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
{tibor.jager,joerg.schwenk}@rub.de

July 27, 2011

Abstract

Aggarwal and Maurer (Eurocrypt 2009) proved that breaking RSA is equivalent to factoring in the *generic ring model*. This model captures algorithms that operate on elements of an algebraic ring, by performing only the ring operations and without exploiting properties of a 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 the standard model, where elements of $\mathbb{Z}_n$ are represented by integers modulo n. But is this reasonable?

We prove in the generic ring model that computing the *Jacobi symbol* of an integer modulo n is equivalent to factoring. Since there are simple and efficient non-generic algorithms which compute the Jacobi symbol, this provides an example for a natural computational problem which is hard in the generic ring model, but easy to solve if elements of $\mathbb{Z}_n$ are given in their standard representation as integers. Thus, a proof in the generic ring model is unfortunately not a strong indicator for the hardness of a computational problem in the standard model.

Despite this negative result, generic hardness results still provide a lower complexity bound for a large class of algorithms, namely all algorithms solving a computational problem independent of a given representation of ring elements. Motivated by this fact, we show also that solving the *quadratic residuosity* problem generically is equivalent to factoring.

Keywords: Generic ring model, Jacobi symbol, subset membership problems, idealized models of computation, quadratic residuosity assumption.

Contents

1 Introduction

The security of many cryptosystems relies on assumptions that certain computational problems, mostly from number theory and algebra, are intractable. Therefore it is important to study the validity of these assumptions. Ideally, we would like to show that these assumptions hold in the *standard model*, where algorithms intending to solve a computational problem are modeled as Turing machines with reasonably restricted running time. Unfortunately, proving useful lower complexity bounds in the standard model seems to be impossible with currently available techniques.

However, many important hardness assumptions are based on computational problems defined over algebraic groups. Famous examples are, for instance, the discrete logarithm problem [12], the RSA problem [21], or the quadratic residuosity problem [14]. A natural approach to analyze these assumptions is to consider algorithms solving a given problem by performing only the abstractly defined properties of a group, without exploiting specific properties of the representation of group elements. This model is known as the *generic group model* (see [24, 17], for instance).

Indeed, there are group representations, such as for instance certain elliptic curve groups, for which only very few properties beyond the abstractly defined properties of a group are known. For such representations, the generic group model may be seen as a reasonable abstraction. However, important computational problems, such as the RSA problem and the quadratic residuosity problem, are defined over the multiplicative group $(\mathbb{Z}_n^*, \cdot)$, represented by integers modulo n. This representation exhibits many properties beyond the abstract group definition, such as for instance the fact that the group $(\mathbb{Z}_n^*, \cdot)$ is embedded into the ring $(\mathbb{Z}_n, +, \cdot)$. The generic group model seems too restrictive to provide a tool for a meaningful analysis of such problems.

As an approach to reflect the additional algebraic structure of a ring, the notion of generic groups was extended to generic *rings*. A long line of research [6, 7, 8, 16, 18, 3, 1, 2] analyzes cryptographically relevant computational problems and their relationships in the *generic ring model*. This model is a simple extension of the generic group model, which allows to compute an additional algebraic operation, such that the resulting structure forms a ring. Clearly, when considering hardness assumptions defined over rings, then this idealization is much more appropiate than the generic group model.

Especially the RSA problem was studied extensively in the generic ring model [7, 8, 16, 1, 2]. For instance, Aggarwal and Maurer [1] have shown that solving the RSA problem with generic ring algorithms is equivalent to factoring integers. It is unclear how to interpret this result. A common conclusion drawn in previous works is that a proof in the generic model supports the conjecture that breaking RSA is also equivalent to factoring integers in a standard model of computation. Is this conclusion reasonable?

1.1 Main Contribution

We prove a main theorem stating that solving certain subset membership problems in $\mathbb{Z}_n$ with generic ring algorithms is equivalent to factoring n. This main theorem allows us to provide an example of a computational problem of high cryptographic relevance which is equivalent to factoring in the generic model, but easy to solve if elements of $\mathbb{Z}_n$ are given in their standard representation as integers. Concretely, we show that in the generic ring model computing the *Jacobi symbol* [26, Chapter 12.2] is equivalent to factoring.

In contrast to previous work [8, 16, 3, 1], where integer factorization is reduced to solving *search* problems (in the sense that the algorithm has to search for a certain ring element or integer), we

show that in order to factor n it suffices to be able to solve *decisional* problems in $\mathbb{Z}_n$. Our results do not only cover the case where n is the product of two primes, but hold in the general case where n is the product of *at least* two primes.

A preliminary version of this paper was published at Asiacrypt 2009 [15]. This version contains several improvements, some corrections, and much simpler and thus more comprehensible proofs.

1.2 Interpretation

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 [4], and the generic group model [13, 11]. All these results have in common that they provide somewhat "artificial" computational problems that deviate from standard cryptographic practice.

Note that both the definition and the algebraic properties of the Jacobi symbol are *remarkably* similar to the *quadratic residuosity problem* [14], which builds the foundation of numerous cryptosystems and is widely conjectured to be hard. Thus, in contrast to previous works, the equivalence of computing the Jacobi symbol generically and factoring is an example of a *natural* computational problem that is provably hard in the generic model, but easy to solve if elements of $\mathbb{Z}_n$ are given in their standard representation as integers modulo n. This is an important aspect for interpreting results in the generic ring model, like [7, 8, 16, 3, 1, 2]. Thus, a proof in the generic model is unfortunately not a strong indicator that the considered problem is indeed useful for cryptographic applications.

