Generically Speeding-Up Repeated Squaring is Equivalent to Factoring: Sharp Thresholds for All Generic-Ring Delay Functions

Lior Rotem[⋆,⋆⋆] and Gil Segev[⋆]

School of Computer Science and Engineering,
Hebrew University of Jerusalem, Jerusalem 91904, Israel.
{lior.rotem,segev}@cs.huji.ac.il

Abstract. Despite the fundamental importance of delay functions, repeated squaring in RSA groups (Rivest, Shamir and Wagner '96) is the only candidate offering both a useful structure and a realistic level of practicality. Somewhat unsatisfyingly, its sequentiality is provided directly by assumption (i.e., the function is assumed to be a delay function).

We prove sharp thresholds on the sequentiality of all generic-ring delay functions relative to an RSA modulus based on the hardness of factoring in the standard model. In particular, we show that generically speeding-up repeated squaring (even with a preprocessing stage and any polynomial number parallel processors) is equivalent to factoring.

More generally, based on the (essential) hardness of factoring, we prove that any generic-ring function is in fact a delay function, admitting a sharp sequentiality threshold that is determined by our notion of *sequentiality depth*. Moreover, we show that generic-ring functions admit not only sharp sequentiality thresholds, but also sharp pseudorandomness thresholds.

1 Introduction

The recent and exciting notion of a verifiable delay function, introduced by Boneh et al. [BBB+18], and the classic notion of time-lock puzzles, introduced by Rivest, Shamir and Wagner [RSW96], are gaining significant interest due to a host of thrilling applications. These include, for example, randomness beacons, resource-efficient blockchains, proofs of replication and computational timestamping. A fundamental notion underlying both of these notions is that of a cryptographic delay function: For a delay parameter T, evaluating a delay function on a randomly-chosen input should require at least T sequential steps (even with a polynomial number of parallel processors and with a preprocessing stage), yet the function can be evaluated on any input in time polynomial in T.

⋆ Supported by the European Union's Horizon 2020 Framework Program (H2020) via an ERC Grant (Grant No. 714253).

⋆⋆ Supported by the Adams Fellowship Program of the Israel Academy of Sciences and Humanities.

A delay function can be easily constructed by iterating a cryptographic hash function. A major benefit of this construction is that its sequentiality is supported by an idealized-model proof of security: When the hash function is modeled as a random oracle, its sequentiality is guaranteed in an information-theoretic sense. Alas, the lack of structure exhibited by this construction seems to disable its practical use for realizing time-lock puzzles or verifiable delay functions. Specifically, for time-lock puzzles, iterated hashing does not seem to admit sufficiently fast generation of input-output pairs [MMV11]; and for verifiable delay functions it does not seem to enable sufficiently fast verification.[1]

The only known construction of a delay function that offers both a useful structure for realizing time-lock puzzles or verifiable delay functions and a realistic level of practicality is the "repeated squaring" function in RSA groups, defined via $x \mapsto x^{2^T} \bmod N$, underlying the time-lock puzzle of Rivest et al. [RSW96].[2] This delay function was recently elegantly extended by Pietrzak [Pie19] and Wesolowski [Wes19] to additionally yield a verifiable delay function.

The sequentiality of this function, however, is provided directly by assumption. That is, the function is assumed to be a delay function, and there is currently no substantial evidence relating its sequentiality to any other, more standard, assumptions such as the RSA or factoring assumptions. This highly unsatisfying state of affairs raises the important challenge of obtaining a better understanding of the sequentiality of repeated squaring in RSA groups. Clearly, given the factorization of the RSA modulus N, it is possible to speed up the computation of the repeated squaring function by reducing 2^T modulo the order of the multiplicative group $\mathbb{Z}_N^*$. Thus, the hardness of factoring is essential for the sequentiality of repeated squaring in RSA groups, leading to the following ambitious question:

Is speeding-up repeated squaring equivalent to factoring?

More generally, and given that delay functions have become a basic primitive underlying a variety of evolving applications, this urges at exploring other candidate delay functions, and obtaining a rigorous understanding of the cryptographic assumptions underlying their sequentiality.

1.1 Our Contributions

We resolve the above-mentioned challenges within the generic-ring model relative to an RSA modulus, capturing all computations that ignore any specific property

[1] Although, asymptotically, for any concrete instantiation of the hash function, such verification can be based on succinct non-interactive arguments for NP languages [Kil92, Mic94, GW11], as suggested by Döttling et al. [DGM+19] and Boneh et al. [BBB+18].

[2] There are additional constructions of delay functions which enable extensions to time-lock puzzles and to verifiable delay functions, but these rely on computational hardness within algebraic structures that are less-explored from a cryptographic standpoint. These include the class groups of imaginary quadratic fields [BW88, BBB+18, Pie19, Wes19] and isogenies of supersingular elliptic curves [FMP+19, Sha19].

of the representation of ring elements. Our main result is a sharp threshold on the sequentiality of all generic-ring delay functions based on the hardness of factoring in the standard model.

Trade-offs between sequentiality and parallelism are still not sufficiently understood from the complexity-theoretic perspective for computations in the standard model, and the generic-ring model provides a framework in which the nature of computation is somewhat better understood on the one hand, and which captures a wide variety of practical constructions and attacks on the other hand. In particular, our results apply to the repeated squaring function, for which we obtain the following theorem:

Theorem 1.1 (informal). *Generically speeding-up repeated squaring is equivalent to standard-model factoring.*

That is, we prove that any generic-ring algorithm that has a non-negligible probability in computing the function $x \mapsto x^{2^T} \bmod N$ for a uniformly chosen input $x \leftarrow \mathbb{Z}_N^*$ using a preprocessing stage and any polynomial number of parallel processors, each of which performs less than T sequential ring operations, yields a polynomial-time standard-model factoring algorithm with polynomially-related success probability and running time.

