E_\kappa)_{\kappa \in \mathbb{N}}$:* For every poly($\kappa$)-step randomized GRA, the probability that given an element a chosen uniformly at random from $\mathbb{Z}_{N_\kappa}$, it computes $x \in \mathbb{Z}_{N_\kappa}$ such that $x^{E_\kappa} - a \equiv 0 \pmod{N_\kappa}$ is negligible, where the probability is taken over the randomness of N_κ, E_κ, a and that of the algorithm.

The following is the main result of this paper.

[4]Note that, by a careful analysis, it is possible to prove a better lower bound on the fraction of polynomials that have a root in $\mathbb{Z}_p$ but a lower bound of $1/2$ is sufficient for our purpose.

Theorem 1: For all $\kappa > 0$, let (N_κ, E_κ) be a pair of jointly distributed random variables such that N_κ is a κ-bit integer that is a product of two primes and $E_\kappa \in \mathbb{N}$ is such that $\gcd(E_\kappa, \phi(N_\kappa)) = 1$. If the factoring assumption for $(N_\kappa)_{\kappa \in \mathbb{N}}$ holds, then the generic RSA assumption holds for $(N_\kappa, E_\kappa)_{\kappa \in \mathbb{N}}$.

We will in fact prove the following stronger result.

Theorem 2: For all $\mu \in (0,1)$, there exists a probabilistic algorithm $\mathcal{A}$ that takes as input an integer $n = pq$ that is a product of two primes, an integer $e > 1$ such that $\gcd(e, \phi(n)) = 1$, and access to an L-step randomized GRA $\mathcal{G}$, runs in time polynomial in $\log n$, $\log e$, L, and $\frac{1}{\mu}$, such that whenever

$$\mathsf{E}_{\mathcal{G}}[\lambda_n(\mathcal{G}, x^{1/e})] \geq \mu,$$

then $\mathcal{A}$ computes a factor of n with probability at least $\frac{1}{2}$, where the probability is taken over the randomness of the algorithm.

We will prove Theorem 2 in Section IV, and here we show that Theorem 1 is an immediate corollary of Theorem 2.

Proof of Theorem 1: Assume to the contrary that the generic RSA assumption does not hold for $(N_\kappa, E_\kappa)_{\kappa \in \mathbb{N}}$. This means that there exists c such that for infinitely many κ, there is a generic ring algorithm that takes as input a uniformly random in $\mathbb{Z}_{N_\kappa}$ and computes $b \in \mathbb{Z}_{N_\kappa}$ such that $b^{E_\kappa} = a$ with probability at least $1/\kappa^c$, where the probability is taken over the randomness of N_κ, E_κ, a and that of the algorithm. This implies that for each such κ, with probability $\frac{1}{2\kappa^c}$ over the choice of $N_\kappa, E_\kappa = (n, e)$, we have

$$\mathsf{E}_{\mathcal{G}}[\lambda_n(\mathcal{G}, x^{1/e})] \geq \frac{1}{2\kappa^c}.$$

If we set $\mu = \frac{1}{2\kappa^c}$, then the algorithm of Theorem 2 runs in polynomial time and computes a factor of n with probability $1/2$, which in turn implies that the algorithm computes a factor of N_κ with probability at least $\frac{1}{2\kappa^c} \cdot \frac{1}{2} = \frac{1}{4\kappa^c}$, where the probability is taken over the randomness of N_κ and that of the algorithm. This contradicts the factoring assumption for N_κ. □

IV. Proof of Theorem 2

In this section, the success probability of any factoring algorithm is always over the randomness of the algorithm, and n and e are fixed.

Throughout this section, we write $f(x) \equiv 0 \pmod{n}$ if f is the constant 0-function modulo n, and for a (possibly partial) function f we write $f(x) \not\equiv 0 \pmod{n}$ otherwise.

In Section IV-A, we show that an SLP that computes e-th roots modulo n with non-negligible probability can be used to factor n. Then, in Section IV-B, we show that from a deterministic GRA that computes e-th roots, we can either obtain an SLP that computes e-th roots, or directly obtain a factor of n. In Section IV-C, we combine the results of Section IV-A and IV-B to show that a randomized GRA that computes e-th roots can be used to factor n.