1.3 Further Results

As another application of our main theorem, we also show that solving the well-known *quadratic residuosity problem* [14] generically is equivalent to factoring. Thus, from a cryptanalytic point of view, we cannot hope to find a representation-independent algorithm solving this problem efficiently, unless factoring integers is easy.

1.4 Related Work

Previous work on fundamental cryptographic assumptions in the generic model considered primarily discrete logarithm-based problems and the RSA problem. Nechaev [20] provided a lower complexity bound on solving the discrete logarithm problem generically. This work was extended by Shoup [24] to a broader and more natural class of algorithms. Starting with Shoup's seminal paper, it was proven that solving the discrete logarithm problem, the Diffie-Hellman problem, and related problems [19, 17, 22] 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 [16] 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 inversions. 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 one-way permutations [6, 18, 3].

2 Preliminaries

2.1 Notation

We denote with $a \xleftarrow{\$} A$ the action of sampling a uniformly random element a from set A. Throughout the paper we let n be the product of at least two different primes, and denote with $n = \prod_{i=1}^{\ell} p_i^{e_i}$ the prime factor decomposition of n such that $\gcd(p_i, p_j) = 1$ for $i \neq j$.

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

2.2 Uniform Closure

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

Definition 1. We say that $\mathcal{U}[\mathcal{C}] \subseteq \mathbb{Z}_n$ is the *uniform closure* of $\mathcal{C} \subseteq \mathbb{Z}_n$, if

$$\mathcal{U}[\mathcal{C}] = \{y \in \mathbb{Z}_n : y = \psi(y_1 \ldots, y_\ell), y_i \in \mathcal{C}_i \text{ for } 1 \leq i \leq \ell\}.$$

Example 1. Let p, q be different primes, $n = pq$, and ψ 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

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

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

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

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

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 or looping, that outputs an element of R. Thus straight line programs are a subclass of generic ring algorithms.

In the sequel we are interested in straight line programs over the particular ring $\mathbb{Z}_n$, where elements are represented by integers. Note that we can not only compute the ring operations addition, subtraction, and multiplication in the ring $\mathbb{Z}_n$, but we also know how to compute *division*, that is, multiplication by multiplicative inverses (if existent), efficiently. In order to make the class of considered algorithms as broad and natural as possible, we therefore include an explicit division operation, though it is generally not explicitly defined for a ring.

The following definition is a simple adaption of [8, Definition 1] to straight line programs that may also compute multiplicative inverses. For our purposes it is sufficient to consider straight-line programs that take as input a *single* ring element $x \in R$, a generalization to algorithms with more input values is straightforward.

Definition 2 (Straight Line Programs)**.** A *straight line program* P of length m over R is a sequence of tuples

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

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

1. Initialize $L_{-1} := 1 \in R$ and $L_0 := x$.
2. For k from 1 to m do:
   - if $\circ_k = /$ and L_{j_k} is not invertible, 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 $x \in R$.

2.4 Generic Ring Algorithms

Similar to straight line programs, generic ring algorithms apply a sequence of ring operations to an input value $x \in R$. In contrast to straight line programs, which 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 $x \in R$. 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 $P_j(x)$ is not invertible, 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 $P_i(x) = P_j(x)$ and false otherwise.

In order to perform computations, the algorithm makes a query of the form

$$(i, j, \circ) \in \{-1, \ldots, |P|\} \times \{-1, \ldots, |P|\} \times \{+, -, \cdot, /\}$$

to $\mathcal{O}$. This query is processed as follows. Whenever the algorithm submits $(i, j, \circ)$, 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 On Adopting Proving Techniques from the Generic Group Model

The generic ring model (GRM) is an extension of the generic group model (GGM), see [24], for instance. Despite many similarities, 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. [24, 17, 16], for instance) introduces a simulation game where group elements are replaced with polynomials that are (implicitly) evaluated with 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 work [24], 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-Zippel Lemma [27, 23]. Usually the bound is useful if the number of roots is sufficiently small. Rupp et al. [22] 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.

This technique could be adopted to the generic ring model, by observing that straight line programs are algorithms which compute *rational functions*, which in turn can be represented by tuples of polynomials. However, here we face a problem. 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 [5]). However, in the generic ring model we explicitly allow for multiplication operations, and we do not want to put a restriction on the number of allowed multiplications, in order to keep the model as realistic as possible. Thus, by repeated squaring an algorithm may compute polynomials of exponential degree. In this case applying the Schwartz-Zippel Lemma does not yield a useful bound on the number of roots.

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

In the following we will state some 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 $P_j(x) \in \mathbb{Z}_n \setminus \mathbb{Z}_n^*$.*

Proof. $P(x) = \perp$ means that there exists an SLP-step $(i, j, \circ) \in P$ such that $\circ = /$ and $L_j = P_j(x) \in \mathbb{Z}_n \setminus \mathbb{Z}_n^*$. 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 straight line program P with a random value $y \xleftarrow{\$} \mathcal{U}[\mathcal{C}]$ and computing $\gcd(n, P(y))$, relative to the probability that $P(x') \in \mathbb{Z}_n \setminus \mathbb{Z}_n^*$ and $P(x) \in \mathbb{Z}_n^*$ for randomly chosen $x, x' \xleftarrow{\$} \mathcal{C}$.

