On the Equivalence of RSA and Factoring regarding Generic Ring Algorithms

Gregor Leander and Andy Rupp

Horst-Görtz Institute for IT-Security,
Ruhr-University Bochum, Germany.

leander@itsc.rub.de
arupp@crypto.rub.de

Abstract. To prove or disprove the computational equivalence of solving the RSA problem and factoring integers is a longstanding open problem in cryptography. This paper provides some evidence towards the validity of this equivalence. We show that any efficient generic ring algorithm which solves the (flexible) low-exponent RSA problem can be converted into an efficient factoring algorithm. Thus, the low-exponent RSA problem is intractable w.r.t. generic ring algorithms provided that factoring is hard.

Keywords: Computational Equivalence, RSA Problem, Factorization Problem, Generic Algorithms.

1 Introduction and Related Work

The security of the well-known RSA encryption and signature scheme [1] relies on the hardness of the so-called RSA or root extraction problem: Let $n = pq$ be the product of two large primes and let e be a positive integer s.t. $\gcd(e, \phi(n)) = 1$. Then given n, e and an element $x \in \mathbb{Z}_n$, the challenge is to find an element $y \in \mathbb{Z}_n$ s.t. $y^e = x$. The RSA problem is closely related to the problem of factoring integers, i.e., in the case of an RSA modulus, finding p and q given n. While it is well-known that the RSA problem can be reduced to the factorization problem it is a longstanding open problem whether the converse is true, i.e., if an algorithm for finding e-th roots can be utilized in order to factor n efficiently.

Theoretical results towards disproving resp. proving the existence of such a reduction from the factorization to the RSA problem have been provided by Boneh and Venkatesan [2] resp. Brown [3]. In both papers the low-exponent variant of the RSA problem (LE-RSA) is considered, where the public exponent e is restricted to be smaller than some fixed constant or a product of small factors. Moreover, the results given in these

papers are limited to (slight extensions) of straight line programs (SLPs). These are non-probabilistic algorithms only allowed to perform a fixed sequence of addition, subtraction and multiplication steps on their inputs without branching or looping. Thus, the result of such a program can be represented by a fixed integer polynomial in its inputs.

Boneh and Venkatesan [2] show that any straight line program that efficiently factors n given access to an oracle solving the LE-RSA problem can be converted into a real polynomial-time factoring algorithm. This means, there exists no straight line reduction from factoring to LE-RSA, unless factoring is easy. The authors also show that this holds for algebraic reductions, which are straight line reductions extended by basic branching steps based on equality tests.

Recently, Brown [3] shows that any straight line program solving the LE-RSA problem also reveals the factorization of the RSA modulus. In other words, the LE-RSA problem is intractable for SLPs provided that factoring is hard. More precisely, he proves that an efficient SLP for breaking LE-RSA can always be transformed into an efficient factoring algorithm. Moreover, Brown outlines (see Appendix F in [3]) how this result extends to a generalization of SLPs (called SLEEPs) which are additionally allowed to perform basic branching steps based on the equality of elements.

At first sight, Brown's result seems to be contradictory to [2], since an SLP for breaking LE-RSA aids in factoring the modulus. However, the factoring algorithms considered by Brown which make use of the LE-RSA SLP are no straight line programs and in addition the LE-RSA SLP is not simply used as a black-box as it is done in [2]. So both results do not contradict but are results in opposite directions.

Another important theoretical result about the hardness of the RSA problem is due to Damgård and Koprowski [4]. They studied the problem of root extraction in finite abelian groups of unknown order and prove that both the standard and the flexible RSA problem, where the parameter e is no fixed input but can be chosen freely, are intractable w.r.t. generic *group* algorithms.

The concept of generic group algorithms has been introduced by Nechaev [5] and Shoup [6]. Loosely speaking, generic algorithms are probabilistic algorithms that given a group G as black box, may only perform a set of basic operations on the elements of G such as applying the group law, inversion of group elements and equality testing. Since the group is treated as black-box, the algorithms cannot exploit any special properties of the representation of group elements.

It is important to note that the generic algorithms for solving the (flexible) RSA problem considered in [4] are restricted in the following respects: They can only exploit the group structure of the multiplicative group $\mathbb{Z}_n^*$ and not the full ring structure of $\mathbb{Z}_n$ which would be more natural in the case of the RSA problem. Moreover, the RSA modulus n is not given as input to them. Instead, the multiplicative group is chosen at random according to a publicly known probability distribution and the algorithms know that the group order lies in a certain interval. Damgård and Koprowski leave it as an open problem to consider the RSA problem in a more natural generic model not having the restrictions described above.

