The Generic Hardness of Subset Membership Problems under the Factoring Assumption

Tibor Jager and Jörg Schwenk

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

February 13, 2009

Abstract. We analyze a large class of *subset membership* problems related to integer factorization. We show that there is no algorithm solving these problems efficiently without exploiting properties of the given representation of ring elements, unless factoring integers is easy. Our results imply that problems with high relevance for a large number of cryptographic applications, such as the *quadratic residuosity* and the *subgroup decision* problems, are generically equivalent to factoring.

Keywords: Cryptographic assumptions, subset membership, quadratic residuosity, subgroup decision problem, generic ring algorithms.

1 Introduction

The security of asymmetric cryptographic systems relies 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 (such as the Turing machine model), it is instructive to analyze these assumptions in a restricted, but still meaningful model of computation. A natural and quite general class of algorithms is considered in the *generic ring model*. This model captures all algorithms solving problems defined over an algebraic ring without exploiting specific properties of a given representation of ring elements. Such algorithms work in a similar way for arbitrary representations of ring elements, thus are *generic*.[1] The generic model is useful to study the *intrinsic* hardness of and relationship between fundamental problems of high cryptographic relevance.

Of course a lower complexity bound in the generic model does not immediately imply a lower bound in the standard model, since there may be algorithms solving a given problem significantly faster by exploiting specific properties of the given representation of ring elements. Nevertheless, the analysis of cryptographic assumptions in the generic model is not only of theoretical interest, but also for the design of cryptanalytic algorithms: a lower complexity bound for a certain computational problem in the generic model implies that any algorithm solving the considered problem more efficiently *must* exploit properties of a given representation of ring elements.

1.1 Our Contribution

We analyze a large class of subset membership problems over $\mathbb{Z}_n$. This includes the well-known *quadratic residuosity* and *subgroup decision* problems. We show that solving these problems with generic ring algorithms is as hard as factoring n. This implies that the quadratic residuosity and the subgroup decision problems are generically equivalent to factoring n. For the subgroup decision

[1] See Appendix A for a comparison between the generic *group* and the generic *ring* model

problem we show that the equivalence holds *even if the algorithm has access to an oracle solving the Diffie-Hellman problem*. In contrast to previous work reducing integer factorization to solving a *computational* problem [Bro05, LR06, AJR08, AM08], we reduce factoring to *decisional* problems in $\mathbb{Z}_n$.

We also show that interpreting a proof in the generic ring model as evidence towards the assumption that solving a problem is hard in the standard model has to be done with care. Though this seems self-evident, we are not aware of any non-trivial example in the literature. We provide an example for a practical problem with cryptographic relevance which is easy to solve in the standard model, but provably hard for generic ring algorithms. Concretely, we show that computing the *Jacobi symbol* is hard w.r.t. generic ring algorithms.

We consider 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. Our results hold in the general case where n is the product of *at least* two different odd primes, thus include the classical case where $n = pq$ with p, q prime and $p \neq q$.

1.2 Related Work

Previous work considering fundamental cryptographic assumptions in restricted models of computation was targeted only on the discrete logarithm and the RSA problem. Starting with Shoup's seminal paper [Sho97], it was proven that solving the discrete logarithm problem, the Diffie-Hellman problem, and related problems [MW98, Mau05, RLB+08] are hard with respect to generic *group* algorithms. Damgård and Koprowski showed the generic intractability of root extraction in groups of hidden order [DK02].

Brown [Bro05] reduced the problem of factoring integers to solving the *low-exponent* RSA problem with *straight line programs*, which are a subclass of generic ring algorithms. Leander and Rupp [LR06] 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 [AM08] extended this result from low-exponent RSA to full RSA and to generic ring algorithms that may also compute multiplicative inverses. Boneh and Venkatesan [BV98] have shown that there is no straight line program reducing integer factorization to the low-exponent RSA problem, unless factoring integers is easy.

The notion of generic ring algorithms has also been applied to study the relationship between the discrete logarithm and the Diffie-Hellman problem and the existence of ring-homomorphic encryption schemes [BL96, MR07, AJR08].

2 Preliminaries

2.1 Notation

For a set A and a probability distribution $\mathcal{D}$ on A, we denote with $a \xleftarrow{\mathcal{D}} A$ the action of sampling an element a from A according to distribution $\mathcal{D}$. We denote with U the *uniform* distribution. When sampling k elements $a_1, \ldots, a_k \xleftarrow{\mathcal{D}} A$, we assume that all elements are chosen *independently*.

Throughout the paper we let n be the product of at least two different primes, and denote with $n = \prod_{i=1}^{k} p_i^{e_i}$ the prime factor decomposition of n such that $\gcd(p_i^{e_i}, p_j^{e_j}) = 1$ for $i \neq j$. We denote with $\circ \in \{+, -, \cdot\}$ the binary operators $\mathbb{Z}_n \times \mathbb{Z}_n \to \mathbb{Z}_n$ mapping $(a, b) \mapsto a \circ b \bmod n$, and with $/$ the binary operator $\mathbb{Z}_n \times \mathbb{Z}_n^* \to \mathbb{Z}_n$ mapping $(a, b) \mapsto ab^{-1} \bmod n$. We use the usual infix notation for $+, -, \cdot$ and $/$. We write $a \equiv_n b$ shorthand for $a \equiv b$ modulo n.

Let $P = (S_1, \ldots, S_m)$ be a sequence. Then $|P|$ denotes the length of P, i.e. $|P| = m$. For $k \leq m$ we denote with P_k the subsequence $(S_1, \ldots, S_k)$ of P. For sequences P_k, P with $|P_k| \leq |P|$ we denote with $P_k \sqsubseteq P$ that P_k is a subsequence of P such that P_k consists of the *first* $|P_k|$ elements of P.

2.2 Subset Membership Problems

Definition 1 (Subset Membership Problem). *Let $\mathcal{C} \subseteq \mathbb{Z}_n$ and $\mathcal{V} \subseteq \mathbb{Z}_n$ be subsets of $\mathbb{Z}_n$ such that $\mathcal{V} \subseteq \mathcal{C} \subseteq \mathbb{Z}_n$. The* subset membership problem *defined by $(\mathcal{C}, \mathcal{V})$ is: given $x \xleftarrow{U} \mathcal{C}$, decide whether $x \in \mathcal{V}$.*

Whenever considering a subset membership problem in the following we assume that $|\mathcal{V}| > 1$.

2.3 Uniform Closure

By the Chinese Remainder Theorem, for $n = \prod_{i=1}^{k} p_i^{e_i}$ the ring $\mathbb{Z}_n$ is isomorphic to a direct product of rings $\mathbb{Z}_{p_1^{e_1}} \times \cdots \times \mathbb{Z}_{p_k^{e_k}}$. Let ϕ be the isomorphism $\mathbb{Z}_{p_1^{e_1}} \times \cdots \times \mathbb{Z}_{p_k^{e_k}} \to \mathbb{Z}_n$, and for $\mathcal{C} \subseteq \mathbb{Z}_n$ let $\mathcal{C}_i := \{y \bmod p_i^{e_i} \mid y \in \mathcal{C}\}$ for $1 \leq i \leq k$.

Definition 2 (Uniform Closure). *We say that $\mathcal{U}[\mathcal{C}] \subseteq \mathbb{Z}_n$ is the* uniform closure *of $\mathcal{C} \subseteq \mathbb{Z}_n$, if*

$$\mathcal{U}[\mathcal{C}] = \{y \in \mathbb{Z}_n \mid y = \phi(y_1 \ldots, y_k), y_i \in \mathcal{C}_i \text{ for } 1 \leq i \leq k\}.$$

A simple example is given in Appendix B. In particular note that $\mathcal{C} \subseteq \mathcal{U}[\mathcal{C}]$, but not necessarily $\mathcal{U}[\mathcal{C}] \subseteq \mathcal{C}$. The following lemma follows directly from the above definition.

Lemma 1. *Sampling $y \xleftarrow{U} \mathcal{U}[\mathcal{C}]$ uniformly random from $\mathcal{U}[\mathcal{C}]$ is equivalent to sampling y_i uniformly and independently from $\mathcal{C}_i$ for $1 \leq i \leq k$ and setting $y = \phi(y_1, \ldots, y_k)$.*

2.4 Straight Line Programs

A straight line program over ring R is a generic ring algorithm performing a fixed sequence of operations, without branching, that outputs an element of R. Straight line programs can be seen as a subclass of generic ring algorithms. The following definition is a simple extension of [Bro05, Definition 1] to straight line programs that may also compute multiplicative inverses.

Definition 3 (Straight Line Programs). *A* straight line program *P of length m over $\mathbb{Z}_n$ is a sequence of tuples*

$$P = ((i_1, j_1, \circ_1), \cdots, (i_m, j_m, \circ_m))$$

where $-1 \leq i_k, j_k < k$ and $\circ_i \in \{+, -, \cdot, /\}$ for $i \in \{1, \ldots, m\}$. The output $P(x)$ of straight line program P on input $x \in \mathbb{Z}_n$ is computed as follows.

1. *Initialize $L_{-1} := 1 \in \mathbb{Z}_n$ and $L_0 := x$.*
2. *For k from 1 to m do:*
   - *if $\circ_k = /$ and $L_{j_k} \notin \mathbb{Z}_n^*$ then return $\perp$,*
   - *else set $L_k := L_{i_k} \circ L_{j_k}$.*
3. *Return $P(x) = L_m$.*

We say that each triple $(i, j, \circ) \in P$ is a SLP-step.

For notational convenience, for a given straight line program P we will denote with P_k the straight line program given by the sequence of the first k elements of P, with the additional convention that $P_{-1} = 1$ and $P_0(x) = x$ for all $x \in \mathbb{Z}_n$.

2.5 Some Lemmas on Straight Line Programs over $\mathbb{Z}_n$

In the following we will state a few lemmas on straight line programs over $\mathbb{Z}_n$ that will be useful for the proof of our main theorem.

Lemma 2. *Suppose there exists a straight line program P such that for $x, x' \in \mathbb{Z}_n$ holds that $P(x') \neq \perp$ and $P(x) = \perp$. Then there exists $P_j \sqsubseteq P$ such that $P_j(x') \in \mathbb{Z}_n^*$ and $P_j(x) \notin \mathbb{Z}_n^*$.*

Proof. $P(x) = \perp$ means that there exists an SLP-step $(i, j, \circ) \in P$ such that $\circ = /$ and $L_j = P_j(x) \notin \mathbb{Z}_n^*$. However, $P(x')$ does not evaluate to $\perp$, thus it must hold that $P_j(x') \in \mathbb{Z}_n^*$.

The following lemma provides a lower bound on the probability of factoring n by evaluating a certain straight line program P with $y \xleftarrow{U} \mathcal{U}[\mathcal{C}]$ and computing $\gcd(n, P(y))$, relative to the probability that $P(x') \notin \mathbb{Z}_n^*$ and $P(x) \in \mathbb{Z}_n^*$ for randomly chosen $x, x' \xleftarrow{U} \mathcal{C}$.

