The Quadratic Residuosity Problem May Be Equivalent to Factoring

Tibor Jager and Jörg Schwenk

Horst Görtz Institute for IT Security
Ruhr-University Bochum, Germany
{tibor.jager,joerg.schwenk}@rub.de

November 14, 2008

Abstract. *Generic ring algorithms* are the class of algorithms running on an algebraic ring, such as $\mathbb{Z}_n$, and working independent of a specific representation of ring elements. We consider the *quadratic residuosity* problem, a problem of high relevance for a large number of cryptographic applications, and show that solving this problem with generic ring algorithms is equivalent to factoring integers. This provides first formal evidence towards the widely adopted conjecture that solving the quadratic residuosity problem in $\mathbb{Z}_n$ is hard, assuming that factoring n is hard. At the same time our results imply that, in order to solve the quadratic residuosity problem more efficiently than factoring integers, specific properties of a given representation of ring elements have to be exploited. This result is relevant for the design of algorithms solving the quadratic residuosity problem. To our best knowledge this is the first time that a *decisional* problem of cryptographic interest is shown to be equivalent to the *computational* problem "integer factorization" for a certain class of algorithms.

1 Introduction

The security of asymmetric cryptographic systems depends usually on assumptions that certain computational problems, mostly from number theory and algebra, are intractable. Since it is unknown whether these assumptions hold in a general model of computation (e.g. the Turing machine model), restricted models have been considered, e.g. [1–6]. One natural and quite general restricted model of computation is the model of *generic ring algorithms.* This model considers a class of algorithms running on an algebraic ring, without exploiting specific properties of a given representation of ring elements. Such algorithms work in a similar way for an arbitrary representation of ring elements, thus are *generic.*

In this work we analyze the *quadratic residuosity problem*, a famous problem in cryptography and number theory, with respect to generic ring algorithms. The quadratic residuosity problem is to decide whether a given element $x \in \mathbb{Z}_n$ is a *quadratic residue*, i.e. whether there exists $y \in \mathbb{Z}_n$ such that $x \equiv y^2 \bmod n$. It is well-known that solving the quadratic residuosity problem is *at most* as hard as factoring n. However, it is a long-standing and important open problem whether solving this problem for all $x \in \mathbb{Z}_n$ is also *at least* as hard as factoring n. Many important cryptographic applications rely on the assumption that solving this problem is hard if factoring n is hard, e.g. the Goldwasser-Micali cryptosystem [7], the Blum-Blum-Shub pseudorandom number generator [8], encryption schemes secure against adaptive chosen-ciphertext attacks [9], interactive proof systems [10], and oblivious transfer protocols [11], to mention a few.

In this work we provide first formal evidence that the quadratic residuosity problem is hard under the factoring assumption, by showing that solving it with generic ring algorithms is equivalent to factoring integers.

1.1 Related Work

Previous theoretical work considering the fundamental cryptographic assumptions in a restricted model of computation was targeted especially on the discrete logarithm and the RSA problem. It is known that solving the discrete logarithm and related problems such as the Diffie-Hellman problem, and the problem of root extraction in groups of hidden order, is hard with respect to generic *group* algorithms [1, 12, 13, 3].

Especially the analysis of the relationship between the RSA problem and factoring integers in restricted models of computation is strongly related to our work. Brown [4] reduced the factoring problem to solving the low-exponent RSA problem with *straight line programs*, which are a subclass of generic ring algorithms. Leander and Rupp [5] augmented this result to generic ring algorithms, where the considered algorithms may only perform the operations addition, subtraction and multiplication modulo n, but not multiplicative inversion operations. Recently Aggarwal and Maurer [6] extended this result from low-exponent RSA to full RSA and to generic ring algorithms that may also compute multiplicative inverses. Boneh and Venkatesan [2] have shown that there is no straight line program reducing integer factorization to the low-exponent RSA problem, unless factoring integers is easy.

The concept of generic ring algorithms has also been applied to study the relationship between the discrete logarithm and the Diffie-Hellman problem, cf. [14–16].

We are not aware of any formal evidence that the quadratic residuosity problem is intractable under the factoring assumption, neither in a general model of computation (e.g. the probabilistic Turing machine model), nor in a restricted model, such as the model of generic ring algorithms.

1.2 Our Contribution

We analyze the quadratic residuosity problem with respect to generic ring algorithms that may exploit the full algebraic structure of $\mathbb{Z}_n$ by performing the operations addition, subtraction, multiplication, and multiplicative inversion modulo n. We consider the general case where n is the product of *at least* two different odd primes, thus including the famous special case where $n = pq$ with p, q prime and $p \neq q$. We show that solving the quadratic residuosity problem with generic ring algorithms is equivalent to factoring n. This result has at least the following important interpretations.

- We provide first formal evidence towards the widely adopted conjecture that solving the quadratic residuosity problem is hard under the assumption that factoring integers is hard. In addition, our result may be a step towards a proof in a general model of computation.
- If solving the quadratic residuosity problem is substantially easier than factoring in general, then our results imply at least that one cannot hope to solve the problem by only using the elementary ring operations and without essentially factoring the modulus. This is an important result for the design of algorithms for the quadratic residuosity problem.

Remarkably, to our best knowledge this is the first time that a *decisional* problem relevant for cryptographic applications is shown to be equivalent to the *computational* problem "integer factorization" for a certain class of algorithms.

Our proof technique extends and generalizes a concept introduced by Leander and Rupp [5]. The model of generic ring algorithms considered in [5] captures only the case where n is the product of exactly two different primes, and does not include algorithms that may compute multiplicative inverses. Since inverses in $\mathbb{Z}_n$ can be computed easily with the extended Euclidean algorithm, excluding this operation appears to be an unfair limitation. We extend the technique of [5] to the case where n is the product of at least two different primes and, in order to capture a large and natural class of algorithms, to algorithms that can perform multiplicative inversion operations, and adapt it to the quadratic residuosity problem. This extension may also be valuable for the study of other cryptographic assumptions in the generic ring model.

2 Generic Ring Algorithms

Generic ring algorithms are the class of algorithms running on the structure of an algebraic ring, without exploiting specific properties of a given representation of ring elements. Hence generic ring algorithms work in a similar way for any representation of ring elements. We adapt the generic group model of Shoup [1] to formalize the notion of generic ring algorithms. To hide any property of an explicitly given representation of a ring R, we encode elements of R as random but unique bit strings, and provide a *generic ring oracle* $\mathcal{O}$ computing the ring operations on encoded ring elements. Let R be a ring with $|R| = n$, and let $S_n \subseteq \{0,1\}^{\lceil \log_2 n \rceil}$ with $|S_n| = n$. An element of R is represented by a uniformly random element of S_n. The function mapping elements of R to elements of S_n is a bijection, which enables an algorithm to check for equality of encoded ring elements by checking equality of encodings.