Algorithm 1 Factoring Algorithm

Input: n, SLP S
Output: A factor of n

1 Choose a monic polynomial $h(x)$ uniformly at random from all monic polynomials of degree L 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 the instructions of SLP S;
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 $\gcd(n, H(h(x), h'(x)))$;

A. The Proof for Straight-Line Programs

We first give an algorithm that factors n given access to an SLP that computes a non-trivial polynomial that is 0 modulo n.

For $b(x), c(x) \in \mathbb{Z}_n[x]$, 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 $b(x), c(x) \in \mathbb{Z}_n[x]$. If $\deg(\gcd_p(b(x), c(x))) \neq \deg(\gcd_q(b(x), c(x)))$, then Euclid's algorithm on $\mathbb{Z}_n[x]$[5] with input $b(x)$ and $c(x)$ yields a non-trivial non-invertible element of $\mathbb{Z}_n$.

We denote by $H(b(x), c(x))$ the non-trivial non-invertible element output when Euclid's algorithm is executed on $Z_n[x]$ with input $b(x)$ and $c(x)$.

Lemma 4: There exists a randomized algorithm (Algorithm 1) that takes as input $n = pq$, where $p, q > 3$ are primes, $L \in \mathbb{N}$, and an L-step SLP S, runs in time $O(L^3 \log^2 n)$, and does the following. If $(P^S(x), Q^S(x)) = (f(x), 1)$, and $f(x) \not\equiv 0 \pmod n$, then Algorithm 1 returns a factor of n with probability at least $\frac{\nu_n(f)}{8L}$, where the probability is over the randomness of the algorithm.

Proof: Consider Algorithm 1. 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. Without loss of generality, we assume that $f(x) \not\equiv 0 \pmod q$. By Lemma 3, the probability that $h(x)$ is irreducible modulo q and has a root modulo p is at least $\frac{1}{2L} \cdot \frac{1}{2} = \frac{1}{4L}$. We assume that this is the case for the rest of the proof and all subsequent probabilities are computed conditioned on this event.

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

- CASE 1: $(x - s)^2$ divides $h(x)$ in $\mathbb{Z}_p[x]$.
  This implies that $(x - s)$ divides $\gcd_p(h(x), h'(x))$. However, since $h(x)$ is irreducible in $\mathbb{Z}_q[x]$, the degree of $\gcd_q(h(x), h'(x))$ is 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: $(x - s)^2$ does not divide $h(x)$ in $\mathbb{Z}_p[x]$.
  Let $h(x) = h_1(x) \cdot (x - s)$ in $\mathbb{Z}_p[x]$. Then:

$$\begin{aligned}\mathbb{Z}_n[x]/h(x) &\cong \mathbb{Z}_p[x]/h(x) \times \mathbb{Z}_q[x]/h(x) \\ &\cong \mathbb{Z}_p[x]/(x - s) \times \mathbb{Z}_p[x]/h_1(x) \times \mathbb{F}_{q^L}\end{aligned}$$

because $\mathbb{Z}_q[x]/(h(x)) \cong \mathbb{F}_{q^L}$ (the finite field containing q^L elements) as $h(x)$ is irreducible in $\mathbb{Z}_q[x]$ by our assumption.
We identify the elements in the quotient ring by the polynomial representing the corresponding element in the quotient ring. Under this isomorphism, let $r(x)$ map to the triple

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

and let $z(x)$ map 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 that the probability that $z(s)$ is equal to 0 modulo p is

$$\begin{aligned}&= \Pr\left(f(r(s)) \equiv 0 \pmod p\right) \\ &\geq \Pr\left(f(r(s)) \equiv 0 \pmod n\right) \\ &= \nu_n(f).\end{aligned}$$