Lemma 3. *For any straight line program P and $\mathcal{C} \subseteq \mathbb{Z}_n$ holds that*

$$\Pr\left[P(x') \notin \mathbb{Z}_n^* \text{ and } P(x) \in \mathbb{Z}_n^* \mid x, x' \xleftarrow{U} \mathcal{C}\right] \leq \left(\frac{|\mathcal{U}[\mathcal{C}]|}{|\mathcal{C}|}\right)^2 \Pr\left[\gcd(n, P(y)) \notin \{1, n\} \mid y \xleftarrow{U} \mathcal{U}[\mathcal{C}]\right].$$

Similar to the above, the following lemma provides a lower bound on the probability of factoring n by computing $\gcd(n, S(y) - T(y))$ with $y \xleftarrow{U} \mathcal{C}$ for two given straight line programs P and Q, relative to the probability $\Pr[(P(x) \equiv_n Q(x) \text{ and } P(x') \not\equiv_n Q(x')) \mid x, x' \xleftarrow{U} \mathcal{C}]$.

Lemma 4. *For any pair (P, Q) of straight line programs and $\mathcal{C} \subseteq \mathbb{Z}_n$ holds that*

$$\begin{aligned}&\Pr\left[P(x) \equiv_n Q(x) \text{ and } P(x') \not\equiv_n Q(x') \mid x, x' \xleftarrow{U} \mathcal{C}\right] \\ &\leq \left(\frac{|\mathcal{U}[\mathcal{C}]|}{|\mathcal{C}|}\right)^2 \Pr\left[\gcd(n, P(y) - Q(y)) \notin \{1, n\} \mid y \xleftarrow{U} \mathcal{U}[\mathcal{C}]\right].\end{aligned}$$

Proofs for Lemma 3 and 4 are given in Appendix C and D, respectively. We also discuss the intuition behind these lemmas in Appendix E.

2.6 Generic Ring Algorithms

Similar to straight line programs, generic ring algorithms perform a sequence of ring operations on the input values $1 \in \mathbb{Z}_n$ and x. However, while straight line programs perform the same fixed sequence on ring operations to any input value, generic ring algorithms can decide adaptively which ring operation is performed next. The decision is made either based on equality checks, or on coin tosses. Moreover, the output of generic ring algorithms is not restricted to ring elements.

We formalize the notion of generic ring algorithms in terms of a game between an algorithm $\mathcal{A}$ and a black-box $\mathcal{O}$, the *generic ring oracle*. The generic ring oracle receives as input a secret value $x \in \mathbb{Z}_n$. It maintains a sequence P, which is set to the empty sequence at the beginning of the game, and implements two internal subroutines $\mathsf{test}()$ and $\mathsf{equal}()$.

- The $\mathsf{test}()$-procedure takes as input an tuple $(j, \circ) \in \{-1, \ldots, |P|\} \times \{+, -, \cdot, /\}$. The procedure returns false if $\circ = /$ and $P_j(x) \notin \mathbb{Z}_n^*$, and true otherwise.
- The $\mathsf{equal}()$-procedure takes as input a tuple $(i, j) \in \{-1, \ldots, |P|\} \times \{-1, \ldots, |P|\}$. The procedure returns true if $P_i(x) \equiv P_j(x) \bmod n$ and false otherwise.

In order to perform computations, the algorithm submits SLP-steps to $\mathcal{O}$. Whenever the algorithm submits $(i, j, \circ)$ with $\circ \in \{+, -, \cdot, /\}$, the oracle runs $\mathsf{test}(j, \circ)$. If $\mathsf{test}(j, \circ) = \mathsf{false}$, the oracle returns the error symbol $\perp$. Otherwise $(i, j, \circ)$ is appended to P. Moreover, the algorithm can query the oracle to check for equality of computed ring elements by submitting a query $(i, j, \circ)$ such that $\circ \in \{=\}$. In this case the oracle returns $\mathsf{equal}(i, j)$. We measure the complexity of $\mathcal{A}$ by the number of oracle queries.

3 Subset Membership Problems in the Generic Ring Model

Let $(\mathcal{C}, \mathcal{V})$ be subsets of $\mathbb{Z}_n$ defining a subset membership problem. We formalize the notion of subset membership problems in the generic ring model in terms of a game between an algorithm $\mathcal{A}$ and a generic ring oracle $\mathcal{O}_{\text{smp}}$. Oracle $\mathcal{O}_{\text{smp}}$ is defined exactly like the generic ring oracle described in Section 2.6, except that $\mathcal{O}_{\text{smp}}$ receives a uniformly random element $x \xleftarrow{U} \mathcal{C}$ as input. We say that $\mathcal{A}$ wins the game, if $x \in \mathcal{V}$ and $\mathcal{A}^{\mathcal{O}_{\text{smp}}}(n) = 1$, or $x \notin \mathcal{V}$ and $\mathcal{A}^{\mathcal{O}_{\text{smp}}}(n) = 0$.

Note that any algorithm for a given subset membership problem $(\mathcal{C}, \mathcal{V})$ has at least the trivial success probability $\Pi(\mathcal{C}, \mathcal{V}) := \max\{|\mathcal{V}|/|\mathcal{C}|, 1 - |\mathcal{V}|/|\mathcal{C}|\}$ by guessing, due to the fact that x is sampled uniformly from $\mathcal{C}$. For an algorithm solving the subset membership problem given by $(\mathcal{C}, \mathcal{V})$ with success probability $\Pr[\mathcal{S}]$, we denote with

$$\mathsf{Adv}_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{\text{smp}}}(n)) := |\Pr[\mathcal{S}] - \Pi(\mathcal{C}, \mathcal{V})|$$

the *advantage* of $\mathcal{A}$.

Theorem 1. *For any generic ring algorithm $\mathcal{A}$ solving a given subset membership problem $(\mathcal{C}, \mathcal{V})$ over $\mathbb{Z}_n$ with advantage $\mathsf{Adv}_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{\text{smp}}}(n))$ by performing m queries to $\mathcal{O}_{\text{smp}}$, there exists an algorithm $\mathcal{B}$ that outputs a factor of n with success probability at least*

$$\frac{\mathsf{Adv}_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{\text{smp}}}(n))}{2m(m^2+5m+3)} \cdot \left(\frac{|\mathcal{C}|}{|\mathcal{U}[\mathcal{C}]|}\right)^2$$

by running $\mathcal{A}$ once and performing $O(m^3)$ additional operations in $\mathbb{Z}_n$, m gcd-computations on $\lceil \log_2 n \rceil$-bit numbers, and sampling m random elements from $\mathcal{U}[\mathcal{C}]$.

Proof Outline. We replace $\mathcal{O}_{\text{smp}}$ with a simulator $\mathcal{O}_{\text{sim}}$. Let $\mathcal{S}_{\text{sim}}$ denote the event that $\mathcal{A}$ is successful when interacting with the simulator, and let $\mathcal{F}$ denote the event that $\mathcal{O}_{\text{sim}}$ answers a query of $\mathcal{A}$ different from how $\mathcal{O}_{\text{smp}}$ would have answered. Then $\mathcal{O}_{\text{smp}}$ and $\mathcal{O}_{\text{sim}}$ are indistinguishable unless $\mathcal{F}$ occurs. Therefore the success probability $\Pr[\mathcal{S}]$ of $\mathcal{A}$ in the simulation game is upper bound by $\Pr[\mathcal{S}_{\text{sim}}] + \Pr[\mathcal{F}]$ (cf. the Difference Lemma [Sho06, Lemma 1]). We derive a bound on $\Pr[\mathcal{S}_{\text{sim}}]$ and describe a factoring algorithm whose success probability is lower bound by $\Pr[\mathcal{F}]$.

3.1 Introducing a Simulation Oracle

We replace oracle $\mathcal{O}_{\text{smp}}$ with a simulator $\mathcal{O}_{\text{sim}}$. $\mathcal{O}_{\text{sim}}$ receives $x \xleftarrow{U} \mathcal{C}$ as input, but never uses this value throughout the game. Instead, all computations are performed *independent* of the challenge value x. Note that the original oracle $\mathcal{O}_{\text{smp}}$ uses x only inside the $\mathsf{test}()$ and $\mathsf{equal}()$ procedures. Let us therefore consider an oracle $\mathcal{O}_{\text{sim}}$ which is defined exactly like $\mathcal{O}_{\text{smp}}$, but replaces the procedures $\mathsf{test}()$ and $\mathsf{equal}()$ with procedures $\mathsf{testsim}()$ and $\mathsf{equalsim}()$.

- The $\mathsf{testsim}()$-procedure samples $x_r \xleftarrow{U} \mathcal{C}$ and returns false if $\circ = /$ and $P_j(x_r) \notin \mathbb{Z}_n^*$, and true otherwise (even if $P_j(x_r) = \perp$).
- The $\mathsf{equalsim}()$-procedure samples $x_r \xleftarrow{U} \mathcal{C}$ and returns true if $P_i(x_r) \equiv P_j(x_r) \bmod n$ and false otherwise (even if $P_i(x_r) = \perp$ or $P_j(x_r) = \perp$).

Note that the simulator samples m random values x_r, $r \in \{1, \ldots, m\}$. Also note that all computations of $\mathcal{A}$ are independent of the challenge value x when interacting with $\mathcal{O}_{\text{sim}}$. Hence, any algorithm $\mathcal{A}$ has at most trivial success probability in the simulation game, and therefore

$$\Pr[\mathcal{S}_{\text{sim}}] \leq \Pi(\mathcal{C}, \mathcal{V}).$$

3.2 Bounding the Probability of Simulation Failure

We say that a *simulation failure*, denoted $\mathcal{F}$, occurs if $\mathcal{O}_{\text{sim}}$ does not simulate $\mathcal{O}_{\text{smp}}$ perfectly. Observe that an interaction of $\mathcal{A}$ with $\mathcal{O}_{\text{sim}}$ is perfectly indistinguishable from an interaction with $\mathcal{O}_{\text{smp}}$, unless at least one of the following events occurs.

1. The testsim()-procedure fails to simulate test() perfectly. This means that testsim() returns false on a procedure call where test() would have returned true, or testsim() returns true where test() would have returned false. Let $\mathcal{F}_{\mathsf{test}}$ denote the event that this happens on at least one call of testsim().
2. The equalsim()-procedure fails to simulate equal() perfectly. This means that equalsim() has returned true where equal() would have returned false, or equalsim() has returned false where equal() would have returned true. Let $\mathcal{F}_{\mathsf{equal}}$ denote the event that this happens at at least one call of equalsim().

Since $\mathcal{F}$ implies that at least one of the events $\mathcal{F}_{\mathsf{test}}$ and $\mathcal{F}_{\mathsf{equal}}$ has occurred, it holds that

$$\Pr[\mathcal{F}] \leq \Pr[\mathcal{F}_{\mathsf{test}}] + \Pr[\mathcal{F}_{\mathsf{equal}}].$$

In the following we will bound $\Pr[\mathcal{F}_{\mathsf{test}}]$ and $\Pr[\mathcal{F}_{\mathsf{equal}}]$ separately.