Lemma 3. *Let $n = \prod_{i=1}^{\ell} p_i^{e_i}$ with $\ell \geq 2$. For any straight line program P and $\mathcal{C} \subseteq \mathbb{Z}_n$ holds that*

$$\begin{aligned}&\Pr\left[P(x') \in \mathbb{Z}_n \setminus \mathbb{Z}_n^* \text{ and } P(x) \in \mathbb{Z}_n^* : x, x' \xleftarrow{\$} \mathcal{C}\right]\\ \leq &\ell \cdot \Pr\left[\gcd(n, P(y)) \notin \{1, n\} : y \xleftarrow{\$} \mathcal{U}[\mathcal{C}]\right].\end{aligned}$$

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 $y \xleftarrow{\$} \mathcal{U}[\mathcal{C}]$ for two given straight line programs P and Q, relative to the probability that $P(x) \equiv_n Q(x)$ and $P(x') \not\equiv_n Q(x')$ for random $x, x' \xleftarrow{\$} \mathcal{C}$.

Lemma 4. *Let $n = \prod_{i=1}^{\ell} p_i^{e_i}$ with $\ell \geq 2$. For any pair (P, Q) of straight line programs and $\mathcal{C} \subseteq \mathbb{Z}_n$ holds that*

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

Before giving proofs for Lemmas 3 and 4, we will give some intuition in the following section.

3.1 Some Intuition for Lemma 3 and 4

Simplifying a little, Lemma 3 and 4 state *essentially* 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 $y \xleftarrow{\$} \mathcal{U}[\mathcal{C}]$ and computing $\gcd(n, P(y))$.[1] At a first glance this seems counterintuitive.

For instance, consider the case $\mathcal{C} = \mathbb{Z}_n$, then we have $\mathcal{U}[\mathcal{C}] = \mathbb{Z}_n$. Assume a straight line program P mapping about one 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 $y \xleftarrow{\$} \mathbb{Z}_n$ yields only trivial factors of n, hence this seems to be a counterexample to Lemma 3 and 4. However, 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, but $p, q > 1$). By the Chinese Remainder Theorem, the ring $\mathbb{Z}_n$ is isomorphic to $\mathbb{Z}_p \times \mathbb{Z}_q$. Let $\psi : \mathbb{Z}_p \times \mathbb{Z}_q \to \mathbb{Z}_n$ 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 ψ is a ring-isomorphism and P performs only ring operations, it holds that

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

and

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

[1] In case of Lemma 3 note that $P(x) \in \mathbb{Z}_n^*$ and $P(x') \in \mathbb{Z}_n \setminus \mathbb{Z}_n^*$ 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 $S(x) := P(x) - Q(x)$.

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 $c = \psi(x' \bmod p, x \bmod q)$ and $d = \psi(x \bmod p, x' \bmod q)$. Evaluating P with c or d yields

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

or

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

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

Generalizing this idea, putting it into a more precise and formal language, and handling some technical obstacles (e.g. the fact that simulator and factoring algorithm sample from *subsets* of $\mathbb{Z}_n$, which made it necessary to define the *uniform closure* of subsets of $\mathbb{Z}_n$), we obtain the proofs given in Appendices 3.2 and 3.3.

3.2 Proof of Lemma 3

First, observe that $P(x') \in \mathbb{Z}_n \setminus \mathbb{Z}_n^*$ implies, that there exists at least one $i \in \{1, \ldots, \ell\}$ such that $P(x') \equiv 0 \bmod p_i$, while $P(x) \in \mathbb{Z}_n^*$ implies that $P(x) \in \mathbb{Z}_{p_i}^*$ for all $i \in \{1, \ldots, \ell\}$. Thus, we have

$$\begin{aligned}
&\Pr\left[P(x') \in \mathbb{Z}_n \setminus \mathbb{Z}_n^* \text{ and } P(x) \in \mathbb{Z}_n^* : x, x' \xleftarrow{\$} \mathcal{C}\right] \\
\leq &\ell \cdot \max_{1 \leq i \leq \ell} \left\{\Pr[P(x') \equiv 0 \bmod p_i \text{ and } P(x) \in \mathbb{Z}_{p_j}^* \text{ for all } j \in \{1, \ldots, \ell\} : x, x' \xleftarrow{\$} \mathcal{C}]\right\} \\
\leq &\ell \cdot \max_{1 \leq i \leq \ell} \left\{\Pr[P(x') \equiv 0 \bmod p_i \text{ and } P(x) \in \mathbb{Z}_{p_j}^* \text{ for all } j \in \{1, \ldots, \ell\} \setminus \{i\} : x, x' \xleftarrow{\$} \mathcal{C}]\right\}.
\end{aligned}$$

Note furthermore, that we have $P(x) \equiv P(x \bmod p_i) \bmod p_i$. This is due to the fact that the projection $\tau_i : \mathbb{Z}_n \to \mathbb{Z}_{p_i^{e_i}}$, mapping $x \mapsto x \bmod p_i^{e_i}$, is a ring-homomorphism, and P performs only ring operations. Thus we have

$$\begin{aligned}
&\ell \cdot \max_{1 \leq i \leq \ell} \left\{\Pr[P(x') \equiv 0 \bmod p_i \text{ and } P(x) \in \mathbb{Z}_{p_j}^* \text{ for all } j \in \{1, \ldots, \ell\} \setminus \{i\} : x, x' \xleftarrow{\$} \mathcal{C}]\right\} \\
= &\ell \cdot \max_{1 \leq i \leq \ell} \left\{\Pr[P(x_i') \equiv 0 \bmod p_i \text{ and } P(x_j) \in \mathbb{Z}_{p_j}^* \text{ for all } j \in \{1, \ldots, \ell\} \setminus \{i\} : x, x' \xleftarrow{\$} \mathcal{C}]\right\},
\end{aligned}$$

where $x_i' := x' \bmod p_i^{e_i}$ and $x_j := x \bmod p_j^{e_j}$.