Therefore, with probability at least $\nu_n(f)$, $(x - s)$ 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 $\mathbb{Z}_n[x]/(h(x))$, $r_q(x)$ is uniformly random in $\mathbb{Z}_q[x]/(h(x)) \cong \mathbb{F}_{q^L}$. A non-zero polynomial over a finite field can have at most as many roots as the degree of the polynomial. Therefore, for random x,

$$\begin{aligned}\Pr\left(z_q(x) = 0\right) &= \Pr\left(f(r_q(x)) = 0\right) \\ &\leq \frac{\deg(f)}{q^L} \\ &\leq \frac{2^L}{q^L} \\ &\leq \frac{1}{2},\end{aligned}$$

for $q > 3$. We use the fact that $\deg(f) \leq 2^L$. This is because, at each step of the SLP which is either an addition, subtraction, or a multiplication operation of two previously computed polynomials, the degree is bounded by the sum of the degrees of the two polynomials. Thus, by induction, the degree is at most 2^L after L steps of the SLP. Hence, $\Pr(z_q(x) \not\equiv 0) \geq \frac{1}{2}$. The condition $z_q(x) \not\equiv 0$ 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 $\frac{1}{4L} \cdot \nu_n(f) \cdot \frac{1}{2} = \frac{\nu_n(f)}{8L}$.

[5]Note that $\mathbb{Z}_n[x]$ is not a Euclidean domain.

Now we compute the time complexity of one run of the loop. Generating random $h(x)$ and $r(x)$ and computing the derivative requires $O(L\log^2 n)$ operations in $\mathbb{Z}_n$. Each operation in $\mathbb{Z}_n[x]/(h(x))$ can be implemented by at most $L^2\log^2 n$ operations in $\mathbb{Z}_n$. The function $f(r(x)) = z(x)$ can be computed in time $O(L^2\log^2 n \cdot L) = O(L^3\log^2 n)$. Euclid's algorithm on $z(x)$ and $h(x)$ and on $h(x)$ and $h'(x)$ can be performed by $O(L^2\log^2 n)$ operations. Thus, the running time of the algorithm is $O(L^3\log^2 n)$. □

Corollary 1: There exists an algorithm that takes as input $n = pq$, where $p, q > 3$ are primes, an integer $e > 1$ such that $\gcd(e, \phi(n)) = 1$, $L \in \mathbb{N}$, and an L-step SLP S, runs in time $O((L^3 + \log^3 e)\log^2 n)$, and returns a factor of n with probability at least $\frac{\lambda_n(S, x^{1/e})}{32(L+\log e)}$.

Proof: Consider the polynomial $f(x) = (P^S(x))^e - x \cdot (Q^S(x))^e$. Note that for any $a \in \mathbb{Z}_n$, $(f^{S,n}(a))^e - a = 0$ if and only if $f(a) = 0$ and $Q^S(a) \neq 0$. Thus, $\lambda_n(S, x^{1/e}) = \nu_n\left((f^{S,n}(x))^e - x\right) \leq \nu_n(f)$. In order to apply Lemma 4, we need an SLP that computes f and we need to argue that $f(x) \not\equiv 0 \pmod n$.

From Lemma 1 we get that from any L-step SLP S, we can get a $4L$-step SLP using only operations $\{+, -, \cdot\}$ that computes $(P^S, 1)$ and $(Q^S, 1)$. Hence we can obtain a $(4L + 4\log e)$-step SLP that computes the pair $((P^S(x))^e - (Q^S(x))^e \cdot x, 1)$.

In order to argue that $f(x) \not\equiv 0 \pmod n$, without loss of generality, we assume that $Q^S(x) \not\equiv 0 \pmod n$, since otherwise $\lambda_n\left(S, x^{1/e}\right) = 0$, and the desired result is vacuously true. Let the leading terms of $P^S(x)$ and $Q^S(x)$ be $a_0x^{d_0}$ and $a_1x^{d_1}$, respectively, where $a_1 \not\equiv 0 \pmod n$. Since, $e > 1$, we cannot have $d_0 \cdot e = d_1 \cdot e + 1$ for any integers d_0, d_1, and hence the leading term of $f(x)$ is non-zero. □

B. From Deterministic GRAs to SLPs

In this section we give an algorithm that, given access to a deterministic GRA that computes e-th roots, outputs either a factor of n or an SLP that computes e-th roots.

Lemma 5: For all $\epsilon > 0$ and any function $g : \mathbb{Z}_n \mapsto \mathbb{Z}_n$, there exists a randomized algorithm (Algorithm 2) that, given $n, L \in \mathbb{N}$, and an L-step deterministic GRA G, runs in time $O\left(\frac{L^{7/2}\log^2 n}{\epsilon^{5/2}}\right)$ and, with probability $1 - \epsilon$ over the randomness of the algorithm, it either outputs a factor of n or an L-step SLP S such that $\lambda_n(S, g) \geq \lambda_n(G, g) - \frac{\epsilon}{2}$.