Bounding the Probability of $\mathcal{F}_{\mathsf{test}}$. The testsim()-procedure fails to simulate test() only if either testsim() has returned false where test() would have returned true, or testsim() has returned true where test() would have returned false. A necessary condition[2] for this is that there exists $P_j \sqsubseteq P$ and $x_r \in \{x_1, \ldots, x_m\}$ such that

$$(P_j(x) \in \mathbb{Z}_n^* \text{ and } P_j(x_r) \notin \mathbb{Z}_n^*) \text{ or } (P_j(x) = \perp \text{ and } P_j(x_r) \notin \mathbb{Z}_n^*),$$

or

$$(P_j(x_r) \in \mathbb{Z}_n^* \text{ and } P_j(x) \notin \mathbb{Z}_n^*) \text{ or } (P_j(x_r) = \perp \text{ and } P_j(x) \notin \mathbb{Z}_n^*).$$

We can simplify this condition a little by applying Lemma 2. The existence of $P_j \sqsubseteq P$ and x_r such that $(P_j(x_r) = \perp \text{ and } P_j(x) \notin \mathbb{Z}_n^*)$ implies the existence of $P_k \sqsubseteq P$ such that $k < j$ and $(P_k(x_r) \notin \mathbb{Z}_n^*$ and $P_k(x) \in \mathbb{Z}_n^*)$. An analogous argument holds for the case $(P_j(x) = \perp \text{ and } P_j(x_r) \notin \mathbb{Z}_n^*)$. Hence, testsim()-procedure fails to simulate test() only if there exists $P_j \sqsubseteq P$ such that

$$(P_j(x) \in \mathbb{Z}_n^* \text{ and } P_j(x_r) \notin \mathbb{Z}_n^*) \text{ or } (P_j(x_r) \in \mathbb{Z}_n^* \text{ and } P_j(x) \notin \mathbb{Z}_n^*).$$

Proposition 1.

$$\Pr[\mathcal{F}_{\mathsf{test}}] \leq 2m(m+2) \max_{0 \leq j \leq m} \left\{ \Pr\left[P_j(x) \notin \mathbb{Z}_n^* \textit{ and } P_j(x') \in \mathbb{Z}_n^* \mid x, x' \xleftarrow{U} \mathcal{C} \right] \right\}$$

We prove Proposition 1 in Appendix F.

Bounding the Probability of $\mathcal{F}_{\mathsf{equal}}$ The equalsim()-procedure fails to simulate equal() only if either equalsim() has returned false where equal() would have returned true, or equalsim() has returned true where equal() would have returned false. A necessary[3] condition for this is that there exist $P_i, P_j \sqsubseteq P$ and $x_r \in \{x_1, \ldots, x_m\}$ such that

$$(P_i(x) \equiv_n P_j(x) \text{ and } P_i(x_r) \not\equiv_n P_j(x_r)) \text{ or } (P_i(x) \equiv_n P_j(x) \text{ and } (P_i(x_r) = \perp \text{ or } P_j(x_r) = \perp))$$

[2] The condition is not sufficient, because algorithm $\mathcal{A}$ need not have queried a division by P_j in its r-th query.
[3] The condition is not sufficient, because algorithm $\mathcal{A}$ need not have queried $(i, j, =)$ in its r-th query.

or

$$(P_i(x_r) \equiv_n P_j(x_r) \text{ and } P_i(x) \not\equiv_n P_j(x)) \text{ or } (P_i(x_r) \equiv_n P_j(x_r) \text{ and } (P_i(x) = \perp \text{ or } P_j(x) = \perp)).$$

Again we can apply Lemma 2 to simplify this a little: the existence of $P_j \in P$ and x_r such that $(P_j(x_r) = \perp$ and $P_j(x) \neq \perp)$ implies the existence of $P_k \in P$ such that $(P_k(x_r) \notin \mathbb{Z}_n^*$ and $P_k(x) \in \mathbb{Z}_n^*)$. Analogous arguments hold for the other cases where one straight line program evaluates to $\perp$. Hence, $\mathsf{equalsim}()$-procedure fails to simulate $\mathsf{equal}()$ only if there exist $P_i, P_j \sqsubseteq P$ or $P_k \sqsubseteq P$ such that

$$(P_i(x) \equiv_n P_j(x) \text{ and } P_i(x_r) \not\equiv_n P_j(x_r)) \text{ or } (P_i(x_r) \equiv_n P_j(x_r) \text{ and } P_i(x) \not\equiv_n P_j(x))$$

or

$$(P_k(x_r) \notin \mathbb{Z}_n^* \text{ and } P_k(x) \in \mathbb{Z}_n^*) \text{ or } (P_k(x) \notin \mathbb{Z}_n^* \text{ and } P_k(x_r) \in \mathbb{Z}_n^*).$$

Proposition 2.

$$\begin{aligned}\Pr[\mathcal{F}_{\mathsf{equal}}] \leq\ & 2m(m^2+3m+1) \max_{-1 \leq i < j \leq m} \left\{ \Pr\left[P_i(x) \equiv_n P_j(x) \textit{ and } P_i(x') \not\equiv_n P_j(x') \mid x, x' \xleftarrow{U} \mathcal{C} \right] \right\} \\ & + 2m(m+1) \max_{0 \leq k \leq m} \left\{ \Pr\left[P_k(x) \notin \mathbb{Z}_n^* \textit{ and } P_k(x') \in \mathbb{Z}_n^* \mid x, x' \xleftarrow{U} \mathcal{C} \right] \right\}\end{aligned}$$

Proposition 2 is proven in Appendix G.

Bounding the Probability of $\mathcal{F}$. Summing up, we obtain that the total probability of $\mathcal{F}$ is at most

$$\begin{aligned}\Pr[\mathcal{F}] &\leq \Pr[\mathcal{F}_{\mathsf{test}}] + \Pr[\mathcal{F}_{\mathsf{equal}}] \\ &\leq 2m(m^2+3m+1) \max_{-1 \leq i < j \leq m} \left\{ \Pr\left[P_i(x) \equiv_n P_j(x) \text{ and } P_i(x') \not\equiv_n P_j(x') \mid x, x' \xleftarrow{U} \mathcal{C} \right] \right\} \\ &\quad + 4m(m+1) \max_{0 \leq k \leq m} \left\{ \Pr\left[P_k(x) \notin \mathbb{Z}_n^* \text{ and } P_k(x') \in \mathbb{Z}_n^* \mid x, x' \xleftarrow{U} \mathcal{C} \right] \right\}.\end{aligned}$$

3.3 Bounding the Success Probability of any Generic Ring Algorithm

Since all computations of $\mathcal{A}$ are independent of the challenge value x in the simulation game, any algorithm has only the trivial success probability when interacting with the simulator. Thus the success probability of any algorithm when interacting with the original oracle is bound by

$$\Pi(\mathcal{C}, \mathcal{V}) + \mathsf{Adv}_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{\mathrm{smp}}}) = \Pr[\mathcal{S}] \leq \Pr[\mathcal{S}_{\mathrm{sim}}] + \Pr[\mathcal{F}] \leq \Pi(\mathcal{C}, \mathcal{V}) + \Pr[\mathcal{F}],$$

which implies

$$\mathsf{Adv}_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{\mathrm{smp}}}) \leq \Pr[\mathcal{F}].$$

3.4 The Factoring Algorithm

Consider a factoring algorithm $\mathcal{B}$ running $\mathcal{A}$, recording the sequence of queries $\mathcal{A}$ issues, and proceeding as follows.

- Whenever the algorithm submits $(i, j, \circ)$ with $\circ \in \{+, -, \cdot, /\}$ in its r-th query, the algorithm samples $y \xleftarrow{U} \mathcal{U}[\mathcal{C}]$ and computes $\gcd(P_k(y), n)$ for $0 \leq k \leq r$.
- Whenever the algorithm submits $(i, j, \circ)$ with $\circ \in \{=\}$ in its r-th query, the algorithm samples $y \xleftarrow{U} \mathcal{U}[\mathcal{C}]$ and computes $\gcd(P_k(y) - P_l(y), n)$ for $-1 \leq k < l \leq r$.

Running time. By assumption, $\mathcal{A}$ submits m queries. Thus, the algorithm evaluates $O(m^2)$ straight line programs. Each query can be evaluated by performing at most m steps, which yields $O(m^3)$ operations in $\mathbb{Z}_n$. Moreover, the algorithm samples m random values y from $\mathcal{U}[\mathcal{C}]$ and performs m gcd-computations on $\lceil \log_2 n \rceil$-bit numbers.

Success probability. $\mathcal{B}$ evaluates any straight line program P_k with a uniformly random element y of $\mathcal{U}[\mathcal{C}]$. In particular, $\mathcal{B}$ computes $\gcd(P_k(y), n)$ for $y \xleftarrow{U} \mathcal{U}[\mathcal{C}]$ and the straight line program $P_k \sqsubseteq P$ satisfying

$$\begin{aligned} &\Pr\left[P_k(x) \notin \mathbb{Z}_n^* \text{ and } P_k(x') \in \mathbb{Z}_n^* \mid x, x' \xleftarrow{U} \mathcal{C}\right] \\ &= \max_{0 \le k \le m} \left\{\Pr\left[P_k(x) \notin \mathbb{Z}_n^* \text{ and } P_k(x') \in \mathbb{Z}_n^* \mid x, x' \xleftarrow{U} \mathcal{C}\right]\right\}. \end{aligned}$$

Let $\gamma_1 := \max_{0 \le k \le m}\{\Pr[P_k(x) \notin \mathbb{Z}_n^* \text{ and } P_k(x') \in \mathbb{Z}_n^* \mid x, x' \xleftarrow{U} \mathcal{C}]\}$, then by Lemma 3 algorithm $\mathcal{B}$ finds a factor in this step with probability at least $\gamma_1 \left(\frac{|\mathcal{C}|}{|\mathcal{U}[\mathcal{C}]|}\right)^2$.

Moreover, $\mathcal{B}$ evaluates any pair P_i, P_j of straight line programs in P with a uniformly random element $y \xleftarrow{U} \mathcal{U}[\mathcal{C}]$. So in particular $\mathcal{B}$ evaluates $\gcd(P_i(y) - P_j(y), n)$ with $y \xleftarrow{U} \mathcal{U}[\mathcal{C}]$ for the pair of straight line programs $P_i, P_j \sqsubseteq P$ satisfying

$$\begin{aligned} &\Pr\left[P_i(x) \equiv_n P_j(x) \text{ and } P_i(x') \not\equiv_n P_j(x') \mid x, x' \xleftarrow{U} \mathcal{C}\right] \\ &= \max_{-1 \le i < j \le m} \left\{\Pr\left[P_i(x) \equiv_n P_j(x) \text{ and } P_i(x') \not\equiv_n P_j(x') \mid x, x' \xleftarrow{U} \mathcal{C}\right]\right\}. \end{aligned}$$

Let $\gamma_2 := \max_{-1 \le i < j \le m}\{\Pr[P_i(x) \equiv_n P_j(x) \text{ and } P_i(x') \not\equiv_n P_j(x') \mid x, x' \xleftarrow{U} \mathcal{C}]\}$, then by Lemma 4 algorithm $\mathcal{B}$ succeeds in this step with probability at least $\gamma_2 \left(\frac{|\mathcal{C}|}{|\mathcal{U}[\mathcal{C}]|}\right)^2$.