Definition 1 (Generic Ring Algorithm). *A* generic ring algorithm $\mathcal{A}$ *is a (possibly probabilistic) polynomial-time Turing machine taking as input a r-tuple $(E_1, \ldots, E_r) \in S_n^r$ of randomly encoded ring elements. The algorithm may query a* generic ring oracle *to perform computations on encoded ring elements.*

Depending on the particular problem the algorithm might take some additional data as input, such as the cardinality n of R, for example. We measure the complexity of a generic ring algorithm by the number of performed ring operations (=oracle queries).

3 Generic Quadratic Residuosity and Factoring

Let us first state a few facts from elementary number theory, as far as required for the following sections. We refer to [17] for details and a comprehensive treatment. In the following let $n = \prod_{i=1}^{k} p_i^{e_i}$ be the prime factor decomposition of n such that $\gcd(p_i, p_j) = 1$ for $i \neq j$.

For $n \in \mathbb{N}$ we denote with $\mathsf{QR}_n \subseteq \mathbb{Z}_n$ the set of *quadratic residues* modulo n, i.e.

$$\mathsf{QR}_n := \{x \in \mathbb{Z}_n \mid x \equiv y^2 \bmod n, y \in \mathbb{Z}_n\}.$$

Definition 2. *Let $x \in \mathbb{Z}$ and $n \in \mathbb{N}$.*

(i) The Legendre symbol $(x \mid n)_L$ *is defined for prime n as*

$$(x \mid n)_L := \begin{cases} 1 & \text{if } x \in \mathsf{QR}_n \text{ and } x \not\equiv 0 \bmod n \\ -1 & \text{if } x \notin \mathsf{QR}_n \\ 0 & \text{if } x \equiv 0 \bmod n. \end{cases}$$

(ii) The Jacobi symbol $(x \mid n)$ *is defined for* $n = \prod_{i=1}^{k} p_i^{e_i}$ *as*

$$(x \mid n) := \prod_{i=1}^{k} (x \mid p_i)_L^{e_i} \, .$$

There exist efficient algorithms computing the Jacobi symbol, even if the factorization of n is unknown, see [17], for instance. Also note that $(x \mid n) = 1$ does not imply that x is a quadratic residue modulo n if n is composite.

Fact 1 *Let $x \in \mathbb{Z}$, p prime, and $n = \prod_{i=1}^{k} p_i^{e_i}$ be the product of odd primes.*

(i) x is a quadratic residue modulo p^e, $e \in \mathbb{N}$, if and only if $(x \mid p)_L = 1$.
(ii) x is a quadratic residue modulo n if and only if $(x \mid p_i)_L = 1$ for all $i \in \{1, \ldots, k\}$.

Hence, determining whether a given element $x \in \mathbb{Z}_n$ is a quadratic residue modulo n can be performed by factoring n and then computing the Legendre symbol for x and each prime factor of n. Thus the quadratic residuosity problem is at most as hard as factoring n.

Note that $x \in \mathsf{QR}_n$ and $\gcd(x, n) = 1$ implies $(x \mid n) = 1$. Let

$$J_n := \{x \in \mathbb{Z}_n \mid (x \mid n) = 1\}$$

be the set of elements of $\mathbb{Z}_n$ having Jacobi symbol 1. Given a non-zero element $x \in \mathbb{Z}_n \backslash J_n$ it is easy to decide that x is not a quadratic residue by computing the Jacobi symbol.

Remark 1. The cardinality $|J_n|$ of the set of elements having Jacobi symbol 1 depends on whether n is a square in $\mathbb{N}$.

$$|J_n| = \begin{cases} \phi(n)/2, & \text{if } n \text{ is not a square in } \mathbb{N}, \\ \phi(n), & \text{if } n \text{ is a square in } \mathbb{N}, \end{cases}$$

where $\phi(\cdot)$ is the Euler totient function. In the sequel we assume that n is not a square (otherwise finding a factor of n is clearly easy), and use the equation $\frac{|J_n|}{|\mathbb{Z}_n|} = \frac{\phi(n)}{2n}$ whenever required in the sequel, however, assuming factoring n is hard it should be safe to approximate $\frac{|J_n|}{|\mathbb{Z}_n|} = \frac{\phi(n)}{2n} \approx \frac{1}{2}$.

The probability that a uniformly random element $x \leftarrow_r J_n$ is a quadratic residue in $\mathbb{Z}_n$ depends on the number of distinct prime factors of n and their multiplicity. Let

$$\Pi'(n) := \Pr[x \in \mathsf{QR}_n \mid x \leftarrow_r J_n] = \frac{|\mathsf{QR}_n|}{|J_n|}$$

denote the probability that a uniformly random element of J_n is a quadratic residue in $\mathbb{Z}_n$. If $x \leftarrow_r J_n$ is chosen uniformly random, then there exists a trivial algorithm determining whether x is a quadratic residue by guessing. This algorithm has success probability

$$\Pi(n) := \max\{\Pi'(n), 1 - \Pi'(n)\}.$$

3.1 The Generic Quadratic Residuosity Problem

In the sequel let n be the product of at least two different primes. We formalize the *generic* quadratic residuosity problem in terms of a game between an algorithm $\mathcal{A}$ and an oracle $\mathcal{O}_{\mathrm{QR}}$.

Game 0 (Original Game). The oracle $\mathcal{O}_{\mathrm{QR}}$ takes as input an integer $n \in \mathbb{N}$ and a uniformly random element $x \leftarrow_r J_n$, and keeps two internal lists $\mathsf{L} \subseteq \mathbb{Z}_n$ and $\mathsf{E} \subseteq S_n$ as internal state to maintain the assignment of encodings to ring elements. Let L_i and E_i denote the i-th element of L and E, respectively, then E_i is the encoding of L_i. The oracle implements the internal procedures `invertible`(), `compute`(), and `encode`(), which are defined as follows.

- The `invertible`()-procedure takes $L \in \mathsf{L}$ as input, and returns `true` if $L \in \mathbb{Z}_n^*$ and `false` if $L \notin \mathbb{Z}_n^*$.
- The `encode`()-procedure is called each time an element is appended to L. The procedure checks whether there exists a list element L_i, $i \in \{1, \dots, |\mathsf{L}| - 1\}$, such that
$$L_{|\mathsf{L}|} \equiv L_i \bmod n.$$
For the first i where this holds, the oracle sets $E_{|\mathsf{L}|} := E_i$. If there is no such i, a new encoding $E_{|\mathsf{L}|} \leftarrow_r S_n \backslash \mathsf{E}$ is chosen at random from the set of unused encodings and appended to E.
- The `compute`()-procedure takes a triple $(i, j, \circ) \in \{1, \dots, |\mathsf{L}|\} \times \{1, \dots, |\mathsf{L}|\} \times \{+, -, \cdot, /\}$ as input. If $\circ = /$, then the the procedure `invertible`(L_j) is called; if `invertible`(L_j) = `false` then the error symbol $\perp$ is returned. Otherwise the procedure computes $\lambda = L_i \circ L_j$, appends λ to L, and calls the `encode`()-procedure.