Sharp sequentiality thresholds. More generally, as mentioned above, we prove sharp thresholds on the sequentiality of all generic-ring delay functions based on the hardness of factoring in the standard model. These include, in particular, all rational (partial) multivariate functions over the ring, as well as more expressive functions which may depend on the equality pattern among all intermediate values in the computation.

We prove that every generic-ring function is in fact a delay function, whose sequentiality depends on the notion *sequentiality depth*, which we put forward. For rather simple polynomials, this notion essentially coincides with the logarithm of their degree (thus leading to Theorem 1.1 for the case of repeated squaring). For general generic-ring functions, our notion of sequentiality depth can be viewed as approximating the minimal degree of a rational function that is equivalent modulo N to the given function. Even for rational functions, however, defining a notion of equivalence is quite subtle given that the ring we consider is not an integral domain.

Equipped with our notion of sequentiality depth for any generic-ring function, we show that it serves as a sharp threshold for the number of sequential ring operations required in order to evaluate the function on a uniformly-chosen input.

Theorem 1.2 (informal). *Let F be a generic-ring function of sequentiality depth d. Assuming the hardness of factoring in the standard model, it holds that:*

- *F can be generically evaluated on any input with d sequential rounds of ring operations issued by a polynomial number of parallel processors.*
- *F cannot be generically evaluated on a uniformly-chosen input with less than d sequential rounds of ring operations, even with a preprocessing stage and any polynomial number of parallel processors.*

Sharp pseudorandomness thresholds. Moreover, we prove that for generic attackers who preform two few sequential rounds of ring operations, generic-ring functions provide not only unpredictability but in fact pseudorandomness. We explore the pseudorandomness of delay functions not merely as a natural strengthening of sequentiality, but also given that various applications of delay functions may directly benefit from it (e.g., randomness beacons [BCG15, BGZ16, PW18, BBB$^+$18]).

Complementing our notion of sequentiality depth, we put forward the notion of *pseudorandomness depth* of a generic-ring function, which can be viewed as an indistinguishability-based variant of sequentiality depth. As above, for rather simple polynomials the notion of pseudorandomness depth essentially coincides with the logarithm of their degree (thus leading to a variant of Theorem 1.1 that considers the pseudorandomness of repeated squaring instead of its sequentiality). For general generic-ring functions, the pseudorandomness depth is always upper bounded by the sequentiality depth, and exploring the exact relation between these two notions is an interesting direction for future research.

We prove that the pseudorandomness depth of any generic-ring function serves as a sharp threshold for the number of sequential rounds of ring operations required in order to distinguish between a uniformly chosen ring element and the output of the function when evaluated on a uniformly-chosen input.

Theorem 1.3 (informal). *Let F be a k-variate generic-ring function of pseudorandomness depth d. Assuming the hardness of factoring in the standard model, it holds that:*

- *$F(x_1, \ldots, x_k)$ can be generically distinguished from a uniform ring element y, where $x_1, \ldots, x_k$ are uniformly chosen in the ring, with d sequential rounds of ring operations issued by a polynomial number of parallel processors,*
- *$F(x_1, \ldots, x_k)$ cannot be generically distinguished from a uniform ring element y, where $x_1, \ldots, x_k$ are uniformly chosen in the ring, with less than d sequential rounds of ring operations, even with a preprocessing stage and any polynomial number of parallel processors.*

1.2 Related Work

We prove our results within the generic-ring model introduced by Aggarwal and Maurer [AM09] and further studied by Jager and Schwenk [JS13], as part of the line of research on idealized models for capturing algebraic constructions and hardness assumptions (see [Nec94, Sho97, BL96, MW98, Mau05, JS08, JR10, FKL18] and the references therein). Within their model, which is more suitable for capturing RSA-based constructions compared to the generic-group model, Aggarwal and Maurer proved that any generic algorithm that is able to compute roots of random ring elements relative to an RSA modulus can be used to produce a standard-model factoring algorithm. That is, they showed that the hardness of the RSA problem in the generic-ring model is equivalent to the hardness of factoring in the standard model. Following-up on previous work on

the relationship between the RSA and factoring assumptions (e.g., [BV98, DK02, Bro05, LR06, JNT07]), this provided substantial evidence towards the security of RSA-based constructions, showing that under the factoring assumption they are not vulnerable to a wide variety of practical cryptanalytic attacks.

Our work is directly inspired by the work of Aggarwal and Maurer in relating the capabilities of generic attackers to the hardness of factoring. The key difference, however, both conceptually and technically is that, based on the hardness of factoring, Aggarwal and Maurer proved that certain functions are completely infeasible to compute in polynomial time, whereas we show a more fine-grained result: It is infeasible to *speed-up* functions that can be computed in polynomial time (even with a preprocessing stage and with any polynomial number of parallel processors).