So, for $\gamma := \max\{\gamma_1, \gamma_2\}$, the total success probability of algorithm $\mathcal{B}$ is at least

$$\gamma \left(\frac{|\mathcal{C}|}{|\mathcal{U}[\mathcal{C}]|}\right)^2.$$

Relating the success probability of $\mathcal{B}$ to the advantage of $\mathcal{A}$. Using the above definitions of γ_1, γ_2, and γ, the fact that $\mathsf{Adv}_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{\mathrm{smp}}}(n)) \le \Pr[\mathcal{F}]$, and the derived bound on $\Pr[\mathcal{F}]$, we can obtain a lower bound on γ by

$$\mathsf{Adv}_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{\mathrm{smp}}}(n)) \le \Pr[\mathcal{F}] \le 4m(m+1)\gamma_1 + 2m(m^2+3m+1)\gamma_2 \le 2m(m^2+5m+3)\gamma,$$

which implies the inequality

$$\gamma \ge \frac{\mathsf{Adv}_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{\mathrm{smp}}}(n))}{2m(m^2+5m+3)}.$$

Therefore the success probability of $\mathcal{B}$ is at least

$$\frac{\mathsf{Adv}_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{\mathrm{smp}}}(n))}{2m(m^2+5m+3)} \cdot \left(\frac{|\mathcal{C}|}{|\mathcal{U}[\mathcal{C}]|}\right)^2.$$

4 The Generic Quadratic Residuosity Problem and Factoring

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

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

Let $(x \mid n)$ denote the *Jacobi symbol* [Sho05, p.287] and 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. Recall that $\mathsf{QR}_n \subseteq J_n$, and therefore given $x \in \mathbb{Z}_n \backslash J_n$ it is easy to decide that x is not a quadratic residue by computing the Jacobi symbol.

Definition 4 (Quadratic Residuosity Problem). *The* quadratic residuosity problem *is the subset membership problem given by $\mathcal{C} = J_n$ and $\mathcal{V} = \mathsf{QR}_n$.*

4.1 The Generic Quadratic Residuosity Problem is Equivalent to Factoring

Given the factorization of n, solving the quadratic residuosity problem in $\mathbb{Z}_n$ is easy, also for generic ring algorithms. Thus, in order to show the equivalence of generic quadratic residuosity and factoring, we have to prove the following theorem.

Theorem 2. *Suppose there exist a generic ring algorithm $\mathcal{A}$ solving the quadratic residuosity problem in $\mathbb{Z}_n$ with advantage $\mathsf{Adv}_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{\mathrm{smp}}}(n))$ by performing m ring operations. Then there exists an algorithm $\mathcal{B}$ finding a factor of n with probability at least*

$$\frac{\mathsf{Adv}_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{\mathrm{smp}}}(n))}{8m(m^2+5m+3)}$$

by running $\mathcal{A}$ once and performing $O(m^3)$ additional operations in $\mathbb{Z}_n$, m gcd-computations on $\lceil \log_2 n \rceil$-bit numbers, and sampling m random elements from $\mathbb{Z}_n^*$.

Proof. 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 [Sho05, p.24]. Note also that $\mathcal{U}[J_n] = \mathcal{U}[\mathcal{C}] = \mathbb{Z}_n^*$. Therefore it holds that $|J_n| = |\mathcal{C}| \geq \phi(n)/2$ and $|\mathcal{U}[\mathcal{C}]| = |\mathbb{Z}_n^*| = \phi(n)$. Thus we can apply Theorem 1, using that

$$\left(\frac{|\mathcal{C}|}{|\mathcal{U}[\mathcal{C}]|}\right)^2 = \left(\frac{|J_n|}{|\mathbb{Z}_n^*|}\right)^2 \geq \left(\frac{\phi(n)/2}{\phi(n)}\right)^2 = \frac{1}{4}.$$

4.2 Computing the Jacobi Symbol with Generic Ring Algorithms

Showing that solving a certain subset membership problem with generic ring algorithms is at least as hard as factoring n may be considered as evidence towards the conjecture that solving the considered problem is also hard in a general model of computation (such as the Turing machine model). While it is true that a proof in the generic model shows that a cryptographic hardness assumption is not totally wrong, in the sense that the problem is not easy to solve for a restricted but meaningful class of algorithms, interpreting a proof in the generic ring model as evidence for the standard model has to be done with care. For the generic *group* model there exists a number of examples of problems which are hard in the generic model, but significantly easier without the

restriction to generic algorithms. Though it seems self-evident that such problems exist in the generic *ring* model as well, we are not aware of any non-trivial example in the literature.

In the following we provide such an example for a problem which is closely related to the quadratic residuosity problem, namely the problem of computing the Jacobi symbol. There exist simple efficient algorithms computing the Jacobi symbol that are not generic, cf. [Sho05, p.288]. However, let us consider the subset membership problem $(\mathcal{C}, \mathcal{V})$ with $\mathcal{C} = \mathbb{Z}_n^*$ and $\mathcal{V} = J_n$. The following theorem states that there is no efficient *generic* algorithm solving this problem, unless factoring n is easy.

Theorem 3. *Suppose there exist a generic ring algorithm $\mathcal{A}$ solving the subset membership problem given by $(\mathcal{C}, \mathcal{V})$ with $\mathcal{C} = \mathbb{Z}_n^*$ and $\mathcal{V} = J_n$ with advantage $\mathsf{Adv}_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{\mathrm{smp}}}(n))$ by performing m ring operations. Then there exists an algorithm $\mathcal{B}$ finding a factor of n with probability at least*

$$\frac{\mathsf{Adv}_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{\mathrm{smp}}}(n))}{2m(m^2+5m+3)}$$

by running $\mathcal{A}$ once and performing $O(m^3)$ additional operations in $\mathbb{Z}_n$, m gcd-computations on $\lceil \log_2 n \rceil$-bit numbers, and sampling m random elements from $\mathbb{Z}_n^*$.

Proof. Again we can apply Theorem 1, this time using that $\mathcal{U}[\mathbb{Z}_n^*] = \mathbb{Z}_n^*$, and

$$\left(\frac{|\mathcal{C}|}{|\mathcal{U}[\mathcal{C}]|}\right)^2 = \left(\frac{|\mathbb{Z}_n^*|}{|\mathbb{Z}_n^*|}\right)^2 = 1$$

5 The Generic Subgroup Decision Problem and Factoring

Let $n = pq$ and let $\mathbb{G}$ be a cyclic group of order n. Then there exists a subgroup $\mathbb{G}_p \subseteq \mathbb{G}$ of order p.

Definition 5 (Subgroup Decision Problem). *The* subgroup decision problem *is the subset membership problem $(\mathcal{C}, \mathcal{V})$ with $\mathcal{C} = \mathbb{G}$ and $\mathcal{V} = \mathbb{G}_p$.*

Clearly solving the subgroup membership problem is easy if the factorization of n is given. In the following we will show that solving the subgroup membership problem is *equivalent* to factoring n with respect to generic algorithms, *even if the algorithm has access to an oracle solving the Diffie-Hellman problem in* $\mathbb{G}$. We are not able to apply the framework described in Section 3 directly, because we had to require that the challenge is sampled *uniformly* from $\mathcal{C}$. Therefore we introduce a different technique that is more specific, but works for challenges chosen according to an distribution $\mathcal{D}$ such that $\Pr[x \in \mathcal{V} \mid x \xleftarrow{\mathcal{D}} \mathcal{C}] \approx 1/2$.

5.1 The Subgroup Decision Problem in the Generic Model

Recall that any cyclic group of order n is isomorphic to the additive group of integers $(\mathbb{Z}_n, +)$. Now, since we are going to consider *generic* algorithms, we may assume that the algorithm operates on the group $\mathbb{G} = (\mathbb{Z}_n, +)$, of course without exploiting any property of this representation.[4] Assuming an oracle DH solving the Diffie-Hellman problem in $\mathbb{G}$, we observe that this operation corresponds to the *multiplication* in $\mathbb{Z}_n$. Hence, the group $\mathbb{G}$ together with oracle DH exhibits the same algebraic structure as the *ring* $\mathbb{Z}_n$.

[4] Technically, we assume that the *generic group oracle* uses the group $(\mathbb{Z}_n, +)$ for the *internal* representation of group elements.

By the Chinese Remainder Theorem, the ring $\mathbb{Z}_n$ is isomorphic to the direct product $\mathbb{Z}_p \times \mathbb{Z}_q$. Let $\phi : \mathbb{Z}_p \times \mathbb{Z}_q \to \mathbb{Z}_n$ denote this isomorphism. The subgroup $\mathbb{G}_p$ of $\mathbb{G}$ with order p consists of the elements $\mathbb{G}_p = \{\phi(x_p, 0) \mid x_p \in \mathbb{Z}_p\}$. So for generic ring algorithms the subgroup decision problem can be stated as: given $x \in \mathbb{Z}_n$, decide whether $x \equiv 0 \bmod q$.

In order to model the generic subgroup decision problem, consider an oracle $\mathcal{O}_{\text{sdp}}$ which is defined exactly like the generic ring oracle described in Section 2.6, except that it does not provide the operation /. $\mathcal{O}_{\text{sdp}}$ receives an element $x \in \mathbb{Z}_n$ as input, where x is constructed as follows: sample $(x_p, x_q) \xleftarrow{U} \mathbb{Z}_p \times \mathbb{Z}_q$ and bit $b \xleftarrow{U} \{0, 1\}$ uniformly random, and let $x := \phi(x_p, bx_q)$. An algorithm can query the oracle for the (inverse) group operation by submitting a query $(i, j, \circ)$ with $\circ \in \{+, -\}$. The Diffie-Hellman oracle is queried by submitting $(i, j, \circ)$ with $\circ \in \{\cdot\}$.

We say that the algorithm wins the game, if $x \in \mathbb{G}_p$ and $\mathcal{A}^{\mathcal{O}_{\text{sdp}}}(n) = 1$, or $x \notin \mathbb{G}_p$ and $\mathcal{A}^{\mathcal{O}_{\text{sdp}}}(n) = 0$. We define the *advantage* of an algorithm $\mathcal{A}$ solving the subgroup decision problem with probability $\Pr[\mathcal{S}]$ as

$$\mathsf{Adv}(\mathcal{A}^{\mathcal{O}_{\text{sdp}}}(n)) := \left| \Pr[\mathcal{S}] - \left(\frac{1}{2} + \frac{1}{q} \right) \right|.$$

Remark 1. If we would also allow to query the oracle for operation / (which corresponds to an "*inverse* Diffie-Hellman oracle" in the above setting), then there would be a simple algorithm determining whether $x \in \mathbb{G}_p$ by returning true iff division by x fails. Interestingly, we will show that there is *no* generic algorithm making similar use of a *standard* Diffie-Hellman oracle, unless factoring n is easy. Therefore a further consequence of the theorem presented in the following section is that a standard Diffie-Hellman oracle does not imply a inverse Diffie-Hellman oracle in general, unless factoring is easy.