1.1 Our Contribution

In this paper we propose a solution to the open problem stated in [4] by considering the hardness of the flexible LE-RSA problem w.r.t. to generic *ring* algorithms. We consider the following model of a generic ring algorithm: Let $\sigma : \mathbb{Z}_n \to S_n$, where $S_n \subset \{0,1\}^{\lceil \log_2(n) \rceil}$ and $|S_n| = n$, denote a random encoding function for $\mathbb{Z}_n$ which is a function randomly chosen from the set of bijective mappings from $\mathbb{Z}_n$ into the set of bit strings of sufficient length. A generic ring algorithm for the flexible RSA problem is a probabilistic algorithm which is given n, S_n and the encodings $\sigma(0), \sigma(1)$ and $\sigma(x)$ as input. These encodings are the initial content of the *encoding list* which contains all encodings $\sigma(x_i)$ of ring elements x_i occurring during a computation. In a computation the algorithm can query a *ring oracle*, which given two indices i and j into this list computes $\sigma(x_i \pm x_j)$ or $\sigma(x_i x_j)$ and appends this encoding to the list. After some queries the algorithm finally outputs a pair $(e, \sigma(y))$ where $e > 1$ and $\gcd(e, \phi(n)) = 1$. It succeeds iff $y^e = x$.

Note that given the factorization of n, computing e-th roots is possible using $O(\log(n))$ oracle queries. So clearly it is not possible to prove that a generic ring algorithm given n needs exponential many oracle queries to solve the problem, since the algorithm might first factor n (without using the oracle) and then compute the e-th root using $O(\log(n))$ queries. Therefore any approach to prove something about the hardness of the problem in this model has to relate the RSA problem to the factorization problem.

We show that any efficient generic ring algorithm which solves the flexible LE-RSA problem with non-negligible probability can be converted into an efficient factoring algorithm having non-negligible probability. The considered generic algorithms can thereby only choose e from the set of exponents having some small fixed constant factor. Thus, the LE-RSA

problem is intractable w.r.t. generic ring algorithms unless factoring is easy.

The paper at hand extends the results by Brown to a broader and more natural class of algorithms: First, the class of generic ring algorithms is clearly larger than the class of SLPs. Moreover, each SLEEP can be implemented as generic ring algorithm. However, it is not known if every generic ring algorithm can be realized as a SLEEP. We note that for part of our proof we use a Theorem given in [3].

2 Relating Flexible LE-RSA to Factoring

2.1 Generic Ring Algorithms

We formalize the notion of a generic algorithm for the ring $\mathbb{Z}_n$ based on Shoup's generic group model [6]. To this end, the group oracle just needs to be extended by a multiplication operation in order to make the full ring structure of $\mathbb{Z}_n$ available. However, the ring oracle $\mathcal{O}$ we present slightly differs from such an extended group oracle in the following sense: Instead of using the ring $\mathbb{Z}_n$ for the internal representation of ring elements, these elements are represented by polynomials in the variable X over $\mathbb{Z}_n$ which are evaluated with x each time the encoding of a newly computed element must be determined. It is easy to see that both versions of a generic ring oracle are actually equivalent. However, we believe that the presented version is a better starting point for doing and understanding proofs in the generic model.

The generic oracle $\mathcal{O}$ is defined as follows:
Input: As input $\mathcal{O}$ receives $x \in_U \mathbb{Z}_n$, the modulus n and a list $\{\sigma_1, \ldots, \sigma_n\}$ of n pairwise distinct bit strings randomly chosen from S_n.
Internal State: As internal state $\mathcal{O}$ maintains two lists L and E which always have the same length. For an index $j \in \{1, \ldots, |L|\}$ let L_j denote the j-th element of L and E_j the j-the element of E. In the list L, polynomials $L_j \in \mathbb{Z}_n[X]$ are stored which represent computed ring elements $L_j(x)$. The list E contains the encodings E_j of the corresponding ring elements $L_j(x)$. Moreover, $\mathcal{O}$ maintains a counter c which counts the number of different elements contained in E and the encoding function $\sigma : \mathbb{Z}_n \to S_n$ which will be gradually defined during computation by the assignments between computed ring elements and the bit strings $\sigma_1, \ldots, \sigma_n$.
Encoding elements: Each time a polynomial P is appended to the list L (during the initialization or query-handling phase described below) it is checked whether the corresponding element $P(x)$ has already been com-

puted. More precisely, $\mathcal{O}$ checks if there exists any $j \in \{1, \ldots, |L|\}$ s.t.