Moreover, the oracle implements the public accessible functions `init`() and `query`(), which may be called by the algorithm $\mathcal{A}$. The public functions are defined as follows.

- When the `init`()-function is called, the lists L and E are set to the empty list. Then the elements $1 \in \mathbb{Z}_n$ and x are appended to L, i.e. $L_1 := 1$ and $L_2 := x$, each time calling the `encode`()-procedure. The `init`()-function returns the triple (n, E_1, E_2).
- The `query`()-function takes as input a triple $(E', E'', \circ) \in \mathsf{E} \times \mathsf{E} \times \{+, -, \cdot, /\}$. It determines the smallest $i, j \in \{1, \dots, |\mathsf{L}|\}$ such that $E_i = E'$ and $E_j = E''$ and calls `compute`$(i, j, \circ)$. If `compute`$(i, j, \circ) = \perp$, then the query function returns $\perp$. Otherwise the encoding $E_{|\mathsf{L}|}$ of the computed element is returned.

At the beginning of the game the algorithm calls the `init`()-function and receives (n, E_1, E_2) as input. Then it may call the `query`()-function to perform computations on encoded ring elements. We assume that the algorithm calls the `query`()-function at most m times.

We say that $\mathcal{A}$ wins the game, if $x \in \mathsf{QR}_n$ and $\mathcal{A}^{\mathcal{O}_{\mathrm{QR}}}(n, E_1, E_2) = 1$, or $x \notin \mathsf{QR}_n$ and $\mathcal{A}^{\mathcal{O}_{\mathrm{QR}}}(n, E_1, E_2) = 0$.

Definition 3 (Generic Quadratic Residuosity Problem). *Given (n, E_1, E_2) and access to oracle $\mathcal{O}_{\mathrm{QR}}$ such that $L_1 = 1$ and $L_2 = x$ for $x \leftarrow_r J_n$. Output 1 if $x \in \mathsf{QR}_n$, else output 0.*

3.2 The Equivalence of Generic Quadratic Residuosity and Factoring

There exist efficient *generic* algorithms solving the quadratic residuosity problem in $\mathbb{Z}_n$ given the factorization of $n = \prod_{i=1}^{k} p_i^{e_i}$, e.g. by checking whether $(x^{(p_i - 1)/2} - 1) \cdot n/p_i \equiv 0 \bmod n$ for all $i \in \{1, \dots, k\}$. Thus, in order to prove the equivalence of the quadratic residuosity problem and factoring integers with respect to generic ring algorithms, we are left with proving the following theorem.

Theorem 1. *Let $n \in \mathbb{N}$ be the product of at least two different odd primes. Suppose there exists a generic ring algorithm $\mathcal{A}$ performing m ring operations and solving the quadratic residuosity problem in $\mathbb{Z}_n$ with probability $\Pi(n) + \epsilon$. Then there exists an algorithm $\mathcal{B}$ performing at most $O(m^4)$ operations in $\mathbb{Z}_n$ and $O(m^3)$ gcd-computations on $\lceil \log_2 n \rceil$-bit numbers, and finding a factor of n with probability at least*

$$\frac{\epsilon}{4(m^2 + 5m + 6)} \left(\frac{\phi(n)}{n} \right)^2 .$$

Remark 2. Suppose there is an algorithm solving the generic quadratic residue problem for arbitrary n. Then n can be factored completely by first running $\mathcal{B}$ on n. If $\mathcal{B}$ outputs a factor n_1 of n, run a new instance of $\mathcal{B}$ on n_1 (unless n_1 is a prime power) and $n_2 = n/n_1$ (unless n_2 is a prime power). Repeating this procedure at most $\lceil \log_2 n \rceil$ times yields the factorization of n, since n has at most $\lceil \log_2 n \rceil$ prime factors.

4 Proof of Theorem 1

Outline. We replace the generic ring oracle $\mathcal{O}_{\text{QR}}$ with a simulator $\mathcal{O}_{\text{sim}}$. To make this step more comprehensible and easier to verify, we introduce the simulator by a *sequence of games* [18]. That is, we make gradual modifications to the oracle described in Game 0 by modifying in Game i the generic ring oracle from Game $i-1$ for $i \in \{1, \ldots, 4\}$. The actual simulation oracle $\mathcal{O}_{\text{sim}}$ is introduced in Game 4. An interaction of $\mathcal{A}$ with $\mathcal{O}_{\text{sim}}$ is indistinguishable from an interaction with the original oracle $\mathcal{O}_{\text{QR}}$, unless a *simulation failure* $\mathbf{F}$ occurs. Let $\mathbf{S}_{\text{sim}}$ denote the event that $\mathcal{A}$ wins in the simulation game. Then it follows that the success probability $\Pi(n) + \epsilon$ in the original game is upper bounded by $\Pr[\mathbf{S}_{\text{sim}}] + \Pr[\mathbf{F}]$. We derive an upper bound on $\Pr[\mathbf{S}_{\text{sim}}]$ and describe a factoring algorithm $\mathcal{B}$ running $\mathcal{A}$ as a subroutine such that $\Pr[\mathbf{F}]$ is essentially a lower bound on the success probability of $\mathcal{B}$.

4.1 Introducing a Simulation Oracle

Game 1. We modify the internal representation of ring elements. The notion of the introduced representation is that an element $L \in \mathbb{Z}_n$ is stored as a tuple $(N, D) \in \mathbb{Z}_n \times \mathbb{Z}_n^*$ such that $L \equiv ND^{-1} \bmod n$. The list L is now defined as $\mathsf{L} \subseteq \mathbb{Z}_n \times \mathbb{Z}_n^*$. For a tuple $L_i \in \mathsf{L}$ we denote with N_i the first and with D_i the second component, i.e. $L_i = (N_i, D_i)$. We have to make the following modifications to the internal procedures and public functions of $\mathcal{O}_{\text{QR}}$.

