On the Analysis of Cryptographic Assumptions in the Generic Ring Model*

Tibor Jager and Jörg Schwenk

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

Abstract. At *Eurocrypt 2009* Aggarwal and Maurer proved that breaking RSA is equivalent to factoring in the *generic ring model*. This model captures algorithms that may exploit the full algebraic structure of the ring of integers modulo n, but no properties of the given representation of ring elements. This interesting result raises the question how to interpret proofs in the generic ring model. For instance, one may be tempted to deduce that a proof in the generic model gives some evidence that solving the considered problem is also hard in a general model of computation. But is this reasonable?
We prove that computing the *Jacobi symbol* is equivalent to factoring in the generic ring model. Since there are simple and efficient non-generic algorithms computing the Jacobi symbol, we show that the generic model cannot give any evidence towards the hardness of a computational problem. Despite this negative result, we also argue why proofs in the generic ring model are still interesting, and show that solving the *quadratic residuosity* and *subgroup decision* problems is generically equivalent to factoring.

1 Introduction

The security of asymmetric cryptographic systems relies on assumptions that certain computational problems, mostly from number theory and algebra, are intractable. Since proving useful lower complexity bounds in a general model of computation seems to be impossible with currently available techniques, these assumptions have been analyzed in restricted models, see [22, 17, 8, 1], for instance. A natural and very 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*.

Considering fundamental cryptographic problems in the generic model is motivated by the following ideas. First, showing that a cryptographic assumption

* This is an extended abstract, the full version is available on eprint [13]. Supported by the European Community (FP7/2007-2013), grant ICT-2007-216646 - European Network of Excellence in Cryptology II (ECRYPT II).

holds with respect to a restricted but meaningful class of algorithms might indicate that the idea of basing the security of cryptosystems on this assumption is not totally flawed, and may therefore be seen as evidence that the assumption is also valid in a general model of computation. Second, showing that a large class of algorithms is not able to solve a computational problem efficiently is an important insight for the search for cryptanalytic algorithms, and can be used to deduce the optimality of certain classes of algorithms. Moreover, the generic model is a valuable tool to study the relationship among computational problems, such as the equivalence of the discrete logarithm and the Diffie-Hellman problem, as done in [6, 18, 19, 16, 2], for instance.

In this paper we prove a general theorem which states that solving certain subset membership problems in the ring $\mathbb{Z}_n$ is equivalent to factoring n. This main theorem allows us to provide an example for a computational problem with high cryptographic relevance which is easy to solve in general, but equivalent to factoring in the generic model. Concretely, we show that computing the *Jacobi symbol* is equivalent to factoring in the generic ring model.

For many common idealized models in cryptography it has been shown that a cryptographic reduction in the ideal model need not guarantee security in the "real world". Well-known examples are, for instance, the random oracle model [9], the ideal cipher model [3], and the generic *group* model [12, 11]. All these results have in common that they used somewhat contrived constructions that deviate from standard cryptographic practice.[1] In contrast, our result on the generic equivalence of computing the Jacobi symbol and factoring is an example for a truly practical computational problem that is provably hard in the generic model, but easy to solve in general. This is an important aspect for interpreting results in the generic ring model, like [7, 8, 15, 2, 1]. Thus a proof in the generic model is unfortunately not even an indicator that the considered problem is indeed useful for cryptographic applications.

This negative result does not affect the other mentioned motivations for the analysis of computational problems in the generic ring model. A lower bound in this model allows to deduce the optimality of certain classes of algorithms, and gives insight into the relationship between cryptographic problems, which is also of interest. Motivated by this fact, we also show that solving the *quadratic residuosity* and *subgroup decision* problems is generically equivalent to factoring. For the latter problem we show that the equivalence holds even in presence of a Diffie-Hellman oracle. Thus, a Diffie-Hellman oracle does not help in solving the subgroup decision problem.

By taking a closer look at the construction of the simulator used in the proof of our main theorem, we furthermore deduce that for a certain class of computational problems there exists an efficient generic ring algorithm if and only if there is an efficient straight line program solving the problem.

[1] An exception is the result of [20], showing a (non-generic) attack on a scheme with provable security in the generic model. However, [14] note that this stems not from a weakness in the generic model, but from an incorrect security proof.