$$(P - L_j)(x) \equiv 0 \bmod n\,.$$

If this is not the case, $\mathcal{O}$ increases the counter c and appends the random bit string $\sigma_c \in S_n \setminus E$ to E which is different from all encodings so far contained in E. Additionally, the partially defined encoding function is updated with the new assignment, i.e., $\sigma := \sigma \cup \{P(x) \mapsto \sigma_c\}$. If the equation holds for any j the bit string E_j is again appended to E.
A run of $\mathcal{O}$ consists of three phases:
Initialization: In this phase all lists are first set to the empty list, c is set to zero and the encoding function σ is set to be undefined for all $x \in \mathbb{Z}_n$. After that, L is appended with the polynomials $0, 1$ and X, E is appended with the respective encodings and σ and c are updated accordingly.
Query-handling: In the query-handling phase $\mathcal{O}$ handles at most m queries. A query is a triple $(\circ, j_1, j_2)$ where $\circ \in \{+, -, \cdot\}$ identifies an operation and j_1, j_2 are indices identifying the list elements the operation should be applied to. A query $(\circ, j_1, j_2)$ is handled by computing the polynomial $P := L_{j_1} \circ L_{j_2}$, appending P to L and the respective encoding to E and updating σ and c accordingly.
Finalization: After an algorithm $\mathcal{A}$ has made at most m queries to $\mathcal{O}$, it signals $\mathcal{O}$ to finalize the computation before it eventually does its final output. Upon receiving this signal, $\mathcal{O}$ updates the encoding function σ by assigning (in some fixed order) the $n-c$ ring elements $x \in \mathbb{Z}_n \setminus \{P(x) | P \in L\}$ which have not already occurred during computation to the random bit strings $\sigma_{c+1}, \ldots, \sigma_n$. After that, $\mathcal{O}$ signals $\mathcal{A}$ to output its solution (e, out), where $\mathit{out} \in S_n$, $e > 1$ and $\gcd(e, \phi(n)) = 1$. We define the following success event

> $\mathcal{S}$: $\mathcal{A}$ outputs an encoding $\mathit{out} = \sigma(y)$ and an integer e such that $y^e \equiv x \bmod n$.

2.2 Main Theorem

Our result lower bounding the hardness of flexible RSA in terms of the hardness of factoring integers can be stated as follows:

Theorem 1. *Let $\mathcal{O}$ be a generic ring oracle for the ring $\mathbb{Z}_n$ of order $n = pq$ as defined above. Let $\mathcal{A}$ be a generic algorithm that makes at most m oracle queries to $\mathcal{O}$ and let $(e, \sigma(y)) \leftarrow \mathcal{A}^{\mathcal{O}}(n, S_n, \sigma(0), \sigma(1), \sigma(x))$, where $e > 1$ and $\gcd(e, \phi(n)) = 1$, be its final output. Then the probability that*

y is an e-th root of x is upper bounded by

$$\Pr[y^e = x] \leq \big(4\phi(e') + 2\big)\,\gamma + \frac{1}{n - m - 3}\,,$$

where e' is the smallest factor of e and γ is a lower bound on the probability that n can be factored using $\mathcal{A}$ and $O((\phi(e')^2 + \log(n))m^2)$ additional operations in $\mathbb{Z}_n$.

Note that the above theorem gives an upper bound on the probability that $\mathcal{A}$ finds an e-th root which depends on the particular exponent e chosen by $\mathcal{A}$. More precisely, it is dependent on the size of the factors of e. This in particular means that we do not obtain a useful lower bound for exponents e consisting of "large" factors only. "Large" in this context means that the factors cannot be bounded by a polynomial in the security parameter $\log(n)$. However, if we restrict the class of allowed exponents $\mathcal{A}$ can choose from to "low exponents", i.e., exponents having at least one factor which is smaller than some fixed constant C, we always obtain a useful bound.