- The `init()`-function appends the elements $(1,1) \in \mathbb{Z}_n \times \mathbb{Z}_n^*$ and $(x, 1)$ to L, each time calling the `encode()`-procedure. The function returns the triple (n, E_1, E_2).
- The `invertible()`-procedure takes $L_j \in \mathsf{L}$ as input, and returns `true` if $N_j \in \mathbb{Z}_n^*$ and `false` if $N_j \notin \mathbb{Z}_n^*$.
- The `compute()`-procedure takes a triple $(i, j, \circ) \in \{1, \ldots, |\mathsf{L}|\} \times \{1, \ldots, |\mathsf{L}|\} \times \{+, -, \cdot, /\}$ as input. If $\circ = /$, then the the procedure `invertible`(L_j) is called; if `invertible`$(L_j) =$ `false` then the error symbol $\perp$ is returned. Otherwise the result λ is computed as follows.

$$\lambda := \begin{cases} (N_i D_j + N_j D_i, D_i D_j), & \text{if } \circ = +, \\ (N_i D_j - N_j D_i, D_i D_j), & \text{if } \circ = -, \\ (N_i N_j, D_i D_j), & \text{if } \circ = \cdot, \\ (N_i D_j, D_i N_j), & \text{if } \circ = /. \end{cases}$$

The result is appended to L.

– The `encode()`-procedure checks whether there exists a list element L_i, $i \in \{1, \ldots, |\mathsf{L}|-1\}$, such that
$$N_{|\mathsf{L}|}D_{|\mathsf{L}|}^{-1} \equiv N_i D_i^{-1} \bmod n.$$
For the first i where this holds, the oracle sets $E_{|\mathsf{L}|} := E_i$. If there is no such i, a new encoding $E_{|\mathsf{L}|} \leftarrow_r S_n \backslash \mathsf{E}$ is chosen uniformly random from the set of unused encodings.

Game 2. We make a simple change to the `encode()`-procedure. To check whether a newly computed ring element $L_{|\mathsf{L}|}$ has already been computed, the oracle tests whether there exists $L_i \in \mathsf{L}$, $i \in \{1, \ldots, |\mathsf{L}|-1\}$, such that
$$(N_{|\mathsf{L}|}D_i - N_i D_{|\mathsf{L}|}) \equiv 0 \bmod n.$$
Note that this equality test is equivalent to testing
$$N_{|\mathsf{L}|}D_{|\mathsf{L}|}^{-1} \equiv N_i D_i^{-1} \bmod n,$$
since $D_i \in \mathbb{Z}_n^*$ for all $i \in \{1, \ldots, |\mathsf{L}|\}$.

Game 3. Observe that, for any $L_i \in \mathsf{L}$, the elements N_i and D_i are computed by performing a sequence of addition, subtraction and multiplication operations on 1 and x.Thus, the elements N_i and D_i can be seen as *polynomial functions* of x.

Now we replace the element x with a wildcard character X and define $\mathsf{L} \subseteq \mathbb{Z}_n[X] \times \mathbb{Z}_n[X]$. We make the following modifications to the internal and public procedures of $\mathcal{O}_{\mathrm{QR}}$.

– The `init()`-function appends the elements $(1, 1)$ and $(X, 1)$ to L, each time calling the `encode()`-procedure.
– The `invertible()`-procedure takes $L_j \in \mathsf{L}$ as input, and returns `true` if $N_j(x) \in \mathbb{Z}_n^*$ and `false` if $N_j(x) \notin \mathbb{Z}_n^*$.
– The `compute()`-procedure performs all computations component-wise in $\mathbb{Z}_n[X] \times \mathbb{Z}_n[X]$.
– The `encode()`-procedure checks whether there exists a list element L_i, $i \in \{1, \ldots, |\mathsf{L}|-1\}$, such that
$$(N_{|\mathsf{L}|}D_i - N_i D_{|\mathsf{L}|})(x) \equiv 0 \bmod n.$$

Note that $L_i(x) \in \mathsf{L}$ corresponds to the element $N_i(x)/D_i(x) = N_i(x)D_i(x)^{-1}$. We refer to an element $L_i(X) = N_i(X)/D_i(X)$ as *rational function* in X over $\mathbb{Z}_n$.

Game 4 (Simulation Game). We replace the oracle $\mathcal{O}_{\mathrm{QR}}$ with a simulator $\mathcal{O}_{\mathrm{sim}}$. The simulator is defined exactly like the oracle described in Game 3, except for the following modification. In Game 3 the computed polynomials are evaluated with x by the `encode()` and the `invertible()`-procedure, where x is the random element of J_n that $\mathcal{O}_{\mathrm{QR}}$ has received as input at the beginning of the game. In order to make all computations of $\mathcal{A}$ independent of x, we replace these procedures with procedures that evaluate the polynomials with randomly sampled elements. More precisely, when $\mathcal{A}$ calls the `query()`-function for the k-th time, then the oracle samples a uniformly random element $x_k \leftarrow_r J_n$. The procedures `encode()` and `invertible()` are modified as follows.

– The `invertible()`-procedure returns `true` if $N_j(x_k) \in \mathbb{Z}_n^*$ and `false` if $N_j(x_k) \notin \mathbb{Z}_n^*$.

– The `encode()`-procedure checks whether there exists a list element L_i, $i \in \{1, \dots, |\mathsf{L}| - 1\}$, such that
$$(N_{|\mathsf{L}|} D_i - N_i D_{|\mathsf{L}|})(x_k) \equiv 0 \bmod n.$$

Note that all computations of $\mathcal{A}$ are independent of x in the simulation game.

4.2 Probability of Simulation Failure

It is straightforward to verify that Games 1-3 are equivalent to Game 0 (the original game) in the sense that the modified oracles are equivalent to the original oracle, hence Games 0-3 are perfectly indistinguishable. Let us compare the oracle $\mathcal{O}_{\text{sim}}$ introduced in Game 4 with the oracle described in Game 3.

We say that a *simulation failure* occurs, if an interaction of $\mathcal{A}$ with $\mathcal{O}_{\text{sim}}$ is not perfectly indistinguishable from an interaction with $\mathcal{O}_{\text{QR}}$, i.e. the input-output behavior of $\mathcal{O}_{\text{sim}}$ differs from the input-output behavior of $\mathcal{O}_{\text{QR}}$. We denote this event with $\mathbf{F}$. Note that a simulation failure implies that at least one of the following events has occurred.

1. The `invertible()`-procedure in Game 4 simulates the `invertible()`-procedure from Game 3 perfectly, unless there exist $L_j \in \mathsf{L}$ such that the algorithm has queried a division by L_j in the k-th step, and it holds that
$$(N_j(x_k) \in \mathbb{Z}_n^* \text{ and } N_j(x) \notin \mathbb{Z}_n^*) \;\text{ or }\; (N_j(x_k) \notin \mathbb{Z}_n^* \text{ and } N_j(x) \in \mathbb{Z}_n^*)\,.$$
We denote this event with $\mathbf{F}_1$.
2. Let $\delta_k := (N_i D_j - N_j D_i)$ be the difference polynomial computed by the simulator to check for equality of elements after the k-th query of $\mathcal{A}$. The `encode()`-procedure in Game 4 simulates the `encode()`-procedure from Game 3 perfectly, unless there exists δ_k, $k \in \{1, \dots, m\}$, such that
$$(\delta_k(x_k) \equiv 0 \bmod n \text{ and } \delta_k(x) \not\equiv 0 \bmod n) \;\text{ or }\; (\delta_k(x_k) \not\equiv 0 \bmod n \text{ and } \delta_k(x) \equiv 0 \bmod n)\,.$$
We denote this event with $\mathbf{F}_2$.