Proof: Let $\delta = \frac{\epsilon}{2L}$. We classify the branching vertices in G into two kinds of vertices – *extreme* and *non-extreme* vertices. For a branching vertex v labeled with (u, w), if

$$\nu_n(P_u^G \cdot Q_w^G - P_w^G \cdot Q_u^G) \in [\delta, 1 - \delta]$$

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

Let G_{ex} be the tree obtained from G by truncating the subtree rooted at v for all non-extreme vertices v. Therefore all non-extreme vertices present in G_{ex} are at the leaves. Also, we can assume, without loss of generality, that a leaf vertex of G is not a branching vertex since that would be of no use. Hence the leaf vertices of G_{ex} are either non-branching or non-extreme branching vertices.

Algorithm 2:

Input: $G.root$, n
Output: A factor of n or an SLP S

```
Initialize S to be a path of length 1 with G.root, G.root.left;
Let v be the grandchild of the root of G;
while v.left ≠ ⊥ do
    if v is a non-branching vertex then Append v with its label to S;
    else
        Let label of v be (u, w);
        for i ← 1 to M do
            Generate a uniformly random element x ∈ Z_n;
            Compute g as the gcd of P_u^G(x)·Q_w^G(x) − P_w^G(x)·Q_u^G(x) and n;
            if g ∉ {1, n} then return g;
        end
        Generate a uniformly random element x' ∈ Z_n;
        if P_u^G(x')·Q_w^G(x') − P_w^G(x')·Q_u^G(x') = 0
        then v = v.left
        else v = v.right;
    end
end
Return S;
```

Let $v^\star = v^\star(G_{ex})$ be the unique leaf vertex of G_{ex} reached by traversing down starting from the root and inputting, for all extreme vertices v labeled with (u, w), and going to the right edge if $\nu_n(P_u^G \cdot Q_w^G - P_w^G \cdot Q_u^G) \in [0, \delta)$ and to the left edge if $\nu_n(P_v^G \cdot Q_v^G - P_v^G \cdot Q_v^G) \in (1 - \delta, 1]$. We call the path from the root to the vertex $v^\star$ 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 (recall that δ is small). Let $S^\star$ denote the straight-line program corresponding to this path.

Let $M = \lceil \frac{L^{3/2}}{(\epsilon/2)^{5/2}} \rceil$. Consider Algorithm 2.

The intuition is that when executing the GRA for a random element $a \in \mathbb{Z}_n$, 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 corrresponds 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 path from the root to a leaf of G (excluding branching vertices). This path corresponds to a unique path in G_{ex}. This path is chosen by generating, for each equality query, a uniformly random element in $\mathbb{Z}_n$ and then testing the equality 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 $v^\star$.

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

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 $v_S = v^\star$, i.e., that the dominating path is found by Algorithm 2. The event E does not occur if there exists an extreme vertex v with label (u, w) in the path from the root of G_{ex} to v_S such that Algorithm 2 proceeds to the child corresponding to the unlikely output, i.e., goes to left child and $\nu_n(P_w^G \cdot Q_u^G - P_u^G \cdot Q_w^G) \in [0, \delta)$ or goes to right child and $\nu_n(P_w^G \cdot Q_u^G - P_u^G \cdot Q_w^G) \in (1-\delta, 1]$. 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 G_{ex} to v_S. Therefore,

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

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

- CASE 1: $v^\star$ is a non-extreme vertex.
  In this case we show that the factoring algorithm is successful with probability at least $1 - \epsilon$.
  Let E' be the event that Algorithm 2 returns a factor of n. We compute $\Pr(\mathsf{E}'|\mathsf{E})$. If E holds, then v_S is a non-extreme vertex. Therefore, by Lemma 2, a factor of n is returned in one test in Step 10 at the equality query corresponding to v_S with probability at least $\delta^{3/2}$. The total number of times step 10 is repeated for this equality query is M. Therefore,[6]

$$\begin{aligned}\Pr(\mathsf{E}'|\mathsf{E}) &\geq 1 - (1-\delta^{3/2})^M \\ &\geq 1 - \exp(-\delta^{3/2} M) \\ &= 1 - \exp(-\tfrac{2}{\epsilon}) \\ &\geq 1 - \frac{\epsilon}{2}.\end{aligned}$$