Corollary 1 (Hardness of Flexible LE-RSA). *Let $\mathcal{O}$ be a generic ring oracle for the ring $\mathbb{Z}_n$ of order $n = pq$ as defined above and let C be an arbitrary constant. Let $\mathcal{A}$ be a generic algorithm that makes at most m oracle queries to $\mathcal{O}$ and let $(e, \sigma(y)) \leftarrow \mathcal{A}^{\mathcal{O}}(n, S_n, C, \sigma(0), \sigma(1), \sigma(x))$ be its final output, where $e > 1$ has a factor smaller than C and $\gcd(e, \phi(n)) = 1$. Then the probability that y is an e-th root of x is upper bounded by*

$$\Pr[y^e = x] \leq (4C + 2)\,\gamma + \frac{1}{n - m - 3}\,,$$

where γ is a lower bound on the probability that n can be factored using $\mathcal{A}$ and $O((C^2 + \log(n))m^2)$ additional operations in $\mathbb{Z}_n$.

Let us assume that the number of queries m is polynomial bounded. Then observe that the probability γ is negligible if factoring is assumed to be hard since γ is a lower bound on the probability of factoring n using a polynomial bounded number of operations in $\mathbb{Z}_n$. Thus, in this case also the upper bound on the probability $\Pr[y^e = x]$ given in the corollary is negligible because m and C are polynomial bounded and γ is negligible. Hence, if factoring is hard Corollary 1 implies that the standard and the flexible LE-RSA problem are intractable w.r.t. generic ring algorithms. On the other hand, if for some special n root extraction is easy for generic algorithms, which might be possible, we know from our corollary that n can easily be factored.

Remark 1. In [4] special care has to be taken of the distribution of the group orders. More precisely, the order of the multiplicative group has to be randomly chosen according to certain so-called "hard" distributions in order to derive the desired exponential lower bounds on the running time of generic group algorithms. This was an extension of Shoup's original model for the purpose of handling groups of hidden order. From this perspective things are easier in our model. As the order n of the additive group of the ring is given we do not have to worry about any special properties of the distribution according to which the order of the multiplicative group is chosen.

3 Proof of the Main Theorem

3.1 Outline

As usually done in proofs within the scope of the generic (group) model, we replace the original oracle $\mathcal{O}$ with an oracle $\mathcal{O}_{sim}$ that simulates $\mathcal{O}$ without using the knowledge of the secret x. Then we show that the behavior of $\mathcal{O}_{sim}$ is perfectly indistinguishable from $\mathcal{O}$ unless a certain simulation failure $\mathcal{F}$ occurs. From this, it immediately follows that the success probability of $\mathcal{A}$ when interacting with $\mathcal{O}$ is upper bounded by the sum of failure probability and the success probability of $\mathcal{A}$ when interacting with $\mathcal{O}_{sim}$. We upper bound these probabilities in terms of the probability γ from Theorem 1 and the number of oracle queries.

Remark 2. The main difficulty in proving Theorem 1 is to bound the probability of a simulation failure $\mathcal{F}$. Usually, $\mathcal{O}_{sim}$ is defined in a way that a simulation failure occurs iff two distinct polynomials $L_i, L_j \in L$ become equal under evaluation with x and one can determine a useful (i.e., negligible) upper bound on the probability of $\mathcal{F}$ in terms of the maximal degree of such a difference polynomial $L_i - L_j$. However, here we face the problem that by using repeated squaring, $\mathcal{A}$ can generate polynomials in L with exponential high degrees. Thus, we cannot derive non-trivial bounds anymore using this well-known technique. Note that this difficulty is inherent to the ring structure and does usually not occur when we consider cryptographic problems over generic groups. We solve it by simulating $\mathcal{O}$ in a new way and relating the probability of $\mathcal{F}$ to the probability γ.

3.2 The Simulation Game

The simulation oracle $\mathcal{O}_{sim}$ is defined exactly like $\mathcal{O}$ except that it determines the encoding of elements differently in order to be independent of the secret x. To this end, each time a polynomial P is appended to the end of list L (during initialization or query-handling), $\mathcal{O}_{sim}$ does the following: Let $L_i = P$ denote the last entry of the updated list. Then for each $j < i$ the oracle chooses a random element $x_j^{(i)} \in_U \mathbb{Z}_n$ and checks whether

$$(L_i - L_j)(x_j^{(i)}) \equiv 0 \bmod n\,.$$

If this equation holds for some $j_1, \ldots, j_k$ the encoding E_j, where $j = \min(j_1, \ldots, j_k)$, is appended as the encoding of the newly computed element to the list E.[1] If no j exists s.t. the equation holds, counter c is increased and the random bit string $\sigma_c \in S_n \setminus E$ which is different from all encodings already contained in E is appended to E. Moreover, σ is updated by the assignment $P(x) \mapsto \sigma_c$.