Since a simulation failure implies that events $\mathbf{F}_1$ or $\mathbf{F}_2$ have occurred, the probability of simulation failure $\mathbf{F}$ is bounded by
$$\Pr[\mathbf{F}] \le \Pr[\mathbf{F}_1] + \Pr[\mathbf{F}_2].$$

In the sequel we will derive bounds on $\Pr[\mathbf{F}_1]$ and $\Pr[\mathbf{F}_2]$ separately.

4.3 Relating the Simulation Failure Probability to Factoring

For $z \in \mathbb{Z}_n[X]$ let us consider the probability of finding a non-trivial factor of n by sampling $x \leftarrow_r \mathbb{Z}_n$ and computing $\gcd(n, z(x))$. This probability is given by
$$\gamma(z) := \Pr[\gcd(n, z(x)) \notin \{1, n\} \mid x \leftarrow_r \mathbb{Z}_n].$$

Bounding the Probability of $\mathbf{F_1}$. For $z \in \mathbb{Z}_n[X]$ let $\pi_1(z)$ denote the probability

$$\pi_1(z) := \Pr[z(x) \in \mathbb{Z}_n^* \text{ and } z(x') \notin \mathbb{Z}_n^* \mid x, x' \leftarrow_r J_n].$$

The probability of $\mathbf{F}_1$ is bounded by

$$\begin{aligned}\Pr[\mathbf{F}_1] =& \Pr\left[\left(N_j(x') \in \mathbb{Z}_n^* \text{ and } N_j(x) \notin \mathbb{Z}_n^*\right) \text{ or } \left(N_j(x') \notin \mathbb{Z}_n^* \text{ and } N_j(x) \in \mathbb{Z}_n^*\right) \mid x, x' \leftarrow_r J_n\right]\\ =& \Pr[\left(N_j(x') \in \mathbb{Z}_n^* \text{ and } N_j(x) \notin \mathbb{Z}_n^*\right) \mid x, x' \leftarrow_r J_n]\\ +& \Pr[\left(N_j(x') \notin \mathbb{Z}_n^* \text{ and } N_j(x) \in \mathbb{Z}_n^*\right) \mid x, x' \leftarrow_r J_n]\\ =& 2\Pr[\left(N_j(x') \in \mathbb{Z}_n^* \text{ and } N_j(x) \notin \mathbb{Z}_n^*\right) \mid x, x' \leftarrow_r J_n]\\ \leq& 2\sum_{L_j \in \mathsf{L}} \pi_1(N_j).\end{aligned}$$

Lemma 1. *Let $n \in \mathbb{N}$ such that n has at least two different prime factors. For $z \in \mathbb{Z}_n[X]$ holds that*

$$4\left(\frac{n}{\phi(n)}\right)^2 \gamma(z) \geq \pi_1(z).$$

Proof (Sketch). Using that $\mathbb{Z}_n \cong \mathbb{Z}_{p_1^{e_1}} \times \cdots \times \mathbb{Z}_{p_k^{e_k}}$ by the Chinese Remainder Theorem, we express $\Pr[\gamma'(z)]$ and $\Pr[\pi_1(z)]$ in terms of $\Pr[z(x) \equiv 0 \bmod p_i \mid x \leftarrow_r \mathbb{Z}_n]$ for $i \in \{1, \ldots, k\}$. The resulting inequality is proven by complete induction. The claim follows, since $\gamma(z) \geq \gamma'(z)$. A full proof is given in Appendix A.

There are at most $m+2$ rational functions in the list L, thus the probability of failure $\mathbf{F}_1$ is bounded by

$$\Pr[\mathbf{F}_1] \leq 2\sum_{L_j \in \mathsf{L}} \pi_1(N_j) \leq 8\left(\frac{n}{\phi(n)}\right)^2 \sum_{L_j \in \mathsf{L}} \gamma(N_j) \leq 8(m+2)\left(\frac{n}{\phi(n)}\right)^2 \gamma_1,$$

where $\gamma_1 := \max_{L_j \in \mathsf{L}}\{\gamma(N_j)\}$. Note that γ_1 is a lower bound on the probability that n can be factored by sampling $x \leftarrow_r \mathbb{Z}_n$ and computing $\gcd(n, N_j(x))$ for all $L_i \in \mathsf{L}$.

Bounding the Probability of $\mathbf{F_2}$. For $z \in \mathbb{Z}_n[X]$ and $x, x' \leftarrow_r J_n$ let $\pi_2(z)$ denote the probability

$$\pi_2(z) := \Pr[z(x) \equiv 0 \bmod n \text{ and } z(x') \not\equiv 0 \bmod n \mid x, x' \leftarrow_r J_n].$$

Let $\varDelta := \{(N_i D_j - N_j D_i) \mid L_i, L_j \in \mathsf{L}, i > j\} \subseteq \mathbb{Z}_n[X]$ be the set of all possible difference polynomials computed by the simulator to check for equality of elements. Then the probability of failure $\mathbf{F}_2$ is bounded by

$$\Pr[\mathbf{F}_2] \leq 2\sum_{\delta \in \varDelta} \pi_2(\delta).$$

Lemma 2. *Let $n \in \mathbb{N}$ such that n has at least two different prime factors. For $z \in \mathbb{Z}_n[X]$ holds that*

$$4\left(\frac{n}{\phi(n)}\right)^2 \gamma(z) \geq \pi_2(z).$$

Proof (Sketch). Similar to the proof of Lemma 1, we express $\Pr[\gamma'(z)]$ and $\Pr[\pi_1(z)]$ in terms of $\Pr[z(x) \equiv 0 \bmod p_i^{e_i} \mid x \leftarrow_r \mathbb{Z}_n]$ for $i \in \{1, \dots, k\}$. The resulting inequality is proven by complete induction. The claim follows, since $\gamma(z) \geq \gamma''(z)$. A full proof is given in Appendix B.