Following-up on the work of Aggarwal and Maurer, Jager and Schwenk [JS13] proved that generically computing the Jacobi symbol of a random $\mathbb{Z}_N$ element is equivalent to factoring, although Jacobi symbols are easy to compute non-generically given the standard integer representation of $\mathbb{Z}_N$ elements. As pointed out by Jager and Schwenk, and as discussed above, lower bounds in the generic-ring model nevertheless capture a wide variety of practical constructions and cryptanalytic attacks.[3]

In an independent work, Katz, Loss and Xu [KLX20] proved that within a quantitative variant of the algebraic group model [FKL18], speeding-up repeated squaring in the group $\mathbb{QR}_N$ of quadratic residues modulo N is equivalent to factoring N, where N is a bi-prime integer. Our results differ from theirs in a few aspects. Firstly, our result holds for any function which may be defined in the generic-ring model, whereas they consider only the repeated squaring function; and we consider both unpredictability and pseudorandomness, whereas they consider only unpredictability. Secondly, the model in which Katz et al. prove their result is incomparable to the model in which we prove our results: On the one hand, in the algebraic group model the adversary may use the concrete representation of group elements (which is unavailable to the adversary in the generic-ring model); but on the other hand, the adversary's output must be explained by a sequence of *group operations* (in $\mathbb{QR}_N$ this translates to a sequence of multiplications modulo N), whereas the generic-ring model permits the two *ring operations* and their inverses (i.e., in addition to multiplication, it also allows for addition, subtraction and division modulo N). Finally, they consider the group $\mathbb{QR}_N$, whereas we consider its super-group $\mathbb{Z}_N^*$. From a technical standpoint, their proof inherently relies on the fact that $\mathbb{QR}_N$ is cyclic, which is not the case for $\mathbb{Z}_N^*$.

Various cryptographic notions that share a somewhat similar motivation with delay functions have been proposed over the years, such as the above-discussed

[3] Aggarwal and Maurer also point out that lower bounds in the generic-ring model remain interesting specifically for problems in which the adversary is required to output elements in the ring, which is the case for evaluation of delay functions and for computing roots in the ring, bit is not the case for computing the Jacobi symbol of ring elements.

notions of time-lock puzzles and verifiable delay functions (e.g., [RSW96, BGJ+16, BBB+18, BBF18, Pie19, Wes19, EFK+20, FMP+19]), as well as other candidate functions [DN92, LW15] and other notions such as sequential functions and proofs of sequential work (e.g., [MMV11, MMV13, CP18]). It is far beyond the scope of this work to provide an overview of these notions and constructions, and we refer the reader to the work of Boneh et al. [BBB+18] for an in-depth discussion of these notions and of the relations among them.

In the generic-*group* model, Rotem, Segev and Shahaf [RSS20] have recently ruled out the possibility of constructing delay functions in cyclic groups, where the group's order is known to the attacker. In the random-oracle model, Döttling, Garg, Malavolta and Vasudevan [DGM+19], and Mahmoody, Smith and Wu [MSW19] recently proved impossibility results for certain classes of *verifiable* delay functions. Our work is of a different flavor, as these works provide negative evidence for the existence of delay functions and verifiable delay functions, whereas our work provides positive evidence for the existence of generic-ring delay functions. Our work is also different from the work of Rotem et al. [RSS20] in that it considers the generic-ring model in order to capture the RSA group which is believed to be of an *unknown order* from a computational perspective; and from the works of Döttling et al. [DGM+19] and Mahmoody et al. [MSW19] both in terms of focusing on the seemingly weaker notion of delay functions (i.e., we do not require verifiability), and in terms of characterizing the sequentiality and pseudorandomness of all functions in the more structured and expressive generic-ring model, based on the hardness of factoring in the standard model.

1.3 Paper Organization

The remainder of the paper is organized as follows. First, in Section 2 we present the generic-ring model, and in Section 3 we describe our framework for generic-ring delay functions. In Section 4 we prove our sharp threshold on the sequentiality of straight-line delay functions. Due to space limitations, our extension of this threshold to arbitrary generic-ring delay functions, as well as our sharp threshold on the pseudorandomness of delay functions, are formally presented and proven in the full version of this paper.

2 The Generic-Ring Model

In this section we present the idealized model of computation that we consider in this work, slightly refining the generic-ring model introduced by Aggarwal and Maurer [AM09] as we discuss below (mainly for the purpose of a more detailed accounting of parallelism vs. sequentiality). Informally, a generic-ring algorithm which receives one or more ring elements as input is restricted to handling these elements only via the two ring binary operations and their inverses, and by checking equality between two ring elements.

More formally, we consider generic computations in a ring R. Concretely, following Aggarwal and Maurer, the ring R we consider is that of integers modulo

N, denoted $\mathbb{Z}_N$, for N which is the product of two primes and is generated by a modulus-generation algorithm $\mathsf{ModGen}(1^\lambda)$, where $\lambda \in \mathbb{N}$ is the security parameter. All generic-ring algorithms in this paper receive the modulus N as an explicit bit-string input. Any computation in this model is associated with a table $\mathbf{B}$, where each entry of this table stores an element of R, and we denote by V_i the ring element that is stored in the ith entry.

Generic-ring algorithms access this table via an oracle $\mathcal{O}$, providing black-box access to $\mathbf{B}$ as follows. A generic-ring algorithm A that takes d ring elements as input does not receive an explicit representation of these elements, but instead, has oracle access to the table $\mathbf{B}$, whose first d entries store the elements of R corresponding to the d ring elements that are included in $\mathcal{A}$'s input. That is, if the input of an algorithm A consists of d ring elements $x_1, \ldots, x_d$, then from A's point of view the input consists of "pointers" $\widehat{x_1}, \ldots, \widehat{x_d}$ to the ring elements $x_1, \ldots, x_d$ (these elements are stored in the table $\mathbf{B}$). Accordingly, when a generic-ring algorithm outputs a ring element $y \in R$, it actually outputs a pointer which we denote by $\widehat{y}$, pointing to an entry in $\mathbf{B}$ containing y.[4] The oracle $\mathcal{O}$ allows for two types of queries:

- **Ring-operation queries:** These queries enable computation of the binary ring operations and their inverses. On input $(i, j, \circ)$ for $i, j \in \mathbb{N}$ and $\circ \in \{+, -, \cdot, /\}$, the oracle checks that the ith and jth entries of the table $\mathbf{B}$ are not empty and are not $\perp$, and in case that $\circ$ is $/$ (i.e., the inverse of the multiplication operation), the oracle also checks that the result of V_j is invertible in the ring. If all checks pass, then the oracle computes $V_i \circ V_j$ and stores the result in the next available entry. Otherwise, it stores $\perp$ in the next available entry.
- **Equality queries:** On input $(i, j, =)$ for $i, j \in \mathbb{N}$, the oracle checks that the ith and jth entries in $\mathbf{B}$ are not empty and are not $\perp$, and then returns 1 if $V_i = V_j$ and 0 otherwise. If either the ith or the jth entries are empty or are $\perp$, the oracle ignores the query.

Straight-line functions. Looking ahead, we will first prove our results for the case in which the delay function is a straight-line program, which is a deterministic generic-ring algorithm that does not issue any equality queries. We refer to such delay functions as straight-line delay functions. Then, we will extend our result to arbitrary generic-ring functions that may issue both ring-operations queries and equality queries.

Parallel computation. In order to reason about delay functions in this model, we need to extend it in a way which accommodates parallel computation. A generic-ring algorithm with w parallel processors invokes the oracle $\mathcal{O}$ with ring-operation queries in "rounds", where in each round, at most w parallel ring-operation queries may be issued. We assume some order on the processors

[4] We assume that all generic-ring algorithms receive a pointer to the multiplicative identity 1 and a pointer to the additive identity 0 as their first two inputs (we capture this fact by always assuming that the first two entries of $\mathbf{B}$ are occupied by $1 \in R$ and $0 \in R$), and we will forgo noting this explicitly from this point on.

so that the results of the queries are also placed in the table $\mathbf{B}$ according to this order. We emphasize, however, that the action that the oracle takes in response to each of the queries in a certain round is with respect to the contents of the table $\mathbf{B}$ *before* this round; meaning, the elements passed as input in the query issued by a processor in some round cannot depend on the result of a query made by any other processor in the same round. We emphasize that parallelism will not play a role when it comes to equality queries, as we allow algorithm to issue all possible such queries and do not account for their sequentiality (i.e., we prove our thresholds for the number of ring-operation queries considering only the total number of equality queries in our lower bound, and without issuing any equality queries in our upper bound).

We are interested in three main efficiency measures when considering generic-ring algorithms: (1) The number of parallel processors; (2) the number of sequential rounds in which ring-operation queries are issued; and (3) the algorithm's internal computation, measured via its running time.

Interactive computations. We consider interactive computations in which multiple algorithms pass ring elements (as well as non-ring elements) as inputs to one another. This is naturally supported by the model as follows: When a generic-ring algorithm A outputs k ring elements (along with a potential bit-string σ), it outputs the indices of k (non-empty) entries in the table $\mathbf{B}$ (together with σ). When these outputs (or some of them) are passed on as inputs to a generic-ring algorithm C, the table $\mathbf{B}$ is re-initialized, and these values (and possibly additional group elements that C receives as input) are placed in the first entries of the table.

Polynomial interpretation. Every ring element computed by the oracle $\mathcal{O}$ in response to a ring-operation query made by a generic-ring algorithm, can be naturally identified with a pair of polynomials in the ring elements given as input to the algorithm. Formally, for a generic-ring algorithm which receives d ring elements as input (in addition to the multiplicative identity 1 and the additive identity 0), we identify the ith input element with the pair $(X_i, 1)$ where X_i is an indeterminate of the polynomials we will consider (the 1 and 0 elements are identified with the pairs $(1, 1)$ and $(0, 1)$, respectively). The rest of the polynomials are defined recursively: For a ring-operation query $(i, j, \circ)$, let $(P_i(\mathbf{X}), Q_i(\mathbf{X}))$ and $(P_j(\mathbf{X}), Q_j(\mathbf{X}))$ be the pairs of polynomials identified with V_i and with V_j, where $\mathbf{X} = (X_1, \ldots, X_d)$. We define the pair of polynomials identified with the result of the query as:

$$(P(\mathbf{X}), Q(\mathbf{X})) = \begin{cases} (P_i(\mathbf{X}) \cdot Q_j(\mathbf{X}) + P_j(\mathbf{X}) \cdot Q_i(\mathbf{X}), Q_i(\mathbf{X}) \cdot Q_j(\mathbf{X})) & \text{, if } \circ \text{ is } + \\ (P_i(\mathbf{X}) \cdot Q_j(\mathbf{X}) - P_j(\mathbf{X}) \cdot Q_i(\mathbf{X}), Q_i(\mathbf{X}) \cdot Q_j(\mathbf{X})) & \text{, if } \circ \text{ is } - \\ (P_i(\mathbf{X}) \cdot P_j(\mathbf{X}), Q_i(\mathbf{X}) \cdot Q_j(\mathbf{X})) & \text{, if } \circ \text{ is } \cdot \\ (P_i(\mathbf{X}) \cdot Q_j(\mathbf{X}), Q_i(\mathbf{X}) \cdot P_j(\mathbf{X})) & \text{, if } \circ \text{ is } / \end{cases}$$

Note that this definition extends to the interactive case, in which one generic-ring algorithm A receives d ring elements $x_1, \ldots, x_d$ as input, computes some ring element from them which is associated with the pair of polynomials $(P(\mathbf{X}), Q(\mathbf{X}))$, and then passes this ring element as input to another generic-ring algorithm C.