1.1 Related Work

Previous work considering fundamental cryptographic assumptions in the generic model considered primarily discrete logarithm-based problems and the RSA problem. Starting with Shoup's seminal paper [22], it was proven that solving the discrete logarithm problem, the Diffie-Hellman problem, and related problems [18, 17, 21] is hard with respect to generic *group* algorithms. Damgård and Koprowski showed the generic intractability of root extraction in groups of hidden order [10].

Brown [8] 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 [15] 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 [1] extended this result from low-exponent RSA to full RSA and to generic ring algorithms that may also compute multiplicative inverses. Boneh and Venkatesan [7] 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 [6, 16, 2].

2 Preliminaries

2.1 Notation

For a set A and a probability distribution $\mathcal{D}$ on A, we denote with $a \overset{\mathcal{D}}{\leftarrow} A$ 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 $a_1, \ldots, a_k \overset{\mathcal{D}}{\leftarrow} A$, 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 $i \neq j$.

Let $P = (S_1, \ldots, S_m)$ be a finite 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 a sequences P with we write $P_k \sqsubseteq P$ to denote that P_k is a subsequence of P such that P_k consists of the *first* $|P_k|$ elements of P.

2.2 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 the 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 1 (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*

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

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* $y \overset{U}{\leftarrow} \mathcal{U}[\mathcal{C}]$ *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* $y = \phi(y_1, \ldots, y_k)$*.*

2.3 Straight Line Programs

A straight line program over a ring R is a generic ring algorithm performing a fixed sequence of ring operations, without branching, that outputs an element of R. Thus straight line programs are a subclass of generic ring algorithms. The following definition is a simple extension of [8, Definition 1] to straight line programs that may also compute multiplicative inverses.

Definition 2 (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 $P_{-1}(x) = 1$ and $P_0(x) = x$ for all $x \in \mathbb{Z}_n$.

2.4 Generic Ring Algorithms

Similar to straight line programs, generic ring algorithms perform a sequence of ring operations on the input values $1, x \in \mathbb{Z}_n$. 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 a tuple $(j, \circ) \in \{-1, \ldots, |P|\} \times \{+, -, \cdot, /\}$ as input. The procedure returns false if $\circ = /$ and $P_j(x) \notin \mathbb{Z}_n^*$, and true otherwise.
- The $\mathsf{equal}()$-procedure takes a tuple $(i, j) \in \{-1, \ldots, |P|\} \times \{-1, \ldots, |P|\}$ as input. 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.

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 $y \overset{U}{\leftarrow} \mathcal{U}[\mathcal{C}]$ 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 $x, x' \overset{U}{\leftarrow} \mathcal{C}$.

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

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

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

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

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

The proofs of Lemma 3 and 4 are based on the Chinese Remainder Theorem. Full proofs are given in Appendix C and D of the full version [13]. We also discuss the intuition behind these lemmas in Appendix E of [13].

3 Subset Membership Problems in Generic Rings

Definition 3 (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* $x \overset{U}{\leftarrow} \mathcal{C}$*, decide whether* $x \in \mathcal{V}$*.*

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

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 2.4, except that $\mathcal{O}_{\text{smp}}$ receives a uniformly random element $x \overset{U}{\leftarrow} \mathcal{C}$ 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 $\Pr[\mathcal{S}_{\text{sim}}] + \Pr[\mathcal{F}]$. 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 $x \overset{U}{\leftarrow} \mathcal{C}$ 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 $x_r \overset{U}{\leftarrow} \mathcal{C}$ 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 $x_r \overset{U}{\leftarrow} \mathcal{C}$ 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, $r \in \{1, \dots, m\}$. 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 $\mathsf{testsim}()$-procedure fails to simulate $\mathsf{test}()$ perfectly. This means that $\mathsf{testsim}()$ returns false on a procedure call where $\mathsf{test}()$ would have returned true, or $\mathsf{testsim}()$ returns true where $\mathsf{test}()$ would have returned false. Let $\mathcal{F}_{\mathsf{test}}$ denote the event that this happens on at least one call of $\mathsf{testsim}()$.
2. The $\mathsf{equalsim}()$-procedure fails to simulate $\mathsf{equal}()$ perfectly. This means that $\mathsf{equalsim}()$ has returned true where $\mathsf{equal}()$ would have returned false, or $\mathsf{equalsim}()$ has returned false where $\mathsf{equal}()$ would have returned true. Let $\mathcal{F}_{\mathsf{equal}}$ denote the event that this happens at at least one call of $\mathsf{equalsim}()$.