Note that due to the modifications to the computation of encodings, it is now possible that both an element $P(x)$ is assigned to two or more different encodings and more than one element is assigned to the same encoding. Thus, the number $r_1 := n - c$ of unused encodings remaining after the query-handling phase may be greater or smaller than the number $r_2 := n - |\{P(x)|P \in L\}|$ of elements not occurring during computation. In the finalization phase $\mathcal{O}_{sim}$ therefore assigns only $\min(r_1, r_2)$ elements from $\mathbb{Z}_n \setminus \{P(x)|P \in L\}$ to the encodings $\sigma_{c+1}, \ldots, \sigma_{c+\min(r_1,r_2)}$ (but using the same order as $\mathcal{O}$).

Let us consider the following events which can occur in an interaction with the simulation oracle:

$\mathcal{F}$: There exists $i > j \in \{1, \ldots, |L|\}$ such that

$$(L_i - L_j)(x) \equiv 0 \bmod n \text{ and } (L_i - L_j)(x_j^{(i)}) \not\equiv 0 \bmod n$$

or

$$(L_i - L_j)(x) \not\equiv 0 \bmod n \text{ and } (L_i - L_j)(x_j^{(i)}) \equiv 0 \bmod n\,.$$

$\mathcal{S}_{sim}$: $\mathcal{A}$ outputs (e, out) such that out is the encoding of an unique element y and $y^e = x$.

[1] Note that it is not important how j is determined from $\{j_1, \ldots, j_k\}$. j can be chosen from this set in an arbitrary way.

The event $\mathcal{S}_{sim}$ is the success event in a simulation game. The event $\mathcal{F}$ is called simulation failure. It is important to observe that the original game and the simulation game proceed identically unless $\mathcal{F}$ occurs: Assume that $\mathcal{O}$ and $\mathcal{O}_{sim}$ receive the same arbitrary but fixed input. Then issuing the same sequence of queries to $\mathcal{O}$ and $\mathcal{O}_{sim}$ results in the same sequence of encodings contained in E, the same sequence of polynomials contained in L and the same bijective encoding function σ, provided that $\mathcal{F}$ does not happen. Furthermore, consider an algorithm $\bar{\mathcal{A}}$ with an arbitrary but fixed input on its random tape. Since $\bar{\mathcal{A}}$ is deterministic, it issues the same sequence of queries in both interactions if it receives the same sequence of encodings from $\mathcal{O}$ and $\mathcal{O}_{sim}$. So assuming that $\mathcal{F}$ does not happen, $\bar{\mathcal{A}}$ outputs the same exponent and encoding in both interactions and wins the simulation game if and only if it wins the original game. Thus, we have the following relation between the considered events

$$\mathcal{S} \wedge \neg\mathcal{F} \Leftrightarrow \mathcal{S}_{sim} \wedge \neg\mathcal{F}\,.$$

Using this relation we immediately obtain the desired upper bound on the success probability $\Pr[\mathcal{S}]$ in the original game in terms of the failure probability $\Pr[\mathcal{F}]$ and the probability $\Pr[\mathcal{S}_{sim} \wedge \neg\mathcal{F}]$ that no failure occurs and the algorithm succeeds in the simulation game.

$$\begin{aligned}\Pr[\mathcal{S}] &= \Pr[\mathcal{S} \wedge \neg\mathcal{F}] + \Pr[\mathcal{S} \wedge \mathcal{F}] \\ &= \Pr[\mathcal{S}_{sim} \wedge \neg\mathcal{F}] + \Pr[\mathcal{S} \wedge \mathcal{F}] \\ &\leq \Pr[\mathcal{S}_{sim} \wedge \neg\mathcal{F}] + \Pr[\mathcal{F}]\end{aligned}$$

In the following we relate these probabilities to the probability γ.