Remark 2. The subgroup decision problem was introduced in [BGN05] for groups with bilinear pairing. Essentially such a pairing can be added to the generic model by allowing the algorithm to perform a single multiplication operation,[5] as done in [BB08]. By providing a Diffie-Hellman oracle, we do not restrict the algorithm to a fixed number of multiplications. Hence, our proof includes the problem stated in [BGN05] as the special case where only a single multiplication is allowed.

5.2 The Subgroup Decision Problem is Generically Equivalent to Factoring

It is easy to see that there exists a generic algorithm solving the subgroup decision problem, if the factorization of the group order is known. In order to show the equivalence of the generic subgroup decision and factoring, it remains to reduce factoring integers to the generic subgroup decision problem. In the sequel we show that solving the subgroup decision problem in groups of order n is as hard as factoring n, even if the algorithm has access to an oracle solving the Diffie-Hellman problem.

Theorem 4. *Suppose there exist a generic ring algorithm $\mathcal{A}$ solving the subgroup membership problem in $\mathbb{G}$ with advantage $\mathsf{Adv}(\mathcal{A}^{\mathcal{O}_{\text{sdp}}}(n))$ by making m queries to an oracle performing the (inverse) group operation and solving the Diffie-Hellman problem. Then there exists an algorithm $\mathcal{B}$ finding a factor of n with probability at least $\mathsf{Adv}(\mathcal{A}^{\mathcal{O}_{\text{sdp}}}(n))$ by running $\mathcal{A}$ once and performing $O(m^3)$ additional operations in $\mathbb{Z}_n$ and m gcd-computations on $\lceil \log_2 n \rceil$-bit numbers.*

Proof. Let us consider an interaction of $\mathcal{A}$ with an oracle $\mathcal{O}_p$ which is defined as follows. $\mathcal{O}_p$ works similar to $\mathcal{O}_{\text{sdp}}$, but performs all computations in $\mathbb{Z}_p$. That is, the $\mathsf{equal}()$-procedure returns true on input (i, j) iff $P_i(x) \equiv P_j(x) \bmod p$. Note that now all computations are performed in the

[5] Plus some minor technical details to distinguish between different groups.

$\mathbb{Z}_p$-component of the decomposition $\mathbb{Z}_p \times \mathbb{Z}_q$ of $\mathbb{Z}_n$, hence the algorithm receives no information on whether $x \equiv 0 \bmod q$. Thus in the simulation game any algorithm has only trivial success probability $\Pr[\mathcal{S}_{\text{sim}}] = 1/2 + 1/q$.

Now consider an interaction of $\mathcal{A}$ with oracle $\mathcal{O}_{\text{sdp}}$. Either this interaction is indistinguishable from an oracle $\mathcal{O}_p$, in which case the algorithm has only trivial success probability, or there exist $P_i, P_j \sqsubseteq P$ with such that $P_i(x) \equiv P_j(x) \bmod p$, but $P_i(x) \not\equiv P_j(x) \bmod n$. In this case a factor of n is found by computing $\gcd(P_i(x) - P_j(x), n)$. Note that

$$\frac{1}{2} + \mathsf{Adv}_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{\text{sdp}}}(n)) \geq \Pr[\mathcal{S}_{\text{sim}}] + \Pr[\mathcal{F}] \iff \mathsf{Adv}_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{\text{sdp}}}(n)) \geq \Pr[\mathcal{F}]$$

Thus, n is factored this way by running $\mathcal{A}$, recording P and computing $\gcd(P_i(x) - P_j(x), n)$ for all $-1 \leq i < j \leq m$ with probability at least $\mathsf{Adv}_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{\text{sdp}}}(n))$.

The above proof generalizes from $n = pq$ to $n = \prod_{i=1}^{k} p_i^{e_i}$ for all subgroups with prime-power order $p_i^{e_i}$ in a straightforward manner.

6 On the Analysis of Computational Problems in the Generic Ring Model

In Section 3 we have constructed a simulator for a generic ring oracle for the ring $\mathbb{Z}_n$. When interacting with the simulator, all computations are independent of the secret challenge value x. Therefore we have been able to conclude that any generic algorithm has only the trivial probability of success in solving certain decisional problems (namely the considered subset membership problems) when interacting with the simulator. Moreover, we have shown that any algorithm that can distinguish between simulator and original oracle can be turned into a factoring algorithm with (asymptotically) the same running time.

In contrast to *decisional* problems, where the algorithm outputs a bit, our construction of the simulator can also be applied to prove the generic hardness of *computational* problems where the algorithm outputs a ring element. Let us sketch two possibilities. One is to formulate a suitable subset membership problem which reduces to the considered computational problem and then apply Theorem 1. Another possibility is to use our construction of the simulator to bound the probability of a simulation failure relative to factoring. In order to bound the success probability in the simulation game, it remains to show that there exists no *straight line program* solving the considered problem efficiently under the factoring assumption. Hence, under the assumption that factoring n is hard, there exists an efficient generic ring algorithm for a certain computational (in the sense that the algorithm has to *compute* a certain ring element) problem over $\mathbb{Z}_n$ *if and only if* there exists an efficient straight line program. This was proven independently (and in a different way) by Aggarwal and Maurer [AM08, Lemma 7].

7 Conclusion and Open Problems

We have shown that generic ring algorithms are not able to solve certain subset membership problems over $\mathbb{Z}_n$ without essentially factoring n. Moreover, we gave an example for a very practical problem which is hard in the generic ring model, but easy in general. Remarkably, in contrast to previous work we have reduced integer factorization to solving *decisional* problems over $\mathbb{Z}_n$.

Though our results cover a large class of problems with high cryptographic relevance, there are still problems where the generic equivalence to factoring is an open question. One such problem is, for instance, Paillier's *decisional composite residuosity* problem [Pai99]. We consider it as an interesting question whether it is possible to prove the assumption that this problem is (generically) equivalent to factoring.

Acknowledgements We would like to thank Andy Rupp and Sven Schäge for helpful discussions, and the program committee members of Eurocrypt 2009 for valuable comments.

References

[AJR08] Kristina Altmann, Tibor Jager, and Andy Rupp. On black-box ring extraction and integer factorization. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP (2)*, volume 5126 of *Lecture Notes in Computer Science*, pages 437–448. Springer, 2008.

[AM08] Divesh Aggarwal and Ueli Maurer. Factoring is equivalent to generic RSA. Cryptology ePrint Archive, Report 2008/260, 2008. http://eprint.iacr.org/ (to be published at *EUROCRYPT 2009)*.

[BB08] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *J. Cryptology*, 21(2):149–177, 2008.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In Joe Kilian, editor, *TCC*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–341. Springer, 2005.

[BL96] Dan Boneh and Richard J. Lipton. Algorithms for black-box fields and their application to cryptography (extended abstract). In Neal Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 283–297. Springer, 1996.

[Bro05] Daniel R. L. Brown. Breaking RSA may be as difficult as factoring. Cryptology ePrint Archive, Report 2005/380, 2005. http://eprint.iacr.org/.

[BV98] Dan Boneh and Ramarathnam Venkatesan. Breaking RSA may not be equivalent to factoring. In Kaisa Nyberg, editor, *EUROCRYPT*, volume 1403 of *Lecture Notes in Computer Science*, pages 59–71. Springer, 1998.

[DK02] Ivan Damgård and Maciej Koprowski. Generic lower bounds for root extraction and signature schemes in general groups. In Lars R. Knudsen, editor, *EUROCRYPT*, volume 2332 of *Lecture Notes in Computer Science*, pages 256–271. Springer, 2002.

[LR06] Gregor Leander and Andy Rupp. On the equivalence of RSA and factoring regarding generic ring algorithms. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT*, volume 4284 of *Lecture Notes in Computer Science*, pages 241–251. Springer, 2006.

[Mau05] Ueli M. Maurer. Abstract models of computation in cryptography. In Nigel P. Smart, editor, *IMA Int. Conf.*, volume 3796 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2005.

[MR07] Ueli Maurer and Dominik Raub. Black-box extension fields and the inexistence of field-homomorphic one-way permutations. In Kaoru Kurosawa, editor, *ASIACRYPT*, volume 4833 of *Lecture Notes in Computer Science*, pages 427–443. Springer-Verlag, 2007.

[MW98] Ueli M. Maurer and Stefan Wolf. Lower bounds on generic algorithms in groups. In Kaisa Nyberg, editor, *Advances in Cryptology - EUROCRYPT '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 72–84, 1998.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.

[RLB+08] Andy Rupp, Gregor Leander, Endre Bangerter, Alexander W. Dent, and Ahmad-Reza Sadeghi. Sufficient conditions for intractability over black-box groups: Generic lower bounds for generalized DL and DH problems. In Josef Pieprzyk, editor, *ASIACRYPT*, volume 5350 of *Lecture Notes in Computer Science*, pages 489–505. Springer, 2008.

[Sch80] Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology - EUROCRYPT 1997*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266, 1997.

[Sho05] Victor Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, 2005.

[Sho06] Victor Shoup. Sequences of games: A tool for taming complexity in security proofs, 2006. URL: `http://eprint.iacr.org/2004/332`.

A Comparing the Generic Ring Model to the Generic Group Model

The generic ring model (GRM) is an extension of the generic *group* model (GGM) (see [Sho97], for instance). Despite many similarities to the GGM, showing the hardness of computational problems

in the GRM seems to be more involved than standard proofs in the GGM. The reason is that a typical proof in the GGM (cf. [Sho97, Mau05, LR06], for instance) introduces a simulation game where group elements are replaced with polynomials that are (implicitly) evaluated with some group elements corresponding to a given problem instance. A key argument in these proofs is that, by construction of the simulator, the degree of these polynomials cannot exceed a certain small bound (often degree one or two). Following Shoup's seminal work [Sho97], a lower bound on the success probability of any generic group algorithm for the given problem is then derived by bounding the number of roots of these polynomials by applying the Schwartz Lemma [Sch80, Sho97]. Usually the bound is useful if the number of roots is sufficiently small. Rupp et al. [RLB+08] have even been able to describe sufficient conditions for the generic hardness of discrete log type problems, that essentially make sure that there is no possibility to compute polynomials with "too large" degree.