  This implies

$$\begin{aligned}\Pr(\mathsf{E}') &\geq \Pr(\mathsf{E}'|\mathsf{E}) \cdot \Pr(\mathsf{E}) \\ &\geq (1 - \tfrac{\epsilon}{2})^2 \\ &\geq 1 - \epsilon.\end{aligned}$$

- CASE 2: $v^\star$ 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 $\lambda_n(S, g) \geq \lambda_n(G, g) - \frac{\epsilon}{2}$.[7]
  The fraction of inputs $a \in \mathbb{Z}_n$ such that when G is run on a, the corresponding path taken on G_{ex} does not terminate in $v^\star$ 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,

$$\lambda_n(S^\star, g) \;\geq\; \lambda_n(G, g) - \tfrac{\epsilon}{2}.$$

  So, if E occurs, then the following event occurs.

$$\lambda_n(S, g) \geq \lambda_n(G, g) - \tfrac{\epsilon}{2}.$$

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

[7]Note that the straight-line program S is a random variable depending on the random choices made by the algorithm.

Hence,

$$\begin{aligned}\Pr\left(\lambda_n(S, g) \geq \lambda_n(G, g) - \tfrac{\epsilon}{2}\right) &\geq \Pr(\mathsf{E}) \\ &\geq 1 - \tfrac{\epsilon}{2}.\end{aligned}$$

The running time of the algorithm is dominated by step 9 which involves evaluating $P_u^G(x) \cdot Q_w^G(x) - P_w^G(x) \cdot Q_u^G(x)$ for a random input x and then computing its gcd with n. This step takes time $O(L \log^2 n)$ and is executed at most $O(LM)$ times. Therefore the time complexity of the algorithm is $O(L^2 \cdot M \cdot \log^2 n) = O\left(\frac{L^{7/2} \log^2 n}{\epsilon^{5/2}}\right)$. □

C. Finishing the Proof of Theorem 2

We combine Lemma 4 and Lemma 5 to get the following result.

Corollary 2: For all $\epsilon \in (0, 1/2]$, there exists an algorithm that takes as input $n = pq$, where $p, q > 3$ are primes, an integer $e > 1$, $L \in \mathbb{N}$, and an L-step GRA G, runs in time $O(\frac{L^{7/2} \log^2 n}{\epsilon^{5/2}} + \log^3 e \log^2 n)$ and returns a factor of n with probability at least $\frac{\lambda_n(G, x^{1/e}) - \epsilon/2}{64(L + \log e)}$ over the randomness of the algorithm.

Proof: First, Algorithm 2 is executed with input n, e, and G. If this yields a factor of n, this factor is returned. Otherwise, Algorithm 1 is executed, with the SLP output by Algorithm 2 as input.

With probability $1 - \epsilon$ over the randomness of Algorithm 2, it either returns a factor of n or an L-step SLP that succeeds with probability $\lambda_n(G, x^{1/e}) - \epsilon/2$. In the latter case, Algorithm 1 returns a factor of n with probability $\frac{\lambda_n(G, x^{1/e}) - \epsilon/2}{32(L + \log e)}$. Thus, the success probability of the factoring algorithm is at least

$$(1 - \epsilon) \cdot \frac{\lambda_n(G, x^{1/e}) - \epsilon/2}{32(L + \log e)} \geq \frac{\lambda_n(G, x^{1/e}) - \epsilon/2}{64(L + \log e)}.$$

□

Now we can conclude the proof of Theorem 2.

Proof: Consider Corollary 2. Let $\epsilon = \frac{\mu}{2} \in (0, 1/2]$. By linearity of expectation, the probability that the algorithm given in Corollary 2 with input $\mathcal{G}, n, e$ factors n is at least

$$\frac{\mu}{64(L + \log e)} - \frac{\epsilon/2}{64(L + \log e)} \geq \frac{\mu}{100(L + \log e)}.$$

By repeating the algorithm $O((L + \log e)/\mu)$ times, the success probability can be increased to $1/2$. The running time of the algorithm is clearly polynomial in L, $\log n$, $\log e$, and $\frac{1}{\mu}$. □

REFERENCES

[1] D. Aggarwal and U. Maurer, "Breaking RSA generically is equivalent to factoring," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), vol. 5479. Cologne, Germany: Antoine Joux, 2009, pp. 36–53.