Next, observe that sampling *two* values $x, x' \xleftarrow{\$} \mathcal{C}$ and computing $x_i' = x' \bmod p_i^{e_i}$ and $x_j = x \bmod p_j^{e_j}$ for all $j \in \{1, \ldots, \ell\} \setminus \{i\}$ is equivalent to sampling a *single* value $y \xleftarrow{\$} \mathcal{U}[\mathcal{C}]$ and computing $y_i = y \bmod p_i^{e_i}$ and $y_j = y \bmod p_j^{e_j}$ for all $j \in \{1, \ldots, \ell\} \setminus \{i\}$, see Lemma 1. Thus we have

$$\begin{aligned}
&\ell \cdot \max_{1 \leq i \leq \ell} \left\{\Pr[P(x_i') \equiv 0 \bmod p_i \text{ and } P(x_j) \in \mathbb{Z}_{p_j}^* \text{ for all } j \in \{1, \ldots, \ell\} \setminus \{i\} : x, x' \xleftarrow{\$} \mathcal{C}]\right\} \\
= &\ell \cdot \max_{1 \leq i \leq \ell} \left\{\Pr\left[P(y_i) \equiv 0 \bmod p_i \text{ and } P(y_j) \in \mathbb{Z}_{p_j}^* \text{ for all } j \in \{1, \ldots, \ell\} \setminus \{i\} : y \xleftarrow{\$} \mathcal{U}[\mathcal{C}]\right]\right\}.
\end{aligned}$$

If we use once again that $\tau_i : \mathbb{Z}_n \to \mathbb{Z}_{p_i^{e_i}}$ is a ring-homomorphism, we obtain that

$$\begin{aligned}
&\ell \cdot \max_{1\le i\le \ell} \left\{ \Pr\left[P(y_i) \equiv 0 \bmod p_i \text{ and } P(y_j) \in \mathbb{Z}_{p_j}^* \text{ for all } j \in \{1,\ldots,\ell\} \setminus \{i\} : y \xleftarrow{\$} \mathcal{U}\left[\mathcal{C}\right] \right] \right\} \\
=&\ell \cdot \max_{1\le i\le \ell} \left\{ \Pr\left[P(y) \equiv 0 \bmod p_i \text{ and } P(y) \in \mathbb{Z}_{p_j}^* \text{ for all } j \in \{1,\ldots,\ell\} \setminus \{i\} : y \xleftarrow{\$} \mathcal{U}\left[\mathcal{C}\right] \right] \right\}.
\end{aligned}$$

Finally, note that we can clearly find a factor of n by computing $\gcd(n, P(y))$ for uniformly random $y \xleftarrow{\$} \mathcal{U}\left[\mathcal{C}\right]$ with probability at least

$$\max_{1\le i\le \ell} \left\{ \Pr\left[P(y) \equiv 0 \bmod p_i \text{ and } P(y) \in \mathbb{Z}_{p_j}^* \text{ for all } j \in \{1,\ldots,\ell\} \setminus \{i\} : y \xleftarrow{\$} \mathcal{U}\left[\mathcal{C}\right] \right] \right\},$$

which implies that

$$\begin{aligned}
&\ell \cdot \max_{1\le i\le \ell} \left\{ \Pr\left[P(y) \equiv 0 \bmod p_i \text{ and } P(y) \in \mathbb{Z}_{p_j}^* \text{ for all } j \in \{1,\ldots,\ell\} \setminus \{i\} : y \xleftarrow{\$} \mathcal{U}\left[\mathcal{C}\right] \right] \right\} \\
\le&\ell \cdot \Pr\left[\gcd(n, P(y)) \notin \{1, n\} : y \xleftarrow{\$} \mathcal{U}\left[\mathcal{C}\right] \right].
\end{aligned}$$

3.3 Proof of Lemma 4

In the sequel let $\Delta(x) := P(x) - Q(x)$ and write $a_i := a \bmod p_i^{e_i}$ for all $a \in \{x', x, y\}$. Then, with the same arguments as in the proof of Lemma 3, we have

$$\begin{aligned}
&\Pr\left[\Delta(x) \equiv_n 0 \text{ and } \Delta(x') \not\equiv_n 0 : x, x' \xleftarrow{\$} \mathcal{C} \right] \\
\le&\ell \cdot \max_{1\le i\le \ell} \left\{ \Pr\left[\Delta(x) \equiv 0 \bmod p_j^{e_j} \text{ for all } j \in \{1,\ldots,\ell\} \text{ and } \Delta(x') \not\equiv 0 \bmod p_i^{e_i} : x, x' \xleftarrow{\$} \mathcal{C} \right] \right\} \\
\le&\ell \cdot \max_{1\le i\le \ell} \left\{ \Pr\left[\Delta(x) \equiv 0 \bmod p_j^{e_j} \text{ for all } j \in \{1,\ldots,\ell\} \setminus \{i\} \text{ and } \Delta(x') \not\equiv 0 \bmod p_i^{e_i} : x, x' \xleftarrow{\$} \mathcal{C} \right] \right\} \\
=&\ell \cdot \max_{1\le i\le \ell} \left\{ \Pr\left[\Delta(x_j) \equiv 0 \bmod p_j^{e_j} \text{ for all } j \in \{1,\ldots,\ell\} \setminus \{i\} \text{ and } \Delta(x_i') \not\equiv 0 \bmod p_i^{e_i} : x, x' \xleftarrow{\$} \mathcal{C} \right] \right\} \\
=&\ell \cdot \max_{1\le i\le \ell} \left\{ \Pr\left[\Delta(y_j) \equiv 0 \bmod p_j^{e_j} \text{ for all } j \in \{1,\ldots,\ell\} \setminus \{i\} \text{ and } \Delta(y_i) \not\equiv 0 \bmod p_i^{e_i} : y \xleftarrow{\$} \mathcal{U}\left[\mathcal{C}\right] \right] \right\} \\
=&\ell \cdot \max_{1\le i\le \ell} \left\{ \Pr\left[\Delta(y) \equiv 0 \bmod p_j^{e_j} \text{ for all } j \in \{1,\ldots,\ell\} \setminus \{i\} \text{ and } \Delta(y) \not\equiv 0 \bmod p_i^{e_i} : y \xleftarrow{\$} \mathcal{U}\left[\mathcal{C}\right] \right] \right\} \\
\le&\ell \cdot \Pr\left[\gcd(n, P(y) - Q(y)) \notin \{1, n\} : y \xleftarrow{\$} \mathcal{U}\left[\mathcal{C}\right] \right].
\end{aligned}$$