In the GGM the number of roots of polynomials is kept small by performing only addition operations on polynomials of degree one in the simulation game (sometimes also a small bounded number of multiplications, for instance when the model is extended to groups with bilinear pairing map, as done in [BB08]). However, in the generic ring model we explicitly allow for multiplication operations, and we do not want to bound the number of allowed multiplication explicitly, in order to keep the model as general as possible. Thus, by repeated squaring an algorithm may compute polynomials of exponential degree. In this case applying the Schwartz Lemma does not yield a useful bound on the number of roots.[6]

It seems that the additional structure provided by a ring, due to the existence of *two* algebraic operations, makes proofs in the generic ring model more challenging. For instance, this is one reason why the optimality of an algorithm for the *black-box field extraction problem* described by Boneh and Lipton [BL96] is still unknown. It is an interesting open question whether there may exist more efficient algorithms, as an efficient algorithm would imply the equivalence of the discrete logarithm problem and the Diffie-Hellman problem. At the same time this would imply the inexistence of field-homomorphic encryption schemes.

B A Simple Example for $\mathcal{U}[\mathcal{C}]$

Let p, q be different primes, $n = pq$, and ϕ be the isomorphism $\mathbb{Z}_p \times \mathbb{Z}_q \to \mathbb{Z}_n$. For $x \in \mathbb{Z}_n$ let $x_p := x \bmod p$ and $x_q := x \bmod q$. Consider the subset $\mathcal{C} \subseteq \mathbb{Z}_n$ such that

$$\mathcal{C} = \{a, b, c\} = \{\phi(a_p, a_q), \phi(b_p, b_q), \phi(c_p, c_q)\}.$$

The uniform closure $\mathcal{U}[\mathcal{C}]$ of $\mathcal{C}$ is the set

$$\mathcal{U}[\mathcal{C}] = \{\phi(d_p, d_q) \mid d_p \in \{a_p, b_p, c_p\}, d_q \in \{a_q, b_q, c_q\}\}.$$

C Proof of Lemma 3

C.1 Helping Lemma

Let us first state a simple lemma which will be useful for the proofs of Lemma 3 and Lemma 4.

Lemma 5. *For $k \in \mathbb{N}$ and $\mu_i \in [0, 1]$ with $i \in \{1, \ldots, k\}$ holds that*

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

[6] There are also some technical obstacles when using the standard technique with polynomials for proofs in the generic ring model, which are one reason why we used the notion of straight line programs instead.

Proof. The inequality holds obviously for $k = 1$. Assuming the inequality holds for k, the step $k \to k+1$ proceeds as follows.

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

C.2 Proving Lemma 3

For notational convenience, let us define $\Gamma(P) := \Pr[P(x') \notin \mathbb{Z}_n^* \text{ and } P(x) \in \mathbb{Z}_n^* \mid x, x' \xleftarrow{U} \mathcal{C}]$ and $\Lambda(P) := \Pr[\gcd(n, P(y)) \notin \{1, n\} \mid y \xleftarrow{U} \mathcal{U}[\mathcal{C}]]$. Thus, in order to prove Lemma 3 we have to show that the inequality

$$\left(\frac{|\mathcal{U}[\mathcal{C}]|}{|\mathcal{C}|}\right)^2 \Lambda(P) \geq \Gamma(P)$$

holds. To this end, we will proceed as follows.

1. We define a helping function $\nu_i(P)$.
2. We express $\Gamma(P)$ and $\Lambda(P)$ in terms of $\nu_i(P)$. More precisely, we will upper bound $\Gamma(P)$ by an expression in $\nu_i(P)$ and lower bound $\Lambda(P)$ by an expression in $\nu_i(P)$.
3. Then we can apply Lemma 5 to shows that resulting inequality holds.

Defining a helping function. Recall that we denote with $n = \prod_{i=1}^{k} p_i^{e_i}$ the prime factor decomposition of n. Let

$$\nu_i(P) := \Pr\left[P(x) \equiv 0 \bmod p_i \mid x \xleftarrow{U} \mathcal{U}[\mathcal{C}]\right]$$

be the probability that $P(x) \equiv 0 \bmod p_i$ for some prime p_i dividing n and $x \xleftarrow{U} \mathcal{U}[\mathcal{C}]$. Recall that $\phi : \mathbb{Z}_{p_1^{e_1}} \times \cdots \times \mathbb{Z}_{p_k^{e_k}} \to \mathbb{Z}_n$ is a ringisomorphism, and P performs only ring operations in $\mathbb{Z}_n$. Therefore P *implicitly* performs all operations on each component $\mathbb{Z}_{p_i^{e_i}}$ separately (and *independently*). Moreover, sampling $x \xleftarrow{U} \mathcal{U}[\mathcal{C}]$ is equivalent to sample $\phi(x_1, \ldots, x_k)$ with x_i chosen *independently* and *uniform* from $\mathcal{C}_i$ for $1 \leq i \leq k$ (cf. Lemma 1). Thus we can express the probability that $P(x) \in \mathbb{Z}_n^*$ for $x \xleftarrow{U} \mathcal{U}[\mathcal{C}]$ as

$$\Pr\left[P(x) \in \mathbb{Z}_n^* \mid x \xleftarrow{U} \mathcal{U}[\mathcal{C}]\right] = \prod_{i=1}^{k} \left(1 - \Pr\left[P(x) \equiv 0 \bmod p_i \mid x \xleftarrow{U} \mathcal{U}[\mathcal{C}]\right]\right) = \prod_{i=1}^{k}(1 - \nu_i(P)).$$

Bounding $\Gamma(P)$ in terms of $\nu_i(P)$. For independently sampled x, x', we have

$$\begin{aligned}\Gamma(P) &= \Pr\left[P(x') \notin \mathbb{Z}_n^* \text{ and } P(x) \in \mathbb{Z}_n^* \mid x, x' \xleftarrow{U} \mathcal{C}\right] \\ &= \Pr\left[P(x) \notin \mathbb{Z}_n^* \mid x \xleftarrow{U} \mathcal{C}\right] \cdot \Pr\left[P(x) \in \mathbb{Z}_n^* \mid x \xleftarrow{U} \mathcal{C}\right]\end{aligned}$$

Note that, since $\mathcal{C} \subseteq \mathcal{U}[\mathcal{C}]$, it holds that

$$\begin{aligned}\Pr\left[P(x) \in \mathbb{Z}_n^* \mid x \xleftarrow{U} \mathcal{C}\right] &\leq \Pr\left[P(y) \in \mathbb{Z}_n^* \mid y \xleftarrow{U} \mathcal{U}[\mathcal{C}]\right] \cdot \Pr\left[y \in \mathcal{C} \mid y \xleftarrow{U} \mathcal{U}[\mathcal{C}]\right]^{-1} \\ &= \Pr\left[P(y) \in \mathbb{Z}_n^* \mid y \xleftarrow{U} \mathcal{U}[\mathcal{C}]\right] \frac{|\mathcal{U}[\mathcal{C}]|}{|\mathcal{C}|}\end{aligned}$$

and

$$\begin{aligned}\Pr\left[P(x) \notin \mathbb{Z}_n^* \mid x \xleftarrow{U} \mathcal{C}\right] &\leq \Pr\left[P(y) \notin \mathbb{Z}_n^* \mid y \xleftarrow{U} \mathcal{U}[\mathcal{C}]\right] \cdot \Pr\left[y \in \mathcal{C} \mid y \xleftarrow{U} \mathcal{U}[\mathcal{C}]\right]^{-1} \\ &= \Pr\left[P(y) \notin \mathbb{Z}_n^* \mid y \xleftarrow{U} \mathcal{U}[\mathcal{C}]\right] \frac{|\mathcal{U}[\mathcal{C}]|}{|\mathcal{C}|} \\ &= \left(1 - \Pr\left[P(y) \in \mathbb{Z}_n^* \mid y \xleftarrow{U} \mathcal{U}[\mathcal{C}]\right]\right) \frac{|\mathcal{U}[\mathcal{C}]|}{|\mathcal{C}|}.\end{aligned}$$

Therefore

$$\begin{aligned}\Gamma(P) &\leq \Pr\left[P(y) \in \mathbb{Z}_n^* \mid y \xleftarrow{U} \mathcal{U}[\mathcal{C}]\right] \left(1 - \Pr\left[P(y) \in \mathbb{Z}_n^* \mid y \xleftarrow{U} \mathcal{U}[\mathcal{C}]\right]\right) \left(\frac{|\mathcal{U}[\mathcal{C}]|}{|\mathcal{C}|}\right)^2 \\ &= \prod_{i=1}^{k}(1-\nu_i(P)) \left(1 - \prod_{i=1}^{k}(1-\nu_i(P))\right) \left(\frac{|\mathcal{U}[\mathcal{C}]|}{|\mathcal{C}|}\right)^2. \qquad (1)\end{aligned}$$

Bounding $\Lambda(P)$ in terms of $\nu_i(P)$. We can find a factor of n by computing $\gcd(n, P(y))$, if $P(y) \equiv 0 \bmod p_i$ for at least one prime p_i dividing n, and $P(y) \not\equiv 0 \bmod n$. Using similar arguments as above, we can therefore express $\Lambda(P)$ in terms of $\nu_i(P)$ as

$$\begin{aligned}\Lambda(P) &= \Pr\left[\gcd(n, P(y)) \notin \{1, n\} \mid y \xleftarrow{U} \mathcal{C}\right] \\ &\geq 1 - \prod_{i=1}^{k} \Pr\left[P(y) \equiv 0 \bmod p_i \mid y \xleftarrow{U} \mathcal{U}[\mathcal{C}]\right] - \prod_{i=1}^{k}\left(1 - \Pr\left[P(y) \equiv 0 \bmod p_i \mid y \xleftarrow{U} \mathcal{U}[\mathcal{C}]\right]\right) \\ &= 1 - \prod_{i=1}^{k}\nu_i(P) - \prod_{i=1}^{k}(1-\nu_i(P)). \qquad (2)\end{aligned}$$

Putting things together. Combining (1) and (2), we see that the inequality

$$\left(\frac{|\mathcal{U}[\mathcal{C}]|}{|\mathcal{C}|}\right)^2 \Lambda(P) \geq \Gamma(P)$$

holds if inequality

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

holds. The latter is equivalent to

$$\left(1 - \prod_{i=1}^{k}(1-\nu_i(P))\right)^2 \geq \prod_{i=1}^{k}\nu_i(P).$$

Applying Lemma 5, we may conclude that this inequality holds for $k \geq 2$.

D Proof of Lemma 4

In the following, for straight line programs P, Q let $\Gamma'(P,Q) := \Pr[P(x) \equiv_n Q(x) \text{ and } P(x') \not\equiv_n Q(x') \mid x, x' \xleftarrow{U} \mathcal{C}]$ and $\Lambda'(P,Q) := \Pr[\gcd(n, P(y) - Q(y)) \notin \{1, n\} \mid y \xleftarrow{U} \mathcal{U}[\mathcal{C}]]$. Then, in order to prove our claim, we have to show that

$$\left(\frac{|\mathcal{U}[\mathcal{C}]|}{|\mathcal{C}|}\right)^2 \Lambda'(P,Q) \geq \Gamma'(P,Q).$$

The proof will proceed very similar to the proof of Lemma 3, except that we have to define a slightly different helping function.