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 $\mathsf{testsim}()$-procedure fails to simulate $\mathsf{test}()$ only if either $\mathsf{testsim}()$ has returned false where $\mathsf{test}()$ would have returned true, or $\mathsf{testsim}()$ has returned true where $\mathsf{test}()$ 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 $(P_j(x_r) = \perp$ 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 $(P_j(x) = \perp$ and $P_j(x_r) \notin \mathbb{Z}_n^*)$. Hence, $\mathsf{testsim}()$-procedure fails to simulate $\mathsf{test}()$ 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.

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

We sketch the proof of Proposition 1 in Appendix B. A full proof is given in Appendix F of the full version.

Bounding the Probability of $\mathcal{F}_{\mathsf{equal}}$ The $\mathsf{equalsim}()$-procedure fails to simulate $\mathsf{equal}()$ only if either $\mathsf{equalsim}()$ has returned false where $\mathsf{equal}()$ would have returned true, or $\mathsf{equalsim}()$ has returned true where $\mathsf{equal}()$ 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

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

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

[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.

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

$$\begin{aligned}&(P_i(x) \equiv_n P_j(x) \text{ and } P_i(x_r) \not\equiv_n P_j(x_r))\\ \text{or } &(P_i(x_r) \equiv_n P_j(x_r) \text{ and } P_i(x) \not\equiv_n P_j(x))\\ \text{or } &(P_k(x_r) \notin \mathbb{Z}_n^* \text{ and } P_k(x) \in \mathbb{Z}_n^*)\\ \text{or } &(P_k(x) \notin \mathbb{Z}_n^* \text{ and } P_k(x_r) \in \mathbb{Z}_n^*).\end{aligned}$$

Proposition 2.

$$\Pr[\mathcal{F}_{\mathsf{equal}}] \leq 2m(m^2 + 3m + 1)\Phi + 2m(m + 1)\Psi,$$

where

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

The proof of Proposition 2, which is based on the same ideas as the proof of Proposition 1, is given in Appendix G of the full version.

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

$$\begin{aligned}\Pr[\mathcal{F}] &\leq \Pr[\mathcal{F}_{\mathsf{test}}] + \Pr[\mathcal{F}_{\mathsf{equal}}]\\ &\leq 2m(m^2 + 3m + 1)\Phi + 4m(m + 1)\Psi.\end{aligned}$$

where Φ and Ψ are defined as above.

3.3 Bounding the Success Probability

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

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

which implies

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

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 $y \overset{U}{\leftarrow} \mathcal{U}[\mathcal{C}]$ 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 $y \overset{U}{\leftarrow} \mathcal{U}[\mathcal{C}]$ and computes $\gcd(P_i(y) - P_j(y), n)$ for $-1 \leq i < j \leq r$.

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 $y \overset{U}{\leftarrow} \mathcal{U}[\mathcal{C}]$ and the straight line program $P_k \sqsubseteq P$ satisfying

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

Let $\gamma_1 := \max_{0 \leq k \leq m}\{\Pr[P_k(x) \notin \mathbb{Z}_n^* \text{ and } P_k(x') \in \mathbb{Z}_n^* \mid x, x' \overset{U}{\leftarrow} \mathcal{C}]\}$, 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 $y \overset{U}{\leftarrow} \mathcal{U}[\mathcal{C}]$. So in particular $\mathcal{B}$ evaluates $\gcd(P_i(y) - P_j(y), n)$ with $y \overset{U}{\leftarrow} \mathcal{U}[\mathcal{C}]$ for the pair of straight line programs $P_i, P_j \sqsubseteq P$ satisfying

$$\Pr\left[P_i(x) \equiv_n P_j(x) \text{ and } P_i(x') \not\equiv_n P_j(x') \mid x, x' \overset{U}{\leftarrow} \mathcal{C}\right]$$
$$= \max_{-1 \leq i < j \leq m} \left\{\Pr\left[P_i(x) \equiv_n P_j(x) \text{ and } P_i(x') \not\equiv_n P_j(x') \mid x, x' \overset{U}{\leftarrow} \mathcal{C}\right]\right\}.$$