4 Subset Membership Problems in the Generic Ring Model

Definition 3. Let $\mathcal{C} \subseteq \mathbb{Z}_N$ and $\mathcal{V} \subseteq \mathcal{C}$ with $|\mathcal{V}| > 1$. The *subset membership problem* defined by $(\mathcal{C}, \mathcal{V})$ is: given $x \xleftarrow{\$} \mathcal{C}$, decide whether $x \in \mathcal{V}$.

In the sequel we will consider only subset membership problems such that $|\mathcal{C}| = 2 \cdot |\mathcal{V}|$.

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

subset membership challenge $x \xleftarrow{\$} \mathcal{C}$ as input. We say that $\mathcal{A}$ wins the game, if $x \in \mathcal{C} \setminus \mathcal{V}$ and $\mathcal{A}^{\mathcal{O}_{\text{smp}}}(n) = 0$, or $x \in \mathcal{V}$ and $\mathcal{A}^{\mathcal{O}_{\text{smp}}}(n) = 1$.

Note that $\Pr[x \in \mathcal{V} : x \xleftarrow{\$} \mathcal{C}] = 1/2$, since x is chosen uniformly random and we have $|\mathcal{C}| = 2 \cdot |\mathcal{V}|$. Note also that any algorithm for a given subset membership problem $(\mathcal{C}, \mathcal{V})$ has at least the trivial success probability $1/2$ by guessing. For an algorithm solving the subset membership problem given by $(\mathcal{C}, \mathcal{V})$ with success probability $\Pr[\mathcal{S}]$, we denote with

$$\mathsf{Adv}_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{\text{smp}}}(n)) := |\Pr[\mathcal{S}] - 1/2|$$

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

Theorem 1. *Let $n = \prod_{i=1}^{\ell} p_i^{e_i}$. Let $(\mathcal{C}, \mathcal{V})$ be a subset membership problem such that $2 \cdot |\mathcal{V}| = |\mathcal{C}|$. For any generic ring algorithm $\mathcal{A}$ solving the subset membership problem 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 non-trivial factor of n with success probability at least*

$$\frac{\mathsf{Adv}_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{\text{smp}}}(n))}{2\ell(m^2 + 4m + 3)}$$

by running $\mathcal{A}$ once, performing at most $2m$ additional operations in $\mathbb{Z}_n$ and at most $(m+2)^2$ gcd-computations on $\lceil \log_2 n \rceil$-bit numbers, and sampling one random element 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 [25, Lemma 1]). We derive a bound on $\Pr[\mathcal{S}_{\text{sim}}]$ and describe a factoring algorithm whose success probability is lower bound by $\Pr[\mathcal{F}]$.

4.1 Introducing a Simulation Oracle

We replace oracle $\mathcal{O}_{\text{smp}}$ with a simulator $\mathcal{O}_{\text{sim}}$. $\mathcal{O}_{\text{sim}}$ receives $x \xleftarrow{\$} \mathcal{C}$ 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 additionally samples a random value $x' \xleftarrow{\$} \mathcal{C}$ at the beginning of the game. Moreover, it replaces the procedures $\mathsf{test}()$ and $\mathsf{equal}()$ with procedures $\mathsf{testsim}()$ and $\mathsf{equalsim}()$.

- The $\mathsf{testsim}()$-procedure returns false if $\circ = /$ and $P_j(x') \notin \mathbb{Z}_n^*$, and true otherwise (note that we may have $P_j(x') = \perp$, where $\perp \notin \mathbb{Z}_n^*$).
- The $\mathsf{equalsim}()$-procedure returns true if $P_i(x') = P_j(x')$ and false otherwise.

Note that the simulator $\mathcal{O}_{\text{sim}}$ proceeds exactly like $\mathcal{O}_{\text{smp}}$, except that it samples a random x' at the beginning of the game, and uses x' instead of x in invertability and equality tests. Therefore 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

$$\Pr[\mathcal{S}_{\text{sim}}] = 1/2.$$

4.2 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.