Let $\gamma_2 := \max_{\delta \in \Delta}\{\gamma(\delta)\}$. Note that, since there are at most $m+2$ rational functions in L, it holds that $|\Delta| \leq (m+2)(m+1)/2$. Hence the probability of failure $\mathbf{F}_2$ is bounded by

$$\Pr[\mathbf{F}_2] \leq 2 \sum_{\delta \in \Delta} \pi_2(\delta) \leq 8 \left(\frac{n}{\phi(n)}\right)^2 \sum_{\delta \in \Delta} \gamma(\delta) \leq 4(m^2+3m+2)\left(\frac{n}{\phi(n)}\right)^2 \gamma_2.$$

Also note that γ_2 is a lower bound on the probability that n can be factored by sampling $x \leftarrow_r \mathbb{Z}_n$ and computing $\gcd(n, \delta(x))$ for all $\delta \in \Delta$.

4.4 The Probability of Success in the Simulation Game

All computations are independent of x, hence the success probability of any algorithm is at most $\Pi(n)$.

$$\Pr[\mathbf{S}_{\text{sim}}] \leq \Pi(n).$$

4.5 The Probability of Success in the Original Game

Applying the inequality $\Pi(n) + \epsilon \leq \Pr[\mathbf{S}_{\text{sim}}] + \Pr[\mathbf{F}]$, we obtain that the success probability of algorithm $\mathcal{A}$ in the original game is bounded by

$$\begin{aligned}
\Pi(n) + \epsilon &\leq \Pr[\mathbf{S}_{\text{sim}}] + \Pr[\mathbf{F}] \\
&\leq \Pr[\mathbf{S}_{\text{sim}}] + \Pr[\mathbf{F}_1] + \Pr[\mathbf{F}_2] \\
&\leq \Pi(n) + 4\left(\frac{n}{\phi(n)}\right)^2 \left((2m+4)\gamma_1 + (m^2+3m+2)\gamma_2\right) \\
&\leq \Pi(n) + 4\left(\frac{n}{\phi(n)}\right)^2 (m^2+5m+6)\gamma.
\end{aligned}$$

where $\gamma := \max\{\gamma_1, \gamma_2\}$.

4.6 The Factoring Algorithm

Consider the following factoring algorithm $\mathcal{B}$ that runs $\mathcal{A}$ on an arbitrary instance of the generic quadratic residuosity problem. Algorithm $\mathcal{B}$ records the sequence of queries that $\mathcal{A}$ issues, i.e. records the same list L of polynomials as the simulation oracle. When algorithm $\mathcal{A}$ calls the procedure `query()` for the k-th time, submitting the parameters $(E_i, E_j, \circ)$ for some $i, j \in \{1, \dots, k+2\}$, algorithm $\mathcal{B}$ picks a uniformly random element $x_k \leftarrow_r \mathbb{Z}_n$ and proceeds as follows.

1. If $\circ = /$, compute $\gcd(n, N_j(x_k))$.
2. For each $L_i \in \mathsf{L}$, $i \in \{1, \dots, k-1\}$, compute $\gcd(n, (N_k D_i - N_i D_k)(x_k))$.

Success probability of $\mathcal{B}$. Note that $\mathcal{B}$ finds a factor of n by proceeding this way with probability at least γ. Using that

$$\Pi(n) + \epsilon \leq \Pi(n) + 4\left(\frac{n}{\phi(n)}\right)^2 (m^2 + 5m + 6)\gamma$$

we obtain that the success probability of this algorithm is at least

$$\gamma \geq \frac{\epsilon}{4(m^2 + 5m + 6)} \left(\frac{\phi(n)}{n}\right)^2 .$$

Running time of $\mathcal{B}$.

1. Computing $\gcd(n, N_j(x_k))$ takes $O(m)$ operations in $\mathbb{Z}_n$ to evaluate $N_j(x_k)$ and one gcd-computation on $\lceil \log_2 n \rceil$-bit numbers.
2. Note that $|\mathsf{L}| \leq m + 2$, hence the algorithm has to evaluate at most $(m+2)(m+1)/2 = O(m^2)$ difference polynomials $(N_k D_i - N_i D_k)(x_k)$ and perform $O(m^2)$ gcd-computations on $\lceil \log_2 n \rceil$-bit numbers. Each difference polynomial can be evaluated by performing $O(m)$ operations in $\mathbb{Z}_n$.

These steps are performed at most m times. In addition, the algorithm has to maintain the list L, which takes time and space $O(m)$. Summing up we see that the algorithm $\mathcal{B}$ performs $O(m^4)$ operations in $\mathbb{Z}_n$ and $O(m^3)$ gcd-computations on $\lceil \log_2 n \rceil$-bit numbers. □

Acknowledgements. We thank Andy Rupp and Sven Schäge for helpful discussions.

References

1. Shoup, V.: Lower bounds for discrete logarithms and related problems. In Walter Fumy, ed.: Advances in Cryptology - EUROCRYPT 1997. Volume 1233 of Lecture Notes in Computer Science. (1997) 256–266
2. Boneh, D., Venkatesan, R.: Breaking RSA may not be equivalent to factoring. In Nyberg, K., ed.: EUROCRYPT. Volume 1403 of Lecture Notes in Computer Science., Springer (1998) 59–71
3. Damgård, I., Koprowski, M.: Generic lower bounds for root extraction and signature schemes in general groups. [19] 256–271
4. Brown, D.R.L.: Breaking RSA may be as difficult as factoring. Cryptology ePrint Archive, Report 2005/380 (2005) http://eprint.iacr.org/.
5. Leander, G., Rupp, A.: On the equivalence of RSA and factoring regarding generic ring algorithms. In Lai, X., Chen, K., eds.: ASIACRYPT. Volume 4284 of Lecture Notes in Computer Science., Springer (2006) 241–251
6. Aggarwal, D., Maurer, U.: Factoring is equivalent to generic RSA. Cryptology ePrint Archive, Report 2008/260 (2008) http://eprint.iacr.org/.
7. Goldwasser, S., Micali, S.: Probabilistic encryption. J. Comput. Syst. Sci. **28**(2) (1984) 270–299
8. Blum, L., Blum, M., Shub, M.: A simple unpredictable pseudo-random number generator. SIAM J. Comput. **15**(2) (1986) 364–383
9. Cramer, R., Shoup, V.: Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. [19] 45–64
10. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM J. Comput. **18**(1) (1989) 186–208
11. Kalai, Y.T.: Smooth projective hashing and two-message oblivious transfer. In Cramer, R., ed.: EUROCRYPT. Volume 3494 of Lecture Notes in Computer Science., Springer (2005) 78–95
12. Nechaev, V.I.: Complexity of a determinate algorithm for the discrete logarithm. Mathematical Notes **55**(2) (1994) 165–172