Defining a helping function. For $n = \prod_{i=1}^{k} p_i^{e_i}$, let

$$\nu_i'(P,Q) := \Pr\left[P(y) - Q(y) \equiv 0 \bmod p_i^{e_i} \mid y \xleftarrow{U} \mathcal{U}[\mathcal{C}]\right].$$

Thus, for two straight line programs P, Q, the function ν_i' determines the probability that $P(y) - Q(y) \equiv 0 \bmod p_i^{e_i}$ for uniform sampled $y \xleftarrow{U} \mathcal{U}[\mathcal{C}]$. Using similar arguments as above, we can express the probability that $P(y) \equiv Q(y) \bmod n$ for $y \xleftarrow{U} \mathcal{U}[\mathcal{C}]$ as

$$\begin{aligned}
\Pr\left[P(y) \equiv Q(y) \bmod n \mid y \xleftarrow{U} \mathcal{U}[\mathcal{C}]\right] &= \Pr\left[P(y) - Q(y) \equiv 0 \bmod n \mid y \xleftarrow{U} \mathcal{U}[\mathcal{C}]\right] \\
&= \prod_{i=1}^{k} \Pr\left[P(y) - Q(y) \equiv 0 \bmod p_i^{e_i} \mid y \xleftarrow{U} \mathcal{U}[\mathcal{C}]\right] \\
&= \prod_{i=1}^{k} \nu_i'(P,Q).
\end{aligned}$$

From here the proof proceeds just like the proof of Lemma 3. We express $\Lambda'(P,Q)$ and $\Gamma'(P,Q)$ in terms of $\nu_i'(P,Q)$, and apply Lemma 5 to prove the resulting inequality.

Bounding $\Gamma'(P,Q)$ in terms of $\nu_i'(P,Q)$. Since x, x' are sampled independently, we have

$$\begin{aligned}
\Gamma'(P,Q) :=& \Pr\left[(P(x) \equiv_n Q(x) \text{ and } P(x') \not\equiv_n Q(x') \mid x, x' \xleftarrow{U} \mathcal{C}\right] \\
=& \Pr\left[(P(x) \equiv_n Q(x) \mid x \xleftarrow{U} \mathcal{C}\right] \cdot \Pr\left[P(x) \not\equiv_n Q(x) \mid x \xleftarrow{U} \mathcal{C}\right].
\end{aligned}$$

Again, in order to be able to sample from $\mathcal{U}[\mathcal{C}]$ instead of $\mathcal{C}$, we use that $\mathcal{C} \subseteq \mathcal{U}[\mathcal{C}]$ to bound

$$\Pr\left[P(x) \equiv_n Q(x) \mid x \xleftarrow{U} \mathcal{C}\right] \leq \Pr\left[P(y) \equiv_n Q(y) \mid y \xleftarrow{U} \mathcal{U}[\mathcal{C}]\right] \frac{|\mathcal{U}[\mathcal{C}]|}{|\mathcal{C}|}$$

and

$$\Pr\left[P(x) \not\equiv_n Q(x) \mid x \xleftarrow{U} \mathcal{C}\right] \leq \Pr\left[P(y) \not\equiv_n Q(y) \mid y \xleftarrow{U} \mathcal{U}[\mathcal{C}]\right] \frac{|\mathcal{U}[\mathcal{C}]|}{|\mathcal{C}|}.$$

Therefore

$$\begin{aligned}
\Gamma'(P,Q) &= \Pr\left[(P(x) \equiv_n Q(x) \mid x \xleftarrow{U} \mathcal{C}\right] \cdot \Pr\left[P(x) \not\equiv_n Q(x) \mid x \xleftarrow{U} \mathcal{C}\right] \\
&\leq \Pr\left[P(y) \equiv_n Q(y) \mid y \xleftarrow{U} \mathcal{U}\left[\mathcal{C}\right]\right] \cdot \Pr\left[P(y) \not\equiv_n Q(y) \mid y \xleftarrow{U} \mathcal{U}\left[\mathcal{C}\right]\right] \left(\frac{|\mathcal{U}\left[\mathcal{C}\right]|}{|\mathcal{C}|}\right)^2 \\
&= \Pr\left[P(y) \equiv_n Q(y) \mid x \xleftarrow{U} \mathcal{U}\left[\mathcal{C}\right]\right] \left(1 - \Pr\left[P(y) \equiv_n Q(y) \mid y \xleftarrow{U} \mathcal{U}\left[\mathcal{C}\right]\right]\right) \left(\frac{|\mathcal{U}\left[\mathcal{C}\right]|}{|\mathcal{C}|}\right)^2 \\
&= \prod_{i=1}^{k} \nu_i'(P,Q) \left(1 - \prod_{i=1}^{k} \nu_i'(P,Q)\right) \left(\frac{|\mathcal{U}\left[\mathcal{C}\right]|}{|\mathcal{C}|}\right)^2. \qquad (3)
\end{aligned}$$

Bounding $\Lambda'(P,Q)$ in terms of $\nu_i'(P,Q)$. As above, we can find a factor of n by computing $\gcd(n, P(y))$, if $P(y) \equiv 0 \bmod p_i^{e_i}$ for at least one prime power $p_i^{e_i}$ dividing n, and $P(y) \not\equiv 0 \bmod n$. Thus we can express $\Lambda'(P,Q)$ in terms of $\nu_i'(P)$ as

$$\Lambda'(P,Q) = \Pr\left[\gcd(n, P(y)) \notin \{1, n\} \mid y \xleftarrow{U} \mathcal{U}\left[\mathcal{C}\right]\right] \geq 1 - \prod_{i=1}^{k} \nu_i'(P,Q) - \prod_{i=1}^{k} (1 - \nu_i'(P,Q)). \qquad (4)$$

Putting things together. Combining (3) and (4), we see that

$$\left(\frac{|\mathcal{U}\left[\mathcal{C}\right]|}{|\mathcal{C}|}\right)^2 \Lambda'(P,Q) \geq \Gamma'(P,Q)$$

holds if

$$1 - \prod_{i=1}^{k} \nu_i'(P,Q) - \prod_{i=1}^{k} (1 - \nu_i'(P,Q)) \geq \left(\prod_{i=1}^{k} \nu_i'(P,Q)\right) \left(1 - \prod_{i=1}^{k} \nu_i'(P,Q)\right)$$

holds. The latter inequality is equivalent to

$$\left(1 - \prod_{i=1}^{k} \nu_i'(P,Q)\right)^2 \geq \prod_{i=1}^{k} (1 - \nu_i'(P,Q)).$$

By Lemma 5 the claim follows now for $k \geq 2$ by letting $\mu_i := 1 - \nu_i'(P,Q)$.

E The Intuition behind Lemma 3 and 4

Simplifying a little, Lemma 3 and 4 state *essentially*[7] that: if we are given a straight line program mapping "many" inputs to zero and "many" inputs to a non-zero value, then we can find a factor of n by sampling $y \xleftarrow{U} \mathcal{U}\left[\mathcal{C}\right]$ and computing $\gcd(n, P(y))$. At a first glance this seems counterintuitive.

For instance, consider the case $\mathcal{C} = \mathbb{Z}_n$, then we have $\mathcal{U}\left[\mathcal{C}\right] = \mathbb{Z}_n$. Assume a straight line program P mapping half of the elements of $\mathbb{Z}_n$ to 0, and the other half to 1. Then P maps "many" inputs to zero and "many" inputs to a non-zero value, but clearly computing $\gcd(n, P(y))$ for any $y \xleftarrow{U} \mathbb{Z}_n$ yields only trivial factors of n, hence this seems to be a counterexample to Lemma 3 and 4. However,

[7] In case of Lemma 3 note that $P(x) \in \mathbb{Z}_n^*$ and $P(x') \notin \mathbb{Z}_n^*$ means that $P(x')$ is zero modulo *at least one* prime factor of n, while $P(x) \not\equiv 0$ modulo *all* prime factors of n. In case of Lemma 4 observe that if we have $P(x) - Q(x) \equiv 0 \bmod n$ and $P(x') - Q(x') \not\equiv 0 \bmod n$, then x is mapped to zero and x' is not mapped to zero by the straight line program $S(x) := P(x) - Q(x)$.

in fact this is *not* a counterexample, since there exists no straight line program P satisfying the assumed property, if n is the product of at least two different primes.

The reason for this is a consequence of the Chinese Remainder Theorem. Let $n = pq$ with $\gcd(p, q) = 1$ (p and q not necessarily prime). By the Chinese Remainder Theorem, the ring $\mathbb{Z}_n$ is isomorphic to $\mathbb{Z}_p \times \mathbb{Z}_q$. Let $\phi : \mathbb{Z}_p \times \mathbb{Z}_q \to \mathbb{Z}_n$ denote this isomorphism. Assume $x, x' \in \mathbb{Z}_n$ and a straight line program P such that $P(x) \equiv 0 \bmod n$ and $P(x') \equiv 1 \bmod n$. Since ϕ is a ringisomorphism and P performs only ring operations, it holds that

$$P(x) = \phi(P(x) \bmod p, P(x) \bmod q) = \phi(0, 0)$$

and

$$P(x') = \phi(P(x') \bmod p, P(x') \bmod q) = \phi(1, 1).$$

The crucial observation is now that for each pair $(x, x') \in \mathbb{Z}_n^2$, there exist $c, d \in \mathbb{Z}_n$ such that $c = \phi(x' \bmod p, x \bmod q)$ and $d = \phi(x \bmod p, x' \bmod q)$. Evaluating P with c or d yields

$$P(c) = \phi(P(x') \bmod p, P(x) \bmod q) = \phi(1, 0)$$

or

$$P(d) = \phi(P(x) \bmod p, P(x') \bmod q) = \phi(0, 1).$$

We therefore have $\gcd(n, P(c)) = q$ and $\gcd(n, P(d)) = p$. Thus, if P has the property that $P(x) = \phi(0, 0)$ and $P(x') = \phi(1, 1)$ with "high" probability for $x, x' \xleftarrow{U} \mathbb{Z}_n$, then we can also sample $y \xleftarrow{U} \mathbb{Z}_n$ such that $P(y) = \phi(0, 1)$ or $P(y) = \phi(1, 0)$ with "high" probability. A factor of n can therefore be found by sampling y and computing $\gcd(n, P(y))$.

Generalizing the notion described above, putting it into a more precise and formal language, and handling some technical obstacles,[8] we obtain the proofs given in Appendices C and D. Instead of our approach based on probabilities, it is also possible to prove Lemma 3 and 4 using combinatoric means, but this seems to require a number of case distinctions. We believe that the proofs given in this paper are easier to verify.