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

$$\begin{aligned}
\Pr[\mathcal{F}] &= \Pr[\mathcal{F}_{\mathsf{test}} \cup \mathcal{F}_{\mathsf{equal}}] \\
&= \Pr[\mathcal{F}_{\mathsf{test}}] + \Pr[\mathcal{F}_{\mathsf{equal}}] - \Pr[\mathcal{F}_{\mathsf{test}} \cap \mathcal{F}_{\mathsf{equal}}] \\
&= \Pr[\mathcal{F}_{\mathsf{test}}] + \Pr[\mathcal{F}_{\mathsf{equal}}] - \Pr[\mathcal{F}_{\mathsf{equal}} \mid \mathcal{F}_{\mathsf{test}}] \cdot \Pr[\mathcal{F}_{\mathsf{test}}] \\
&= \Pr[\mathcal{F}_{\mathsf{test}}] + \Pr[\mathcal{F}_{\mathsf{equal}} \mid \mathcal{F}_{\mathsf{test}}] \cdot \Pr[\mathcal{F}_{\mathsf{test}}] + \Pr[\mathcal{F}_{\mathsf{equal}} \mid \neg\mathcal{F}_{\mathsf{test}}] \cdot \Pr[\neg\mathcal{F}_{\mathsf{test}}] \\
&\qquad - \Pr[\mathcal{F}_{\mathsf{equal}} \mid \mathcal{F}_{\mathsf{test}}] \cdot \Pr[\mathcal{F}_{\mathsf{test}}] \\
&= \Pr[\mathcal{F}_{\mathsf{test}}] + \Pr[\mathcal{F}_{\mathsf{equal}} \mid \neg\mathcal{F}_{\mathsf{test}}] \cdot \Pr[\neg\mathcal{F}_{\mathsf{test}}] \\
&\leq \Pr[\mathcal{F}_{\mathsf{test}}] + \Pr[\mathcal{F}_{\mathsf{equal}} \mid \neg\mathcal{F}_{\mathsf{test}}]
\end{aligned}$$

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

4.2.1 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 vice versa. This happens only if there exists $P_j \sqsubseteq P$ such that

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

Note that we may have $P_j(\tilde{x}) = \perp \notin \mathbb{Z}_n^*$ for $\tilde{x} \in \{x, x'\}$. However, we can simplify our analysis a little by applying Lemma 2. The existence of $P_j \sqsubseteq P$ such that $P_j(\tilde{x}) = \perp$ implies the existence of $P_k \sqsubseteq P_j$ such that $P_k(\tilde{x}) \in \mathbb{Z}_n \setminus \mathbb{Z}_n^*$. Hence, $\mathcal{F}_{\mathsf{test}}$ occurs only if there exists $P_j \sqsubseteq P$ such that

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

Note that for one *fixed* P_j we have

$$\begin{aligned}
&\Pr\left[(P_j(x') \in \mathbb{Z}_n \setminus \mathbb{Z}_n^* \text{ and } P_j(x) \in \mathbb{Z}_n^*) \text{ or } (P_j(x) \in \mathbb{Z}_n \setminus \mathbb{Z}_n^* \text{ and } P_j(x') \in \mathbb{Z}_n^*) : x, x' \xleftarrow{\$} \mathcal{C}\right] \\
\leq &2 \cdot \Pr\left[P_j(x) \in \mathbb{Z}_n \setminus \mathbb{Z}_n^* \text{ and } P_j(x') \in \mathbb{Z}_n^* : x, x' \xleftarrow{\$} \mathcal{C}\right].
\end{aligned}$$

Thus, by taking the maximum probability over all P_j, we get

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

4.2.2 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 vice versa. This happens only if there exists $P_i, P_j \sqsubseteq P$ such that

$$(P_i(x) = P_j(x) \text{ and } P_i(x') \neq P_j(x')) \text{ or } (P_i(x) \neq P_j(x) \text{ and } P_i(x') = P_j(x')). \tag{1}$$

Note that we want to consider the event $\mathcal{F}_{\mathsf{equal}}$, conditioned on that event $\mathcal{F}_{\mathsf{test}}$ did not occur. Therefore we may assume that there exists no straight line program $P_k \sqsubseteq P$ such that $P_k(x) = \bot$ and $P_k(x') \neq \bot$, or vice versa. This allows us to simplify our analysis slightly, since in this case (1) is equivalent to

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

Thus, like in the previous section, we have

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

4.2.3 Bounding the Probability of $\mathcal{F}$

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

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

4.3 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

$$1/2 + \mathsf{Adv}_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{\mathrm{smp}}}) = \Pr[\mathcal{S}] \leq \Pr[\mathcal{S}_{\mathrm{sim}}] + \Pr[\mathcal{F}] \leq 1/2 + \Pr[\mathcal{F}],$$

which implies

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

4.4 The Factoring Algorithm

Consider a factoring algorithm $\mathcal{B}$ which runs $\mathcal{A}$ as a subroutine by implementing the generic ring oracle for $\mathcal{A}$. That is, it performs all computations queried by $\mathcal{A}$ to a random element $x \in \mathcal{C}$.

In parallel, $\mathcal{B}$ applies all queried operations to $y \in \mathbb{Z}_n$, where $y \xleftarrow{\$} \mathcal{U}[\mathcal{C}]$ is chosen uniformly random at the beginning of the game. Moreover, each time a triple $(i, j, \circ)$ is appended to P, $\mathcal{B}$ computes

- $\gcd(P(y), n)$, and
- $\gcd(P(y) - P_i(y), n)$ for all $i \in \{-1, \ldots, |P| - 1\}$.

4.4.1 Running time

$\mathcal{B}$ samples one random value y from $\mathcal{U}[\mathcal{C}]$. Since by assumption $\mathcal{A}$ submits m queries, $\mathcal{B}$ has to perform at most $2m$ operations in $\mathbb{Z}_n$ in order to perform all computations queried by $\mathcal{A}$ simultaneously on $x \in \mathcal{C}$ and $y \in \mathcal{U}[\mathcal{C}]$. In addition, $\mathcal{B}$ performs at most $(m+2)^2$ gcd-computations on $\lceil \log_2 n \rceil$-bit numbers.