Let $\gamma_2 := \max_{-1 \leq i < j \leq m}\{\Pr[P_i(x) \equiv_n P_j(x) \text{ and } P_i(x') \not\equiv_n P_j(x') \mid x, x' \overset{U}{\leftarrow} \mathcal{C}]\}$, 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 $\mathsf{Adv}_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{\mathrm{smp}}}(n)) \leq \Pr[\mathcal{F}]$, and the derived bound on $\Pr[\mathcal{F}]$, we can obtain a lower bound on γ by

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

which implies the inequality

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

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

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

4 Computing the Jacobi Symbol with Generic Ring Algorithms

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* [23, 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.

There exist simple efficient algorithms computing the Jacobi symbol in $\mathbb{Z}_n$ without factoring n. These algorithms are not generic, cf. [23, p.288].

Theorem 2. *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. The theorem follows by applying Theorem 1 and the fact that $\mathcal{U}\left[\mathbb{Z}_n^*\right] = \mathbb{Z}_n^*$, since

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

5 The Generic Quadratic Residuosity Problem and Factoring

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

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

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

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 [23, 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

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

6 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 $\mathcal{V} = \mathbb{G}_p$.*

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* $\mathbb{Z}_n$.

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 $\mathcal{O}_{\mathrm{sdp}}$ which is defined exactly like the generic ring oracle described in Section 2.4, except that it does not provide the operation $/$. $\mathcal{O}_{\mathrm{sdp}}$ receives an element $x \in \mathbb{Z}_n$ as input, where x is constructed as follows: sample $(x_p, x_q) \overset{U}{\leftarrow} \mathbb{Z}_p \times \mathbb{Z}_q$ and bit $b \overset{U}{\leftarrow} \{0, 1\}$ 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 $\mathcal{A}^{\mathcal{O}_{\mathrm{sdp}}}(n) = 1$, or $x \notin \mathbb{G}_p$ and $\mathcal{A}^{\mathcal{O}_{\mathrm{sdp}}}(n) = 0$. We define the *advantage* of an algorithm $\mathcal{A}$ solving the subgroup decision problem with probability $\Pr[\mathcal{S}]$ as

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

Remark 1. If we would also allow to query the oracle for divisions (which correspond 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 [5] 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 when evaluating the bilinear pairing map,[5] as done in [4]. 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 [5] as a special case.

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

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

6.1 Generic Equivalence to Factoring

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 $\mathbb{Z}_p$-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

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

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.

7 Analyzing Search 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 *search* problems where the algorithm outputs a ring element or integer. Let us sketch two possibilities. The first one is to formulate a suitable subset membership problem which reduces to the considered search 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.

Acknowledgements. We would like to thank Andy Rupp and Sven Schäge for helpful discussions, and Yvo Desmedt and the program commitee members for valuable suggestions.

References

1. Divesh Aggarwal and Ueli Maurer. Breaking RSA generically is equivalent to factoring. In Antoine Joux, editor, *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 36–53. Springer, 2009.
2. 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.
3. John Black. The ideal-cipher model, revisited: An uninstantiable blockcipher-based hash function. In Matthew J. B. Robshaw, editor, *FSE*, volume 4047 of *Lecture Notes in Computer Science*, pages 328–340. Springer, 2006.
4. Dan Boneh and Xavier Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *J. Cryptology*, 21(2):149–177, 2008.
5. 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.
6. 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.
7. 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.
8. Daniel R. L. Brown. Breaking RSA may be as difficult as factoring. Cryptology ePrint Archive, Report 2005/380, 2005. http://eprint.iacr.org/.
9. Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.
10. 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.