13. Maurer, U.M.: Abstract models of computation in cryptography. In Smart, N.P., ed.: IMA Int. Conf. Volume 3796 of Lecture Notes in Computer Science., Springer (2005) 1–12
14. Boneh, D., Lipton, R.J.: Algorithms for black-box fields and their application to cryptography (extended abstract). In Koblitz, N., ed.: CRYPTO. Volume 1109 of Lecture Notes in Computer Science., Springer (1996) 283–297
15. Maurer, U., Raub, D.: Black-box extension fields and the inexistence of field-homomorphic one-way permutations. In Kurosawa, K., ed.: ASIACRYPT. Volume 4833 of Lecture Notes in Computer Science., Springer-Verlag (2007) 427–443
16. Altmann, K., Jager, T., Rupp, A.: On black-box ring extraction and integer factorization. In Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I., eds.: ICALP (2). Volume 5126 of Lecture Notes in Computer Science., Springer (2008) 437–448
17. Shoup, V.: A Computational Introduction to Number Theory and Algebra. Cambridge University Press (2005)
18. Shoup, V.: Sequences of games: A tool for taming complexity in security proofs (2006) URL: `http://eprint.iacr.org/2004/332`.
19. Knudsen, L.R., ed.: Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings. In Knudsen, L.R., ed.: EUROCRYPT. Volume 2332 of Lecture Notes in Computer Science., Springer (2002)

A Proof of Lemma 1

For a polynomial $z \in \mathbb{Z}_n[X]$, and $t \in \mathbb{N}$ let

$$\nu(t,z) := \frac{|\{x \in \mathbb{Z}_n \mid z(x) \equiv 0 \bmod t\}|}{n}$$

denote the probability that $z(x) \equiv 0 \bmod t$ for uniformly random $x \leftarrow_r \mathbb{Z}_n$.

A factor of n is found by sampling $x \leftarrow_r \mathbb{Z}_n$ and computing $\gcd(n, z(x))$ if (but not only if) there exist prime p_i and p_j dividing n such that $z(x) \equiv 0 \bmod p_i$ and $z(x) \not\equiv 0 \bmod p_j$. Using that $\mathbb{Z}_n \cong \mathbb{Z}_{p_1^{e_1}} \times \cdots \times \mathbb{Z}_{p_k^{e_k}}$ by the Chinese Remainder Theorem, the success probability of this algorithm can be expressed as

$$\begin{aligned}\gamma'(z) :=& 1 - \prod_{i=1}^{k} \Pr[z(x) \equiv 0 \bmod p_i \mid x \leftarrow_r \mathbb{Z}_n] - \prod_{i=1}^{k} \Pr[z(x) \not\equiv 0 \bmod p_i \mid x \leftarrow_r \mathbb{Z}_n] \\ =& 1 - \prod_{i=1}^{k} \nu(p_i, z) - \prod_{i=1}^{k} (1 - \nu(p_i, z)) \,.\end{aligned}$$

Note that $\gamma(z) \geq \gamma'(z)$.

The probability $\pi_1(z)$ can be expressed in terms of $\nu(\cdot,\cdot)$ as

$$\begin{aligned}\pi_1(z) &= \Pr[z(x') \in \mathbb{Z}_n^* \wedge z(x) \notin \mathbb{Z}_n^* \mid x, x' \leftarrow_r J_n] \\ &= \Pr[z(x) \in \mathbb{Z}_n^* \mid x \leftarrow_r J_n] \Pr[z(x) \notin \mathbb{Z}_n^* \mid x \leftarrow_r J_n] \\ &\leq \Pr[z(x) \in \mathbb{Z}_n^* \mid x \leftarrow_r \mathbb{Z}_n] \Pr[z(x) \notin \mathbb{Z}_n^* \mid x \leftarrow_r \mathbb{Z}_n] \left(\frac{|\mathbb{Z}_n|}{|J_n|}\right)^2 \\ &= \Pr[z(x) \in \mathbb{Z}_n^* \mid x \leftarrow_r \mathbb{Z}_n] (1 - \Pr[z(x) \in \mathbb{Z}_n^* \mid x \leftarrow_r \mathbb{Z}_n]) \left(\frac{2n}{\phi(n)}\right)^2 \\ &= \left(\prod_{i=1}^{k} (1 - \nu(p_i, z)) - \prod_{i=1}^{k} (1 - \nu(p_i, z))^2\right) \left(\frac{2n}{\phi(n)}\right)^2,\end{aligned}$$

where we use that

$$\Pr[z(x) \in \mathbb{Z}_n^* \mid x \leftarrow_r \mathbb{Z}_n] \geq \Pr[z(x) \in \mathbb{Z}_n^* \mid x \leftarrow_r J_n] \frac{|J_n|}{|\mathbb{Z}_n|}$$

and

$$\Pr[z(x) \notin \mathbb{Z}_n^* \mid x \leftarrow_r \mathbb{Z}_n] \geq \Pr[z(x) \notin \mathbb{Z}_n^* \mid x \leftarrow_r J_n] \frac{|J_n|}{|\mathbb{Z}_n|}.$$

We prove Lemma 1 by showing that

$$\left(\frac{2n}{\phi(n)}\right)^2 \gamma'(z) \geq \left(\prod_{i=1}^{k} (1 - \nu(p_i, z)) - \prod_{i=1}^{k} (1 - \nu(p_i, z))^2\right) \left(\frac{2n}{\phi(n)}\right)^2.$$

This inequality is equivalent to

$$1 - \prod_{i=1}^{k} \nu(p_i, z) - \prod_{i=1}^{k} (1 - \nu(p_i, z)) \geq \prod_{i=1}^{k} (1 - \nu(p_i, z)) - \prod_{i=1}^{k} (1 - \nu(p_i, z))^2$$

which in turn is equivalent to

$$\left(1 - \prod_{i=1}^{k} (1 - \nu(p_i, z))\right)^2 \geq \prod_{i=1}^{k} \nu(p_i, z).$$

Lemma 3. *For $k \in \mathbb{N}$ and $\mu_i \in [0, 1]$ with $i \in \{1, \ldots, k\}$ holds that*

$$\left(1 - \prod_{i=1}^{k}(1 - \mu_i)\right)^k \geq \prod_{i=1}^{k} \mu_i.$$

Proof. The proof proceeds by induction on $k \in \mathbb{N}$. The case $k = 1$ is obvious. The step $k \to k + 1$ proceeds as follows.

$$\begin{aligned}
\left(1 - \prod_{i=1}^{k+1}(1 - \mu_i)\right)^{k+1} &= \left(1 - \prod_{i=1}^{k+1}(1 - \mu_i)\right)^k \left(1 - \prod_{i=1}^{k+1}(1 - \mu_i)\right) \\
&\geq \left(1 - \prod_{i=1}^{k}(1 - \mu_i)\right)^k (1 - (1 - \mu_{k+1})) \\
&\overset{hyp.}{\geq} \prod_{i=1}^{k} \mu_i \cdot \mu_{k+1} = \prod_{i=1}^{k+1} \mu_i
\end{aligned}$$

△

The claim follows since $\left(1 - \prod_{i=1}^{k}(1 - \mu_i)\right)^2 \geq \left(1 - \prod_{i=1}^{k}(1 - \mu_i)\right)^k$ for $k \geq 2$.