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

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

Let $\gamma_1 := \max_{0 \le k \le m}\{\Pr[P_k(x) \in \mathbb{Z}_n \setminus \mathbb{Z}_n^* \text{ and } P_k(x') \in \mathbb{Z}_n^* : x, x' \xleftarrow{\$} \mathcal{C}]\}$, then by Lemma 3 algorithm $\mathcal{B}$ finds a factor in this step with probability at least γ_1/ℓ.

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

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

Let $\gamma_2 := \max_{-1 \le i < j \le m}\{\Pr[P_i(x) \equiv_n P_j(x) \text{ and } P_i(x') \not\equiv_n P_j(x') : x, x' \xleftarrow{\$} \mathcal{C}]\}$, then by Lemma 4 algorithm $\mathcal{B}$ succeeds in this step with probability at least γ_2/ℓ.

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

4.4.3 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

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

which implies the inequality

$$\gamma \geq \frac{\mathsf{Adv}_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{\mathrm{smp}}}(n))}{2(m^2+4m+3)}.$$

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

$$\frac{\mathsf{Adv}_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{\mathrm{smp}}}(n))}{2\ell(m^2+4m+3)}.$$

5 Applications

In this section, we apply our general theorem to two specific subset membership problems with high cryptographic relevance. The first application shows that computing Jacobi symbols modulo n with generic ring algorithms is as hard as factoring n.

Then we apply our main theorem to the well-known quadratic residuosity problem. It is unknown whether there exists an efficient algorithm for this problem, and it is widely conjectured that this problem is hard if factoring the modulus n is hard. We show that any algorithm solving this problem efficiently needs to exploit specific properties of the representation of elements of $\mathbb{Z}_n$ (possibly in a way similar to the known algorithms for computing Jacobi symbols).

5.1 Computing the Jacobi Symbol with Generic Ring Algorithms

In order to define and analyze the *Jacobi symbol* we need the *Legendre symbol*. The Legendre symbol $L(x|p)$ of an integer x modulo a prime p is defined as

$$L(x|p) = \begin{cases} 0, \text{ if } \gcd(x,p) \neq 1, \\ 1, \text{ if } \gcd(x,p) = 1 \text{ and } x \text{ has a square root modulo } p, \\ -1, \text{ if } \gcd(x,p) = 1 \text{ and } x \text{ has no square root modulo } p. \end{cases}$$

The Jacobi symbol generalizes the Legendre symbol from prime to composite moduli. If $n = \prod_{i=1}^{l} p_i^{e_i}$ is the prime factor decomposition of n, then the Jacobi symbol of an integer x modulo n is defined as

$$J(x|n) := \prod_{i=1}^{\ell} L(x|p_i)^{e_i}. \tag{2}$$

There exists an algorithm computing the Jacobi symbol $J(x|n)$ efficiently, even if the factorization of n is not given, using the law of quadratic reciprocity, see [26, Chapter 12.3] for instance.

Properties of the Jacobi symbol. In the sequel we will consider the problem of computing the Jacobi symbol as a subset membership problem over $\mathbb{Z}_n$. To this end, let us summarize some properties of the Jacobi symbol, which will become relevant.

1. Note that for $x \in \mathbb{Z}_n^*$ we have $J(x|n) \in \{1,-1\}$. Let

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

   be the set of elements of $\mathbb{Z}_n$ having Jacobi symbol 1. Thus, we can perceive the problem of computing the Jacobi symbol as a subset membership problem $(\mathcal{C},\mathcal{V})$ over $\mathbb{Z}_n$ with $\mathcal{C} = \mathbb{Z}_n^*$ and $\mathcal{V} = J_n$.

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

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

where $\varphi(\cdot)$ is the Euler totient function [26, Chapter 2.6]. This is an immediate consequence of the definition of the Jacobi symbol.

Now we are ready to apply our main theorem to show that there is no efficient *generic* ring algorithm computing the Jacobi symbol efficiently, unless factoring n is easy.

Theorem 2. *Let $n = \prod_{i=1}^{\ell} p_i^{e_i}$. 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 non-trivial factor of n with probability at least*

$$\frac{\mathsf{Adv}_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{\mathrm{smp}}}(n))}{2\ell(m^2+4m+3)}$$

by running $\mathcal{A}$ once, performing at most $2m$ additional operations in $\mathbb{Z}_n$ and at most $(m+2)^2$ gcd-computations on $\lceil \log_2 n \rceil$-bit numbers, and sampling one random element from $\mathbb{Z}_n^*$.

Proof. If n is a square in $\mathbb{N}$ then the theorem is trivially true, since in this case it is easy to find a factor of n. Therefore we only need to consider the case where n is not a square.

Note that in this case we have $2 \cdot |J_n| = \varphi(n) = |\mathbb{Z}_n^*| = |\mathcal{C}|$. Furthermore, it holds that $\mathcal{U}[\mathcal{C}] = \mathcal{U}[\mathbb{Z}_n^*] = \mathbb{Z}_n^*$. The result follows by applying Theorem 1. □

5.2 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.

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

It holds 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. The *quadratic residuosity problem* [14] is the subset membership problem given by $\mathcal{C} = J_n$ and $\mathcal{V} = \mathsf{QR}_n$.