11. Alexander W. Dent. Adapting the weaknesses of the random oracle model to the generic group model. In Yuliang Zheng, editor, *ASIACRYPT*, volume 2501 of *Lecture Notes in Computer Science*, pages 100–109. Springer, 2002.
12. Marc Fischlin. A note on security proofs in the generic model. In Tatsuaki Okamoto, editor, *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 458–469. Springer, 2000.
13. Tibor Jager and Jörg Schwenk. On the analysis of cryptographic assumptions in the generic ring model, full version. Cryptology ePrint Archive, 2009. http://eprint.iacr.org/.
14. Neal Koblitz and Alfred J. Menezes. Another look at generic groups. pages 13–28, 2006.
15. 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.
16. 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.
17. 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.
18. 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.
19. Ueli M. Maurer and Stefan Wolf. The relationship between breaking the Diffie-Hellman protocol and computing discrete logarithms. *SIAM J. Comput.*, 28(5):1689–1721, 1999.
20. Phong Q. Nguyen and Igor Shparlinski. On the insecurity of a server-aided RSA protocol. In Colin Boyd, editor, *ASIACRYPT*, volume 2248 of *Lecture Notes in Computer Science*, pages 21–35. Springer, 2001.
21. 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.
22. 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.
23. Victor Shoup. *A Computational Introduction to Number Theory and Algebra.* Cambridge University Press, 2005.

A Proof Sketch for Lemma 3

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

$$\left(\frac{|\mathcal{U}[\mathcal{C}]|}{|\mathcal{C}|}\right)^2 \Lambda(P) \geq \Gamma(P) \tag{1}$$

holds. To this end, we will define an auxiliary function $\nu_i(P)$. Then we express $\varGamma(P)$ and $\varLambda(P)$ in terms of $\nu_i(P)$. More precisely, we will upper bound $\varGamma(P)$ by an expression in $\nu_i(P)$ and lower bound $\varLambda(P)$ by an expression in $\nu_i(P)$. The resulting inequality is proven easily by complete induction.

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

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

be the probability that $P(x) \equiv 0 \bmod p_i$ for some prime p_i dividing n and $x \overset{U}{\leftarrow} \mathcal{U}[\mathcal{C}]$. 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 $x \overset{U}{\leftarrow} \mathcal{U}[\mathcal{C}]$ 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 $x \overset{U}{\leftarrow} \mathcal{U}[\mathcal{C}]$ as

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

Bounding $\boldsymbol{\varGamma(P)}$ in terms of $\boldsymbol{\nu_i(P)}$. For independently sampled x, x', we have

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

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

$$\Pr\left[P(x) \in \mathbb{Z}_n^* \mid x \overset{U}{\leftarrow} \mathcal{C}\right] \leq \Pr\left[P(y) \in \mathbb{Z}_n^* \mid y \overset{U}{\leftarrow} \mathcal{U}[\mathcal{C}]\right] \frac{|\mathcal{U}[\mathcal{C}]|}{|\mathcal{C}|}$$

and similarly

$$\Pr\left[P(x) \notin \mathbb{Z}_n^* \mid x \overset{U}{\leftarrow} \mathcal{C}\right] \leq \left(1 - \Pr\left[P(y) \in \mathbb{Z}_n^* \mid y \overset{U}{\leftarrow} \mathcal{U}[\mathcal{C}]\right]\right) \frac{|\mathcal{U}[\mathcal{C}]|}{|\mathcal{C}|}.$$

Therefore we can conclude that

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

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

$$\begin{aligned}\Lambda(P) &= \Pr\left[\gcd(n, P(y)) \notin \{1, n\} \mid y \overset{U}{\leftarrow} \mathcal{C}\right] \\ &= 1 - \prod_{i=1}^{k} \nu_i(P) - \prod_{i=1}^{k}(1 - \nu_i(P)). \qquad (3)\end{aligned}$$

Putting things together. Combining (2) and (3), we see that (1) holds if

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

holds, which is shown easily by complete induction on $k \geq 2$.

B Proof Sketch for Proposition 1

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 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

$$(P_j(x_r) \notin \mathbb{Z}_n^* \text{ and } P_j(x) \in \mathbb{Z}_n^*) \text{ or } (P_j(x) \notin \mathbb{Z}_n^* \text{ and } P_j(x_r) \in \mathbb{Z}_n^*) \qquad (4)$$

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

By (essentially) applying the union bound we can see that for *fixed* P_j this probability is bounded by

$$2m \Pr\left[P_j(x) \notin \mathbb{Z}_n^* \text{ and } P_j(x') \in \mathbb{Z}_n^* \mid x, x' \overset{U}{\leftarrow} \mathcal{C}\right].$$

Using this, we obtain the following bound on the probability that there exists *any* $P_j \sqsubseteq P$ satisfying (4).

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