Then, this definition allows us to reason about the values computed by C as pairs of polynomials in the elements $x_1, \ldots, x_d$.

Each pair of polynomials is naturally interpreted as a rational function (by setting the first polynomial to be the numerator and the second to be the denominator). For the time being, however, we simply think of these polynomials as polynomials in $\mathbb{Z}[\mathbf{X}]$, and so the problem of division by 0 does not arise yet, since the second polynomial is always non-zero as a polynomial in $\mathbb{Z}[\mathbf{X}]$. For a straight-line program S, we will denote by $(P^S_\sigma[i,j], Q^S_\sigma[i,j])$ the pair of polynomials computed by the jth query of the ith processor of S, when invoked on explicit input $\sigma \in \{0,1\}^*$. For the special case of a straight-line program S that outputs a single ring element, the pair of polynomials corresponding to this element is fixed for every explicit input σ, and we denote it by $\left(P^S_\sigma, Q^S_\sigma\right)$. When we are working in the ring $\mathbb{Z}_N$, we will sometimes consider all of the aforementioned polynomials as polynomials in $\mathbb{Z}_N[\mathbf{X}]$ (instead of allowing arbitrary integer coefficients). This is naturally done by reducing all coefficients of the polynomial modulo N, and will be clear from context.

Passing ring elements explicitly. Throughout the paper we refer to values as either "explicit" ones or "inexplicit"/"implicit" ones. Explicit values are all values whose representation (e.g., binary strings of a certain length) is explicitly provided to the generic algorithms under consideration. Inexplicit values are all values that correspond to ring elements and that are stored in the table $\mathbf{B}$ – thus generic algorithms can access them only via oracle queries. We will sometimes interchange between providing ring elements as input to generic-ring algorithms inexplicitly, and providing them explicitly. Note that moving from the former to the latter is well defined, since a generic-ring algorithm A that receives some of its input ring elements explicitly can always simulate the computation as if they were received as part of the table $\mathbf{B}$. For a ring element x, we will differentiate between the case where x is provided explicitly and the case where it is provided implicitly via the table $\mathbf{B}$, using the notation x in the former case, and the notation $\widehat{x}$ in the latter.

In cases where all inputs to a generic-ring element A are provided explicitly, we may be interested in obtaining its outputs explicitly as well (note that this is indeed possible, since in this case the algorithm may preform all ring operations internally in an explicit manner). When this is the case, we will use the oracle notation A^R instead of $A^{\mathcal{O}}$, where R is the ring being considered. For example, consider a generic-ring algorithm A which receives two ring elements x_1 and x_2 as input, and outputs $x_1 + x_2$, and the is ring $\mathbb{Z}_{15}$ of integers modulo 15. The notation $A^{\mathbb{Z}_{15}}(7, 10)$ indicates that the output of A (i.e., the integer 2) is obtained explicitly as an integer.

Finally, note that if we replace a proper subset of the input ring-elements to a generic algorithm A with explicit integers, than the intermediate ring elements which A computes via the oracle $\mathcal{O}$ can be interpreted as pairs of polynomials in the remaining inexcplicit ring elements, as described above.

Comparison with the model of Aggarwal and Maurer. Our model slightly refines that of Aggarwal and Maurer [AM09] in the following natural respects:

- As mentioned above, we consider algorithms with possibly many parallel processors, whereas Aggarwal and Maurer consider algorithms which may invoke the oracle on a single query at a time. Considering multiple processors is essential when reasoning about delay function, as their security guarantees should hold even against parallel adversaries.
- Algorithms in our model may receive multiple ring elements as input, as opposed to a single ring element in the model of Aggarwal and Maurer. This allows us to reason about the sequentiality of computing arbitrary generic functions in the ring (e.g., multivariate rational functions).
- We consider interactive computations, which allows us to reason about security properties which are defined via an interactive security experiment. In particular, it allows us to account for a preprocessing stage when reasoning about delay functions.
- Algorithms in our model may receive an explicit bit-string input (in addition to the modulus N), which allows us to consider families of functions (via the delay parameter T passed to the function evaluation algorithm), and explicit states passed from one adversarial algorithm to another in interactive security experiments.

In addition to the above extensions, it should be noted that Aggarwal and Maurer present graph-based definitions for straight-line programs and generic-ring algorithms, which we forgo here. However, both our definitions and the ones in the work of Aggarwal and Maurer can be rendered as special cases of Maurer's generic model of computation [Mau05]; and when restricting our definitions to single-processor algorithms with one ring element input, they are equivalent to the ones found in the work of Aggarwal and Maurer.

The reason we choose to base our definitions on oracle-aided algorithms is that we find it more convenient to explicitly consider the running time of such algorithms in terms of their internal computational efforts. This comes up when analyzing the running time of our factorization algorithms (in the plain model) outputted by the reduction. Even when considering the simple case where the input to the reduction is a straight-line program; once this program receives an explicit input (e.g., the modulus N) in addition to ring elements, its queries may be (and are indeed expected to be) a function of this input. Since this function is not necessarily efficiently computable, reasoning about the running time of the underlying straight-line program is necessary.

3 Generic-Ring Delay Functions

A generic-ring delay function in the ring $\mathbb{Z}_N$ is given by a generic-ring algorithm DF. This is a deterministic generic-ring algorithm, which receives as input the modulus N, the delay parameter T and implicit access, as defined in Section 2, to k_{in} ring elements $x_1, \ldots, x_{k_{\mathsf{in}}}$, and outputs (implicitly) a single ring element.[5]

[5] For concreteness, we consider the case where the output consists of a single ring element, and note that all of our bounds easily extend to the case where the output consists of several ring elements and an explicit bit-string.