If $n = pq$ is the product of two different odd primes, then it holds that $|\mathsf{QR}_n| = \varphi(n)/4$ and $|J_n| = \varphi(n)/2$ (see for instance [26, p.348]). Thus, for $n = pq$ we have $2 \cdot |\mathcal{V}| = |\mathcal{C}|$.

Given the factorization of an integer n, the quadratic residuosity problem in $\mathbb{Z}_n$ can be solved easily by a generic ring algorithm. Thus, in order to show the equivalence of generic quadratic residuosity and factoring, we have to prove the following theorem.

Theorem 3. *Let $n = pq$ be the product of two different odd primes. Suppose there exist a generic ring algorithm $\mathcal{A}$ solving the subset membership problem given by $(\mathcal{C}, \mathcal{V})$ with $\mathcal{C} = J_n$ and $\mathcal{V} = \mathsf{QR}_n$*

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 non-trivial factor of* n *with probability at least*

$$\frac{\mathsf{Adv}_{(\mathcal{C},\mathcal{V})}(\mathcal{A}^{\mathcal{O}_{\mathrm{smp}}}(n))}{2\ell(m^2+4m+3)}$$

by running $\mathcal{A}$ *once, performing at most* $2m$ *additional operations in* $\mathbb{Z}_n$ *and at most* $(m+2)^2$ gcd*-computations on* $\lceil \log_2 n \rceil$*-bit numbers, and sampling one random element from* $\mathbb{Z}_n^*$.

Proof. If $n = pq$ is the product of two different odd primes, then we have $\mathcal{U}[\mathcal{C}] = \mathcal{U}[J_n] = \mathbb{Z}_n^*$ and $|\mathcal{C}| = |J_n| = 2 \cdot |\mathsf{QR}_n| = 2 \cdot |\mathcal{V}|$. Thus the claim follows by applying Theorem 1. □

Acknowledgements. We would like to thank Andy Rupp and Sven Schäge for helpful discussions, Yvo Desmedt and the program commitee members of Asiacrypt 2009 for valuable suggestions, and the anonymous referees of the Journal of Cryptology for providing very detailed and helpful comments leading to various improvements.

References

[1] Divesh Aggarwal and Ueli Maurer. Breaking RSA generically is equivalent to factoring. In Antoine Joux, editor, *Advances in Cryptology – EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 36–53, Cologne, Germany, April 26–30, 2009. Springer, Berlin, Germany.

[2] Divesh Aggarwal, Ueli Maurer, and Igor Shparlinski. The equivalence of strong RSA and factoring in the generic ring model of computation. In *Workshop on Coding and Cryptography - WCC 2011*, Lecture Notes in Computer Science. Springer-Verlag, April 2011.

[3] 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 2008: 35th International Colloquium on Automata, Languages and Programming, Part II*, volume 5126 of *Lecture Notes in Computer Science*, pages 437–448, Reykjavik, Iceland, July 7–11, 2008. Springer, Berlin, Germany.

[4] John Black. The ideal-cipher model, revisited: An uninstantiable blockcipher-based hash function. In Matthew J. B. Robshaw, editor, *Fast Software Encryption – FSE 2006*, volume 4047 of *Lecture Notes in Computer Science*, pages 328–340, Graz, Austria, March 15–17, 2006. Springer, Berlin, Germany.

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

[6] Dan Boneh and Richard J. Lipton. Algorithms for black-box fields and their application to cryptography (extended abstract). In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 283–297, Santa Barbara, CA, USA, August 18–22, 1996. Springer, Berlin, Germany.

[7] Dan Boneh and Ramarathnam Venkatesan. Breaking RSA may not be equivalent to factoring. In Kaisa Nyberg, editor, *Advances in Cryptology – EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 59–71, Espoo, Finland, May 31 – June 4, 1998. Springer, Berlin, Germany.

[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, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 256–271, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Berlin, Germany.

[11] Alexander W. Dent. Adapting the weaknesses of the random oracle model to the generic group model. In Yuliang Zheng, editor, *Advances in Cryptology – ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 100–109, Queenstown, New Zealand, December 1–5, 2002. Springer, Berlin, Germany.

[12] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.

[13] Marc Fischlin. A note on security proofs in the generic model. In Tatsuaki Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 458–469, Kyoto, Japan, December 3–7, 2000. Springer, Berlin, Germany.

[14] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[15] Tibor Jager and Jörg Schwenk. On the analysis of cryptographic assumptions in the generic ring model. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 399–416, Tokyo, Japan, December 6–10, 2009. Springer, Berlin, Germany.

[16] Gregor Leander and Andy Rupp. On the equivalence of RSA and factoring regarding generic ring algorithms. In Xuejia Lai and Kefei Chen, editors, *Advances in Cryptology – ASIACRYPT 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 241–251, Shanghai, China, December 3–7, 2006. Springer, Berlin, Germany.

[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 Dominik Raub. Black-box extension fields and the inexistence of field-homomorphic one-way permutations. In Kaoru Kurosawa, editor, *Advances in Cryptology – ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 427–443, Kuching, Malaysia, December 2–6, 2007. Springer, Berlin, Germany.

[19] 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, Espoo, Finland, May 31 – June 4, 1998. Springer, Berlin, Germany.

[20] V. I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55(2):165–172, 1994.

[21] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[22] 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, *Advances in Cryptology – ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 489–505, Melbourne, Australia, December 7–11, 2008. Springer, Berlin, Germany.

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

[24] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266, Konstanz, Germany, May 11–15, 1997. Springer, Berlin, Germany.

[25] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. `http://eprint.iacr.org/`.

[26] Victor Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, second edition, 2008.

[27] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Symbolic and Algebraic Computation*, pages 216–226, 1979.