3.3 Simulation Failure Probability

For arbitrary but fixed indices $i > j \in \{1, \ldots, m+3\}$ we consider the difference polynomial $\Delta := L_i - L_j$. Let

$$N(\Delta) := \{a \in \mathbb{Z}_n \mid \Delta(a) \equiv 0 \bmod n\}$$

denote the set of zeros of this polynomial. Using the Chinese Remainder Theorem we can split $N(\Delta)$ into two sets

$$N(\Delta) \cong N^p(\Delta) \times N^q(\Delta), \text{ where}$$

$N^p(\Delta) = \{a \in \mathbb{Z}_p \mid \Delta(a) \equiv 0 \bmod p\}$ and $N^q(\Delta) = \{a \in \mathbb{Z}_q \mid \Delta(a) \equiv 0 \bmod q\}$.

Let the value $|N^p(\Delta)|/p$ be denoted by μ_Δ and $|N^q(\Delta)|/q$ by ν_Δ. The probability that a randomly chosen element $a \in_U \mathbb{Z}_n$ is a zero of the polynomial Δ can then be written as

$$\Pr[\Delta(a) \equiv 0 \bmod n; a \in_U \mathbb{Z}_n] = \nu_\Delta \mu_\Delta \,.$$

Thus, the probability $\Pr[\mathcal{F}_\Delta]$ that for a fixed polynomial Δ a simulation failure occurs is given by

$$\begin{aligned}\Pr[\mathcal{F}_\Delta] &= \Pr[\Delta(x) \equiv 0 \bmod n; x \in_U \mathbb{Z}_n](1 - \Pr[\Delta(x_j^{(i)}) \equiv 0 \bmod n; x_j^{(i)} \in_U \mathbb{Z}_n]) \\ &\quad + \Pr[\Delta(x_j^{(i)}) \equiv 0 \bmod n; x_j^{(i)} \in_U \mathbb{Z}_n](1 - \Pr[\Delta(x) \equiv 0 \bmod n; x \in_U \mathbb{Z}_n]) \\ &= 2\nu_\Delta \mu_\Delta (1 - \nu_\Delta \mu_\Delta) \,.\end{aligned}$$

Now, we relate the failure probability $\Pr[\mathcal{F}_\Delta]$ with the probability γ from Theorem 1. First observe that if we can find an element

$$a \in ((\mathbb{Z}_p \setminus N^p(\Delta)) \times N^q(\Delta)) \cup (N^p(\Delta) \times (\mathbb{Z}_q \setminus N^q(\Delta))) \,,$$

the polynomial Δ gives us the factorization of n by computing $\gcd(\Delta(a), n)$. Thus, the probability γ_Δ that the factorization can be revealed in this way by choosing a random $a \in_U \mathbb{Z}_n$ is given by

$$\gamma_\Delta = \mu_\Delta(1 - \nu_\Delta) + (1 - \mu_\Delta)\nu_\Delta = \mu_\Delta + \nu_\Delta - 2\mu_\Delta\nu_\Delta \,.$$

The crucial observation is the following lemma.

Lemma 1. *For any polynomial $\Delta \in \mathbb{Z}_n[X]$ it holds that $\Pr[\mathcal{F}_\Delta] \leq 2\gamma_\Delta$.*

Proof. We can see that $2\gamma_\Delta - \Pr[\mathcal{F}_\Delta] \geq 0$ by considering the following function:

$$\begin{aligned} f &: \mathbb{R} \times \mathbb{R} \to \mathbb{R} \\ f(\mu, \nu) &= (\mu\nu)^2 - 3\mu\nu + \mu + \nu \end{aligned}$$

In order to prove the lemma, we have to show that this function does not reach any negative values in $[0, 1]$. The only critical point in the set $[0, 1] \times [0, 1]$ and therefor the only possible extremum is

$$(\mu_0, \nu_0) = \left(\frac{\sqrt{3} - 1}{2}, \frac{\sqrt{3} - 1}{2} \right)$$

and we have $f(\mu_0, \nu_0) > 0$. Furthermore for the boundaries of the set $[0,1] \times [0,1]$ we get

$$\begin{aligned} f(0,\nu) &= \nu \geq 0\,, \\ f(\mu,0) &= \mu \geq 0\,, \\ f(1,\nu) &= (\nu-1)^2 \geq 0\,, \\ f(\mu,1) &= (\mu-1)^2 \geq 0\,. \end{aligned}$$