In this section, we define the security of generic-ring delay functions (see Section 3.1) and our notions of sequentially depth and pseudorandomness depth (see Section 3.2).

3.1 The Security of Generic-Ring Delay Functions

We consider two definitions that capture the fact that a generic-ring delay function needs to be "inherently sequential". The first requires that for a delay parameter T, no algorithm which makes less than T sequential rounds of ring-operation queries should be successful with non-negligible probability in evaluating a delay function on a randomly-chosen input – even with any polynomial number of parallel processors and with a preprocessing stage. This definition is an adaptation of the sequentiality definition for verifiable delay functions of Boneh et al. [BBB+18] to the generic-ring model.

Definition 3.1 (Sequentiality). *Let $k_{\mathsf{in}} = k_{\mathsf{in}}(\lambda), T = T(\lambda)$ and $w = w(\lambda)$ be functions of the security parameter $\lambda \in \mathbb{N}$. A generic-ring delay function DF is (T, w)-sequential if for every polynomial $q = q(\cdot, \cdot)$ and for every pair $A = (A_0, A_1)$ of generic-ring algorithms, where A_0 issues at most $q(\lambda, T)$ ring-operation queries and A_1 consists of at most $w(\lambda)$ parallel processors each of which issues at most T sequential rounds of ring-operation queries, there exists a negligible function $\nu(\cdot)$ such that*

$$\Pr\left[\mathsf{Exp}^{\mathsf{Seq}}_{\mathsf{DF},A}(\lambda) = 1\right] \leq \nu(\lambda)$$

for all sufficiently large $\lambda \in \mathbb{N}$, where the experiment $\mathsf{Exp}^{\mathsf{Seq}}_{\mathsf{DF},A}(\lambda)$ is defined as follows:

1. $N \leftarrow \mathsf{ModGen}(1^\lambda)$.
2. $\mathsf{st} \leftarrow A_0^{\mathcal{O}}(N, T)$.
3. $\widehat{y} := \mathsf{DF}^{\mathcal{O}}(N, T, \widehat{x_1}, \ldots, \widehat{x_{k_{\mathsf{in}}}})$, *where* $x_1, \ldots, x_{k_{\mathsf{in}}} \leftarrow \mathbb{Z}_N$.
4. $\widehat{y'} \leftarrow {A_1}^{\mathcal{O}}(N, \mathsf{st}, \widehat{x_1}, \ldots, \widehat{x_{k_{\mathsf{in}}}})$.
5. *Output* 1 *if* $y' = y$, *and otherwise output* 0.

Note that the state st passed from A_0 to A_1 in the definition of $\mathsf{Exp}^{\mathsf{Seq}}_{\mathsf{DF},A}(\lambda)$ may include both explicit bit-strings and implicit ring elements.

Our second definition is a seemingly stronger one, and it requires that the for a delay parameter T, no algorithm which makes less than T sequential rounds of ring-operation queries should be successful with non-negligible probability in distinguishing between the true output of the function and a uniformly chosen ring element – even with any polynomial number of parallel processors and with a preprocessing stage. Satisfying this definition is desirable not merely because it is stronger in principal, but also because applications of delay functions often do rely on the assumption that the output of the function is pseudorandom for any algorithm which runs in sequential time which is less than the delay parameter T. Such application include for example the use of verifiable delay-functions for constructing randomness beacons (see, for example [BCG15, BGZ16, PW18, BBB+18] and the references within).

Definition 3.2 (Pseudorandomness). *Let $k_{\mathsf{in}} = k_{\mathsf{in}}(\lambda), T = T(\lambda)$ and $w = w(\lambda)$ be functions of the security parameter $\lambda \in \mathbb{N}$. A generic-ring delay function DF whose input includes k_{in} ring elements is (T, w)-pseudorandom if for every polynomial $q = q(\cdot, \cdot)$ and for every pair $A = (A_0, A_1)$ of generic-ring algorithms, where A_0 issues at most $q(\lambda, T)$ ring-operation queries and A_1 consists of at most $w(\lambda)$ parallel processors each of which issues at most T sequential rounds of ring-operation queries, there exists a negligible function $\nu(\cdot)$ such that*

$$\mathsf{Adv}_{\mathsf{DF},A}(\lambda) \stackrel{\mathsf{def}}{=} \left|\Pr\left[\mathsf{Exp}^{\mathsf{SP}}_{\mathsf{DF},A,0}(\lambda) = 1\right] - \left[\mathsf{Exp}^{\mathsf{SP}}_{\mathsf{DF},A,1}(\lambda) = 1\right]\right| \leq \nu(\lambda)$$

for all sufficiently large $\lambda \in \mathbb{N}$, where for $b \in \{0,1\}$, the experiment $\mathsf{Exp}^{\mathsf{SP}}_{\mathsf{DF},A,0}(\lambda)$ is defined as:

1. $N \leftarrow \mathsf{ModGen}(1^\lambda)$.
2. $\mathsf{st} \leftarrow A_0^{\mathcal{O}}(N, T)$.
3. $y_0 \leftarrow \mathbb{Z}_N$.
4. $\widehat{y_1} := \mathsf{DF}^{\mathcal{O}}(N, T, \widehat{x_1}, \ldots, \widehat{x_{k_{\mathsf{in}}}})$, *where* $x_1, \ldots, x_{k_{\mathsf{in}}} \leftarrow \mathbb{Z}_N$.
5. $b' \leftarrow {A_1}^{\mathcal{O}}(N, \mathsf{st}, \widehat{x_1}, \ldots, \widehat{x_{k_{\mathsf{in}}}}, \widehat{y_b})$.
6. *Output* b'.