F Proof of Proposition 1

By construction of the simulator, $\Pr[\mathcal{F}_{\mathsf{test}}]$ is bound by the probability that there exists $P_j \sqsubseteq P$ such that either

1. testsim() has returned false where test() would have returned true, i.e. it holds that
   (a) $(P_j(x) \in \mathbb{Z}_n^*$ and $P_j(x_r) \notin \mathbb{Z}_n^*)$ or
   (b) $(P_j(x) = \perp$ and $P_j(x_r) \notin \mathbb{Z}_n^*)$
2. or testsim() has returned true where test() would have returned false, i.e. it holds that
   (a) $(P_j(x_r) \in \mathbb{Z}_n^*$ and $P_j(x) \notin \mathbb{Z}_n^*)$ or
   (b) $(P_j(x_r) = \perp$ and $P_j(x) \notin \mathbb{Z}_n^*)$

for $x \xleftarrow{U} \mathcal{C}$ and one of the values $x_1, \ldots, x_m \xleftarrow{U} \mathcal{C}$ sampled by the simulator.

Note that if there exists P_j such that $(P_j(x) = \perp$ and $P_j(x_r) \neq \perp)$, then this implies that there exists $P_k \sqsubseteq P$ with $k < j$ such that $(P_j(x_r) \notin \mathbb{Z}_n^*$ and $P_j(x) \in \mathbb{Z}_n^*)$ by Lemma 2. Hence, in order

[8] E.g. the fact that simulator and factoring algorithm sample from *subsets* of $\mathbb{Z}_n$ (what made it necessary to define the *uniform closure* of subsets of $\mathbb{Z}_n$).

to bound the probability of $\mathcal{F}_{\mathsf{test}}$, it suffices to consider the probability that there exists a straight line program $P_j \sqsubseteq P$ such that

$$(P_j(x_r) \notin \mathbb{Z}_n^* \text{ and } P_j(x) \in \mathbb{Z}_n^*) \text{ or } (P_j(x) \notin \mathbb{Z}_n^* \text{ and } P_j(x_r) \in \mathbb{Z}_n^*) \tag{5}$$

for $x, x_1, \ldots, x_m \xleftarrow{U} \mathcal{C}$.

For *fixed* P_j we can bound this probability as follows.

$$\begin{aligned}
&\Pr\left[(P_j(x_r) \notin \mathbb{Z}_n^* \text{ and } P_j(x) \in \mathbb{Z}_n^*) \text{ or } (P_j(x) \notin \mathbb{Z}_n^* \text{ and } P_j(x_r) \in \mathbb{Z}_n^*) \mid x, x_1, \ldots, x_m \xleftarrow{U} \mathcal{C}\right] \\
\leq & m \Pr\left[(P_j(x') \notin \mathbb{Z}_n^* \text{ and } P_j(x) \in \mathbb{Z}_n^*) \text{ or } (P_j(x) \notin \mathbb{Z}_n^* \text{ and } P_j(x') \in \mathbb{Z}_n^*) \mid x, x' \xleftarrow{U} \mathcal{C}\right] \\
= & m \left(\Pr\left[P_j(x') \notin \mathbb{Z}_n^* \text{ and } P_j(x) \in \mathbb{Z}_n^* \mid x, x' \xleftarrow{U} \mathcal{C}\right] + \Pr\left[P_j(x) \notin \mathbb{Z}_n^* \text{ and } P_j(x') \in \mathbb{Z}_n^* \mid x, x' \xleftarrow{U} \mathcal{C}\right]\right) \\
= & 2m \Pr\left[P_j(x) \notin \mathbb{Z}_n^* \text{ and } P_j(x') \in \mathbb{Z}_n^* \mid x, x' \xleftarrow{U} \mathcal{C}\right].
\end{aligned}$$

Using this, we obtain the following bound on the probability that there exists *any* $P_j \sqsubseteq P$ satisfying (5).

$$\begin{aligned}
\Pr[\mathcal{F}_{\mathsf{test}}] &\leq 2m \sum_{j=0}^{m} \Pr\left[P_j(x) \notin \mathbb{Z}_n^* \text{ and } P_j(x') \in \mathbb{Z}_n^* \mid x, x' \xleftarrow{U} \mathcal{C}\right] \\
&\leq 2m(m+1) \max_{0 \leq j \leq m} \left\{\Pr\left[P_j(x) \notin \mathbb{Z}_n^* \text{ and } P_j(x') \in \mathbb{Z}_n^* \mid x, x' \xleftarrow{U} \mathcal{C}\right]\right\}
\end{aligned}$$

G Proof of Proposition 2

By construction of the simulator, $\Pr[\mathcal{F}_{\mathsf{equal}}]$ is bound by the probability that there exist $P_i, P_j \sqsubseteq P$ and $x_r \in \{x_1, \ldots, x_m\}$ such that either

1. $\mathsf{equalsim}()$ has returned false where $\mathsf{equal}()$ would have returned true, i.e. it holds that
   (a) $(P_i(x) \equiv_n P_j(x)$ and $P_i(x_r) \not\equiv_n P_j(x_r))$ or
   (b) $(P_i(x) \equiv_n P_j(x)$ and $P_i(x_r) = \perp)$ or
   (c) $(P_i(x) \equiv_n P_j(x)$ and $P_j(x_r) = \perp)$
2. or $\mathsf{equalsim}()$ has returned true where $\mathsf{equal}()$ would have returned false, i.e. it holds that
   (a) $(P_i(x_r) \equiv_n P_j(x)$ and $P_i(x) \not\equiv_n P_j(x))$ or
   (b) $(P_i(x_r) \equiv_n P_j(x)$ and $P_i(x) = \perp)$ or
   (c) $(P_i(x_r) \equiv_n P_j(x)$ and $P_j(x) = \perp)$

for $x \xleftarrow{U} \mathcal{C}$ and one of the values $x_1, \ldots, x_m \xleftarrow{U} \mathcal{C}$ sampled by the simulator.

Applying Lemma 2 to cases 1.b), 1.c), 2.b), and 2.c), we see that in order to bound the probability of $\mathcal{F}_{\mathsf{equal}}$, it suffices to consider the probability that there exist $P_i, P_j \sqsubseteq P$ or $P_k \sqsubseteq P$ such that

1. $\mathsf{equalsim}()$ has returned false where $\mathsf{equal}()$ would have returned true, i.e. it holds that
   (a) $(P_i(x) \equiv_n P_j(x)$ and $P_i(x_r) \not\equiv_n P_j(x_r))$ or
   (b) $(P_k(x) \in \mathbb{Z}_n^*$ and $P_k(x_r) \notin \mathbb{Z}_n^*)$
2. or $\mathsf{equalsim}()$ has returned true where $\mathsf{equal}()$ would have returned false, i.e. it holds that
   (a) $(P_i(x_r) \equiv_n P_j(x_r)$ and $P_i(x) \not\equiv_n P_j(x))$ or
   (b) $(P_k(x_r) \in \mathbb{Z}_n^*$ and $P_k(x) \notin \mathbb{Z}_n^*)$

for $x \xleftarrow{U} \mathcal{C}$ and one of the values $x_1, \ldots, x_m \xleftarrow{U} \mathcal{C}$ sampled by the simulator.

Let us first consider the cases 1.a) and 2.a). For *fixed* P_i, P_j we can bound the probability of 1.a) and 2.a) as follows.

$$\begin{aligned}
&\Pr[(P_i(x) \equiv_n P_j(x) \text{ and } P_i(x_r) \not\equiv_n P_j(x_r)) \\
&\quad \text{or } (P_i(x_r) \equiv_n P_j(x_r) \text{ and } P_i(x) \not\equiv_n P_j(x)) \mid x, x_1, \ldots, x_m \xleftarrow{U} \mathcal{C}] \\
\leq & m \Pr[(P_i(x) \equiv_n P_j(x) \text{ and } P_i(x')) \not\equiv_n P_j(x') \\
&\quad \text{or } ((P_i(x') \equiv_n P_j(x') \text{ and } P_i(x) \not\equiv_n P_j(x)) \mid x, x' \xleftarrow{U} \mathcal{C}] \\
= & m \Pr\left[P_i(x) \equiv_n P_j(x) \text{ and } P_i(x') \not\equiv_n P_j(x') \mid x, x' \xleftarrow{U} \mathcal{C}\right] \\
+ & m \Pr\left[P_i(x') \equiv_n P_j(x') \text{ and } P_i(x) \not\equiv_n P_j(x) \mid x, x' \xleftarrow{U} \mathcal{C}\right] \\
= & 2m \Pr\left[P_i(x) \equiv_n P_j(x) \text{ and } P_i(x') \not\equiv_n P_j(x') \mid x, x' \xleftarrow{U} \mathcal{C}\right]
\end{aligned}$$

Using the last term, the probability that there exists *any* pair $P_i, P_j \in P$ such that 1.a) or 2.a) holds is at most

$$\begin{aligned}
& 2m \sum_{-1 \leq i < j \leq m} \Pr\left[P_i(x) \equiv_n P_j(x) \text{ and } P_i(x') \not\equiv_n P_j(x') \mid x, x' \xleftarrow{U} \mathcal{C}\right] \\
\leq & 2m(m+2)(m+1) \max_{-1 \leq i < j \leq m} \left\{\Pr\left[P_i(x) \equiv_n P_j(x) \text{ and } P_i(x') \not\equiv_n P_j(x') \mid x, x' \xleftarrow{U} \mathcal{C}\right]\right\} \\
= & 2m(m^2+3m+1) \max_{-1 \leq i < j \leq m} \left\{\Pr\left[P_i(x) \equiv_n P_j(x) \text{ and } P_i(x') \not\equiv_n P_j(x') \mid x, x' \xleftarrow{U} \mathcal{C}\right]\right\}
\end{aligned}$$

Now let us consider the cases 1.b) and 2.b). Comparing these cases to Expression (5), the proof of Proposition 1 shows that the probability that there exists *any* $P_k \in P$ such that 1.b) or 2.b) holds is at most

$$2m(m+1) \max_{0 \leq k \leq m} \left\{\Pr\left[P_k(x) \notin \mathbb{Z}_n^* \text{ and } P_k(x') \in \mathbb{Z}_n^* \mid x, x' \xleftarrow{U} \mathcal{C}\right]\right\}.$$

Since $\mathcal{F}_{\mathsf{equal}}$ implies that there exists either a pair $P_i, P_j \in P$ such that 1.a) or 2.a) holds, or $P_k \in P$ such that 1.b) or 2.b) holds, we may conclude

$$\begin{aligned}
\Pr[\mathcal{F}_{\mathsf{equal}}] \leq & 2m(m^2+3m+1) \max_{-1 \leq i < j \leq m} \left\{\Pr\left[P_i(x) \equiv_n P_j(x) \text{ and } P_i(x') \not\equiv_n P_j(x') \mid x, x' \xleftarrow{U} \mathcal{C}\right]\right\} \\
& + 2m(m+1) \max_{0 \leq k \leq m} \left\{\Pr\left[P_k(x) \notin \mathbb{Z}_n^* \text{ and } P_k(x') \in \mathbb{Z}_n^* \mid x, x' \xleftarrow{U} \mathcal{C}\right]\right\}
\end{aligned}$$