Thus it follows that for all $(\mu,\nu) \in [0,1] \times [0,1]$ we have $f(\mu,\nu) \geq 0$. □

Now, given $\mathcal{A}$ consider an algorithm that evaluates all possible difference polynomials Δ with a randomly chosen element $a \in_U \mathbb{Z}_n$ and computes for each integer $\Delta(a)$ the value $\gcd(\Delta(a), n)$. The probability that n can be factored in this way is given by

$$\sum_{1 \leq j < i \leq |L| : \Delta := L_i - L_j} \gamma_\Delta\,.$$

The evaluation of all polynomials Δ can be done using a total of $O(m^2)$ operations. Computing all greatest common divisors requires $O(\log(n)m^2)$ operations using the Euclidean algorithm. So the probability of this factoring algorithm can be upper bounded by γ (cf. Theorem 1).

Using Lemma 1 we obtain the following bound on the probability of a simulation failure

$$\begin{aligned} \Pr[\mathcal{F}] &\leq \sum_{1 \leq j < i \leq |L| : \Delta := L_i - L_j} \Pr[\mathcal{F}_\Delta] \\ &\leq \sum_{1 \leq j < i \leq |L| : \Delta := L_i - L_j} 2\gamma_\Delta \\ &\leq 2\gamma\,. \end{aligned}$$

3.4 Success Probability in the Simulation Game

Let us split up the success event $\mathcal{S}_{sim}$ in two sub-events: We say that the generic algorithm wins if either it outputs a new encoding $out \notin E$ corresponding to an unique element y which is an e-th root of x or if a polynomial in the list yields an e-th root when evaluated with the element x. We denote these events by

$\mathcal{S}^1_{sim}$: $\mathcal{A}$ outputs (e, out) s.t. $out \notin E$, out is the encoding of an unique element y and $y^e = x$.

$\mathcal{S}^2_{sim}$: There exists a polynomial $P \in L$ s.t. $P(x)^e = x$.

Note that $\mathcal{S}^2_{sim}$ is more than actually needed: Here we do not require that $\mathcal{A}$ actually outputs an encoding corresponding to $P(x)$, the existence of such a polynomial P in L is sufficient. We therefore have

$$\mathcal{S}_{sim} \Rightarrow \mathcal{S}^1_{sim} \vee \mathcal{S}^2_{sim}$$

and thus

$$\Pr[\mathcal{S}_{sim} \wedge \neg\mathcal{F}] \leq \Pr[\mathcal{S}^1_{sim} \wedge \neg\mathcal{F}] + \Pr[\mathcal{S}^2_{sim}] \,.$$

Probability of Event $\mathcal{S}^1_{sim} \wedge \neg\mathcal{F}$. Assume that the event $\mathcal{F}$ has not happened during computation and $\mathcal{A}$ outputs a pair (e, out) s.t. $out \notin E$. Since no simulation failure has occurred, σ is a bijective mapping and in particular the encodings $\sigma_{c+1}, \ldots, \sigma_n$ not used in the query-handling phase are uniquely associated with the $n-c$ elements in $\mathbb{Z}_n \setminus \{P(x)|P \in L\}$. So the encoding *out* corresponds to a randomly chosen element $y \in \mathbb{Z}_n \setminus \{P(x)|P \in L\}$. Thus, we have

$$\Pr[\mathcal{S}^1_{sim} \wedge \neg\mathcal{F}] \leq \Pr[\mathcal{S}^1_{sim} \mid \neg\mathcal{F} \wedge out \notin E] \leq \frac{1}{n-(m+3)} \,.$$

Probability of Event $\mathcal{S}^2_{sim}$. Here we use the following Lemma which corresponds to (a slight extension of) Theorem 6 in [3].

Lemma 2. *Let $n = pq$, p, q prime and $e \in \mathbb{N}$ with $\gcd(e, \phi(n)) = 1$. Let a polynomial $P \in \mathbb{Z}_n[X]$ be given that can be evaluated for any element $x \in \mathbb{Z}_n$ using at most m additions and multiplications in $\mathbb{Z}_n$. For random $x \in_U \mathbb{Z}_n$ let the probability $\Pr[P(x)^e = x]$ be denoted by ε_P. Then using this polynomial n can be factored with probability*

$$\gamma_P \geq \frac{(e'-1)(N-1)}{\phi(e')e'N}\varepsilon_P$$

with at most $O(3\phi(e')^2 m)$ operations in $\mathbb{Z}_n$, where e' is the smallest factor of e and N is the base of the natural logarithm.