3.2 The Depth of Generic-Ring Delay Functions

Our bounds on the sequentiality and pseudorandomness of generic-ring delay functions depend on the notions of sequentiality depth and pseudorandomness depth that we now introduce. We will begin by defining these notions for straight-line functions, and then we will extend them to arbitrary generic-ring functions.

Straight-line delay functions. We begin, in Definition 3.3, by defining the sequentiality depth of a straight-line delay function. Informally, if a straight-line delay function DF has sequentiality depth at most d, it means that it is possible (with high probability and with a preprocessing stage) to compute a rational function which is equivalent modulo N to the rational function computed by DF using d or less sequential rounds of ring operations. By equivalence of rational functions, we mean that the numerator of the difference between the two functions is the zero polynomial modulo N.

We remind the reader that for a straight-line program S and explicit input σ, the pair (P^S_σ, Q^S_σ) describes the output of S on the explicit input σ, as a pair of polynomials (which we may think of as a rational function) in the ring elements given as input to S (see Section 2).

Definition 3.3. *Let $T = T(\lambda)$ and $d = d(\lambda)$ be functions of the security parameter $\lambda \in \mathbb{N}$, and let DF be a straight-line delay function. We say that DF has* sequentiality depth at least d *if for every pair $G = (G_0, G_1)$ of polynomial-time generic-ring algorithms, where G_1 is a straight-line program with polynomially-many parallel processors each of which issues at most $d-1$ sequential rounds of ring-operation queries, there exists a negligible function $\nu(\cdot)$ such that*

$$\Pr\left[\mathsf{Exp}^{\mathsf{SeqDepth}}_{\mathsf{DF},G}(\lambda) = 1\right] \leq \nu(\lambda),$$

for all sufficiently large $\lambda \in \mathbb{N}$*, where the experiment* $\mathsf{Exp}^{\mathsf{SeqDepth}}_{\mathsf{DF},G}(\lambda)$ *is defined as follows:*

1. $N \leftarrow \mathsf{ModGen}(1^\lambda)$.
2. $\mathsf{st} \leftarrow G_0^{\mathcal{O}}(N,T)$*, where* $\mathsf{st} = (\mathsf{st}_0, \widehat{\mathsf{st}_1}, \ldots, \widehat{\mathsf{st}_\ell})$.
3. *Output* 1 *if*

$$g_{\mathsf{num}}, g_{\mathsf{den}} \not\equiv 0 \ (mod\ N)$$

and

$$P^{\mathsf{DF}}_{N,T} \cdot g_{\mathsf{den}} - Q^{\mathsf{DF}}_{N,T} \cdot g_{\mathsf{num}} \equiv 0 \ (mod\ N),$$

where $(g_{\mathsf{num}}(\mathbf{X}), g_{\mathsf{den}}(\mathbf{X})) = (P^{G_1}_{N,T,\mathsf{st}_0}(\mathsf{st}_1, \ldots, \mathsf{st}_\ell, \mathbf{X}), Q^{G_1}_{N,T,\mathsf{st}_0}(\mathsf{st}_1, \ldots, \mathsf{st}_\ell, \mathbf{X}))$. *Otherwise, output* 0.

If the sequentiality depth of DF *is not at least* $d+1$*, we say that it is* at most d*. If the sequentiality depth of* DF *is at least* d *and at most* d*, we say that* DF has sequentiality depth d*.*

We clarify that g_{num} and g_{den} are polynomials only in the formal variables replacing the input elements to the function DF, and are obtained from G_1 by fixing its explicit input to be (N, T, st_0) and assigning the integer values $\mathsf{st}_1, \ldots, \mathsf{st}_\ell$ to the variables replacing the ring elements passed from G_0 to G_1 as part of the state. By the notation $P \equiv 0 \pmod N$ for a polynomial P, we mean that all of the coefficients of P are 0 modulo N. Note that a k-variate polynomial might have a value of 0 for all inputs in $(\mathbb{Z}_N)^k$, but still have non-zero coefficients modulo N.

Intuitively, Definition 3.3 captures the fact that if we multiply both the numerator and the denominator of the function computed by DF by the same polynomial p, then this does not change the number of sequential ring operations required to evaluate the function. For example, the function $f_{N,T}(X_1, X_2, X_3) = \left(X_1^{2^T} \cdot X_2\right) \Big/ \left(X_1^{2^T} \cdot X_3\right) \pmod N$ can be evaluated using a single ring operation, since it is equivalent to the function $g_{N,T}(X_1, X_2, X_3) = X_2/X_3 \pmod N$. On the other hand, the function $f_{N,T}(X) = X^{\varphi(N)} \pmod N$ (where φ is Euler's totient function) is not equivalent under our definition to the function $g_{N,T}(X) = 1$, even though the two functions agree almost everywhere in the ring.

Note that since we wish to relate Definition 3.3 to Definition 3.1, allowing for a preprocessing stage is paramount. Consider for example the function $f_{N,T}(X) = 2^T \cdot X \pmod N$. Without preprocessing, trivially evaluating this function requires $T+1$ ring operations. However, T of them are independent of the input, and may be moved to the preprocessing stage, leaving just a single ring operation to be computed in the online stage.