B Proof of Lemma 2

The proof proceeds very similar to the proof of Lemma 1 Observe that n can be factored by sampling $x \leftarrow_r \mathbb{Z}_n$ and computing $\gcd(n, z(x))$ if (but not only if) there exist $p_i^{e_i}$ and $p_j^{e_j}$ dividing n such that $z(x) \equiv 0 \bmod p_i^{e_i}$ and $z(x) \not\equiv 0 \bmod p_j^{e_j}$. Using the notation from the proof of Lemma 1, the success probability of this algorithm can be expressed as

$$\begin{aligned}\gamma''(z) :=& 1 - \prod_{i=1}^{k} \Pr[z(x) \equiv 0 \bmod p_i^{e_i} \mid x \leftarrow_r \mathbb{Z}_n] - \prod_{i=1}^{k} \Pr[z(x) \not\equiv 0 \bmod p_i^{e_i} \mid x \leftarrow_r \mathbb{Z}_n] \\ =& 1 - \prod_{i=1}^{k} \nu(p_i^{e_i}, z) - \prod_{i=1}^{k} (1 - \nu(p_i^{e_i}, z)) .\end{aligned}$$

Note that it holds that $\gamma(z) \geq \gamma''(z)$.

The probability $\pi_2(z)$ can be expressed in terms of $\nu(\cdot, \cdot)$ as

$$\begin{aligned}\pi_2(z) &= \Pr[z(x) \equiv 0 \wedge z(x') \not\equiv 0 \mid x, x' \leftarrow_r J_n] \\ &= \Pr[z(x) \equiv 0 \bmod n \mid x \leftarrow_r J_n] \Pr[z(x) \not\equiv 0 \bmod n \mid x \leftarrow_r J_n] \\ &\leq \Pr[z(x) \equiv 0 \bmod n \mid x \leftarrow_r \mathbb{Z}_n] \Pr[z(x) \not\equiv 0 \bmod n \mid x \leftarrow_r \mathbb{Z}_n] \left(\frac{\mathbb{Z}_n}{|J_n|}\right)^2 \\ &= \Pr[z(x) \equiv 0 \bmod n \mid x \leftarrow_r \mathbb{Z}_n] (1 - \Pr[z(x) \equiv 0 \bmod n \mid x \leftarrow_r \mathbb{Z}_n]) \left(\frac{2n}{\phi(n)}\right)^2 \\ &= \left(\prod_{i=1}^{k} \nu(p_i^{e_i}, z) - \prod_{i=1}^{k} \nu(p_i^{e_i}, z)^2\right) \left(\frac{2n}{\phi(n)}\right)^2\end{aligned}$$

where we use that

$$\Pr[z(x) \equiv 0 \bmod n \mid x \leftarrow_r \mathbb{Z}_n] \geq \Pr[z(x) \equiv 0 \bmod n \mid x \leftarrow_r J_n] \frac{|J_n|}{|\mathbb{Z}_n|}$$

and

$$\Pr[z(x) \not\equiv 0 \bmod n \mid x \leftarrow_r \mathbb{Z}_n] \geq \Pr[z(x) \not\equiv 0 \bmod n \mid x \leftarrow_r J_n] \frac{|J_n|}{|\mathbb{Z}_n|}$$

We prove Lemma 2 by showing that

$$\left(\frac{2n}{\phi(n)}\right)^2 \gamma''(z) \geq \left(\prod_{i=1}^{k} \nu(p_i^{e_i}, z) - \prod_{i=1}^{k} \nu(p_i^{e_i}, z)^2\right) \left(\frac{2n}{\phi(n)}\right)^2 .$$

Note that this inequality is equivalent to

$$1 - \prod_{i=1}^{k} \nu(p_i^{e_i}, z) - \prod_{i=1}^{k} (1 - \nu(p_i^{e_i}, z)) \geq \left(\prod_{i=1}^{k} \nu(p_i^{e_i}, z)\right) \left(1 - \prod_{i=1}^{k} \nu(p_i^{e_i}, z)\right),$$

which in turn is equivalent to

$$\left(1 - \prod_{i=1}^{k} \nu(p_i^{e_i}, z)\right)^2 \geq \prod_{i=1}^{k} (1 - \nu(p_i^{e_i}, z)) .$$

Lemma 4. *For $k \in \mathbb{N}$ and $\mu'_i \in [0,1]$ with $i \in \{1,\dots,k\}$ holds that*

$$\left(1 - \prod_{i=1}^{k} \mu'_i\right)^k \geq \prod_{i=1}^{k} (1 - \mu'_i).$$

Proof. Let $\mu_i := 1 - \mu'_i$ and apply Lemma 3. △

The claim follows since $\left(1 - \prod_{i=1}^{k} \mu_i\right)^2 \geq \left(1 - \prod_{i=1}^{k} \mu_i\right)^k$ for $k \geq 2$.

C Storing and Evaluating Rational Functions.

We have to take care of the way how the polynomials $(N_i, D_i) \in \mathbb{Z}_n[X] \times \mathbb{Z}_n[X]$ corresponding to the rational function $L_i \in \mathsf{L}$ are computed and stored. Consider an oracle representing a polynomial $f \in \mathbb{Z}_n[X]$ by the list of non-zero coefficients of f together with the corresponding degree. For instance, let us assume that the algorithm computes the rational function $L_3 = L_1 + L_2 = (1,1) + (X,1) = (X+1,1)$ in the first query, and then applies a sequence of m repeated squaring operations to L_3. The expansion of the polynomial $(X+1)^m$ consists of 2^m monomials with non-zero coefficients, which grows too fast to be stored efficiently.

However, we assume that the oracle stores the polynomials N_i and D_i as *straight line programs*. That is, the oracle stores the sequence of queries that $\mathcal{A}$ performs to compute the polynomials N_i and D_i for $i \in \{3,\dots,|\mathsf{L}|\}$. Note that each polynomial N_i is computed by performing at most $3m$ operations in $\mathbb{Z}_n[X]$, and each polynomial D_i is computed by performing at most m operations, hence the size of this representation is $O(m)$. Moreover, replacing the wildcard character X by $x \in \mathbb{Z}_n$ and performing the same sequence of operations on 1 and x corresponds to evaluating $N_i(x)$ and $D_i(x)$. Computing $N_i(x)/D_i(x)$ if $D_i(x) \in \mathbb{Z}_n^*$ yields $L_i(x)$. Hence each polynomial can be evaluated by performing $O(m)$ operations in $\mathbb{Z}_n$.