The main idea behind this result is to evaluate P over an appropriate extension of $\mathbb{Z}_n$, where the mapping $x \mapsto x^{e'}$ is not a bijection anymore. Then one can use the well-known techniques to factor n given two different e'-th roots of the same element.

We now apply this result in our setting. First, observe that clearly all polynomials $P \in L$ can be evaluated using at most m operations in $\mathbb{Z}_n$.

Thus, we can apply Lemma 2 to each P, i.e., we consider an algorithm that applies the procedure outlined in the proof of Theorem 6 in [3] to every polynomial in L. The running time of this algorithm is $O(\phi(e')^2 m^2)$. The probability that n can be factored this way is given by $\sum_{P \in L} \gamma_P$ and by the definition of γ (cf. Theorem 1) it follows that

$$\sum_{P \in L} \gamma_P \leq \gamma \,.$$

Furthermore, it is easy to see that

$$\varepsilon_P \leq \frac{\phi(e')e'N}{(e'-1)(N-1)} \gamma_P \leq 4\phi(e')\gamma_P \,.$$

So we can conclude that the probability of the event $\mathcal{S}^2_{sim}$ is bounded by

$$\begin{aligned} \Pr(\mathcal{S}^2_{sim}) &\leq \sum_{P \in L} \varepsilon_P \\ &\leq \sum_{P \in L} 4\phi(e')\gamma_P \leq 4\phi(e')\gamma \,. \end{aligned}$$

3.5 Putting Things Together

Using the bounds on the probabilities in the simulation game we can bound the success probability in the original game. For a generic algorithm $\mathcal{A}$ which makes m queries to $\mathcal{O}$ and outputs a pair $(e, \sigma(y))$ consider an algorithm which

- chooses an element $a \in_U \mathbb{Z}_n$,
- computes $\gcd((L_i - L_j)(a), n)$ for each $i > j \in \{1, \ldots, m+3\}$ and
- applies the procedure given in the proof of Theorem 6 in [3] to each L_i.

The running time of this algorithm is $O((\phi(e')^2 + \log(n))m^2)$ and by definition of γ its probability to factor n is less than γ.

Hence, the probability that y is an e-th root of the randomly chosen element x is bounded by

$$\begin{aligned} \Pr[y^e = x] &\leq \Pr[\mathcal{S}_{sim} \wedge \neg\mathcal{F}] + \Pr[\mathcal{F}] \\ &\leq \Pr[\mathcal{S}^1_{sim} \wedge \neg\mathcal{F}] + \Pr[\mathcal{S}^2_{sim}] + \Pr[\mathcal{F}] \\ &\leq \frac{1}{n-m-3} + 4\phi(e')\gamma + 2\gamma \\ &= \big(4\phi(e') + 2\big)\,\gamma + \frac{1}{n-m-3} \,. \end{aligned}$$

This completes the proof of Theorem 1 and Corollary 1.

Acknowledgments. We would like to thank Ivan Damgård, Daniel Brown as well as the anonymous reviewers for their valuable comments.

References

1. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2) (1978) 120–126
2. Boneh, D., Venkatesan, R.: Breaking RSA may not be equivalent to factoring. In: Advances in Cryptology: Proceedings of EUROCRYPT 1998. Volume 1403 of Lecture Notes in Computer Science., Springer-Verlag (1998) 59–71
3. Brown, D.R.L.: Breaking RSA may be as difficult as factoring. Cryptology ePrint Archive, Report 2005/380 (2006) http://eprint.iacr.org/.
4. Damgård, I., Koprowski, M.: Generic lower bounds for root extraction and signature schemes in general groups. In: Advances in Cryptology: Proceedings of EUROCRYPT 2002. Volume 2332 of Lecture Notes in Computer Science., Springer-Verlag (2002) 256–271
5. Nechaev, V.I.: Complexity of a determinate algorithm for the discrete logarithm. Mathematical Notes **55**(2) (1994) 165–172
6. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Advances in Cryptology: Proceedings of EUROCRYPT 1997. Volume 1233 of Lecture Notes in Computer Science., Springer-Verlag (1997) 256–266