[2] K. Altmann, T. Jager, and A. Rupp, "On black-box ring extraction and integer factorization," in *Proc. 35th Int. Colloq. (ICALP)*, (Lecture Notes in Computer Science), vol. 5126. 2008, pp. 437–448.

[3] D. Boneh and R. J. Lipton, "Algorithms for black-box fields and their application to cryptography," in *Advances in Cryptology—CRYPTO* (Lecture Notes in Computer Science), vol. 1109. Santa Barbara, CA, USA: Neal Koblitz, 1996, pp. 283–297.

[4] D. Boneh and R. Venkatesan, "Breaking RSA may not be equivalent to factoring," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), vol. 1403. Espoo, Finland: Kaisa Nyberg, 1998, pp. 59–71.

[5] D. R. L. Brown, "Breaking RSA may be as difficult as factoring," *J. Cryptol.*, vol. 29, no. 1, pp. 220–241, 2016.

[6] I. Damgård and M. Koprowski, "Generic lower bounds for root extraction and signature schemes in general groups," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), vol. 2332. Amsterdam, The Netherlands: Lars R. Knudsen, 2002, pp. 256–271.

[7] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theory*, vol. 22, no. 6, pp. 644–654, Nov. 1976.

[8] T. Jager, "Generic group algorithms," M.S. thesis, Dept. Comput. Sci., Ruhr Univ. Bochum, Bochum, Germany, 2007.

[9] T. Jager and J. Schwenk, "On the analysis of cryptographic assumptions in the generic ring model," in *Advances in Cryptology—ASIACRYPT* (Lecture Notes in Computer Science), vol. 5912. Tokyo, Japan: Mitsuru Matsui, 2009, pp. 399–416.

[10] A. Joux, D. Naccache, and E. Thomé, "When e-th roots become easier than factoring," in *Advances in Cryptology—ASIACRYPT* (Lecture Notes in Computer Science), vol. 4833. Kuching, Malaysia: Kaoru Kurosawa, 2007, pp. 13–28.

[11] G. Leander and A. Rupp, "On the equivalence of RSA and factoring regarding generic ring algorithms," in *Advances in Cryptology—ASIACRYPT* (Lecture Notes in Computer Science), vol. 4284. Shanghai, China: Xuejia Lai, Kefei Chen, 2006, pp. 241–251.

[12] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and Their Applications*. Cambridge, U.K.: Cambridge Univ. Press, 1994.

[13] U. Maurer, "Towards the equivalence of breaking the Diffie–Hellman protocol and computing discrete logarithms," in *Advances in Cryptology—CRYPTO* (Lecture Notes in Computer Science), vol. 839. Santa Barbara, CA, USA: Yvo Desmedt, 1994, pp. 271–281.

[14] U. Maurer, "Abstract models of computation in cryptography," in *Cryptography and Coding* (Lecture Notes in Computer Science), vol. 3796. Cirencester, U.K.: Nigel P. Smart, 2005, pp. 1–12.

[15] U. Maurer and D. Raub, "Black-box extension fields and the inexistence of field-homomorphic one-way permutations," in *Advances in Cryptology—ASIACRYPT* (Lecture Notes in Computer Science), vol. 4833. Kuching, Malaysia: Kaoru Kurosawa, 2007, pp. 427–443.

[16] U. Maurer and S. Wolf, "The relationship between breaking the Diffie–Hellman protocol and computing discrete logarithms," *SIAM J. Comput.*, vol. 28, no. 5, pp. 1689–1721, Apr./May 1999.

[17] V. I. Nečhaev, "Complexity of a deterministic algorithm for the discrete logarithm," *Math. Notes*, vol. 55, no. 2, pp. 91–101, 1994.

[18] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[19] V. Shoup, "Lower bounds for discrete logarithms and related problems," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), vol. 1233. Konstanz, Germany: Walter Fumy, 1997, pp. 256–266.

Authors' photographs and biographies not available at the time of publication.