We now define, in Definition 3.3, the pseudorandomness depth of a straight-line delay function. The definition will use the notation $P^{G_1}_{N,T,\mathsf{st}_0}(\mathsf{st}_1, \ldots, \mathsf{st}_\ell, \mathbf{X}, P^{\mathsf{DF}}_{N,T}(\mathbf{X})/Q^{\mathsf{DF}}_{N,T}(\mathbf{X}))$ for a straight-line program G_1. This can be seen as the polynomial obtained by invoking G_1 on explicit input (N, T, st_0), ℓ *explicit* state ring elements, k_{in} input ring elements $\widehat{x_1}, \ldots, \widehat{x_{k_{\mathsf{in}}}}$, and an ring element $\widehat{y}$ which is the output of DF on $\widehat{x_1}, \ldots, \widehat{x_{k_{\mathsf{in}}}}$, and looking at the numerator of the output of G_1

as a polynomial in the variables $X_1, \ldots, X_{k_{\mathsf{in}}}$ replacing $\widehat{x_1}, \ldots, \widehat{x_{k_{\mathsf{in}}}}$ (as discussed in Section 2).

Definition 3.4. *Let $T = T(\lambda)$ and $d = d(\lambda)$ be functions of the security parameter $\lambda \in \mathbb{N}$, and let DF be a straight-line delay function. We say that DF has* pseudorandomness depth at least d *if for every pair $G = (G_0, G_1)$ of polynomial-time generic-ring algorithms, where G_1 is a straight-line program with polynomially-many parallel processors each of which issues at most $d-1$ rounds of ring-operation queries, there exists a negligible function $\nu(\cdot)$ such that*

$$\Pr\left[\mathsf{Exp}^{\mathsf{PRDepth}}_{\mathsf{DF},G}(\lambda) = 1\right] \leq \nu(\lambda),$$

for all sufficiently large $\lambda \in \mathbb{N}$, where the experiment $\mathsf{Exp}^{\mathsf{PRDepth}}_{\mathsf{DF},G}(\lambda)$ is defined as follows:

1. $N \leftarrow \mathsf{ModGen}(1^\lambda)$.
2. $\mathsf{st} \leftarrow G_0^{\mathcal{O}}(N, T)$, *where* $\mathsf{st} = (\mathsf{st}_0, \widehat{\mathsf{st}_1}, \ldots, \widehat{\mathsf{st}_\ell})$.
3. *Output* 1 *if*

$$P^{G_1}_{N,T,\mathsf{st}_0}(\mathsf{st}_1, \ldots, \mathsf{st}_\ell, \mathbf{X}, Y) \not\equiv 0 \ (mod\ N)$$

and

$$P^{G_1}_{N,T,\mathsf{st}_0}\left(\mathsf{st}_1, \ldots, \mathsf{st}_\ell, \mathbf{X}, \frac{P^{\mathsf{DF}}_{N,T}(\mathbf{X})}{Q^{\mathsf{DF}}_{N,T}(\mathbf{X})}\right) \equiv 0 \ (mod\ N)$$

Otherwise, output 0.

If the pseudorandomness depth of DF is not at least $d+1$, we say that it is at most *d. If the pseudorandomness depth of DF is at least d and at most d, we say that* DF has pseudorandomness depth *d.*

Informally, if the pseudorandomness depth of a straight-line delay function DF which takes in k_{in} ring elements is at most d, it means that it is possible (with high probability and with a preprocessing stage) to compute a $(k_{\mathsf{in}}+1)$-variate polynomial p which is not the zero polynomial, but becomes the zero (k_{in}-variate) polynomial when the last variable is replaced with the output of the rational function computed by DF (in the previous k_{in} input variables). Intuitively, this notion captures the following trivial attack: Given access to $\widehat{x_1}, \ldots, \widehat{x_{k_{\mathsf{in}}}}$ and $\widehat{y}$, evaluate p at these inputs and zero test the result. Note that if the sequentiality depth of DF is at most d, then so is its pseudorandomness depth.

Sequentiality and pseudorandomness depths vs. degree. The sequentiality and pseudorandomness depths of a straight-line delay function are inherently related to the degree the rational function it computes. For a rational function $f = f_{\mathsf{num}}/f_{\mathsf{den}}$, we let its degree be the difference (in absolute value) between the degrees of its numerator and denominator polynomials, where by degree of a multi-variate polynomial, we mean its *total* degree;[6] i.e.,

$$\mathsf{deg}(f) = \left|\mathsf{deg}(f_{\mathsf{num}}) - \mathsf{deg}(f_{\mathsf{den}})\right|.$$

[6] E.g., the degree of the polynomial X_1X_2 is 2.

Informally, the following claim establishes that the sequentiality and pseudorandomness depths of a straight-line delay function DF are lower bounded by the logarithm of the degree of the rational function it computes.

Before stating the claim (which is proved in the full version of the paper), we introduce the following notation. For a concrete modulus $N \in \mathbb{N}$ outputted by $\mathsf{ModGen}(1^\lambda)$, denote by $\mathsf{Exp}^{\mathsf{SeqDepth}}_{\mathsf{DF},G}(\lambda, N)$ and $\mathsf{Exp}^{\mathsf{PRDepth}}_{\mathsf{DF},G}(\lambda, N)$ the experiments obtained from $\mathsf{Exp}^{\mathsf{SeqDepth}}_{\mathsf{DF},G}(\lambda)$ and $\mathsf{Exp}^{\mathsf{PRDepth}}_{\mathsf{DF},G}(\lambda)$ by fixing the modulus to be N (instead of sampling it at the onset), respectively.

Claim 3.5 *Let $T = T(\lambda), d = d(\lambda)$ and $k_{\mathsf{in}} = k_{\mathsf{in}}(\lambda)$ be functions of the security parameter $\lambda \in \mathbb{N}$, let N be an integer outputted by $\mathsf{ModGen}(1^\lambda)$ and let DF be a straight-line delay function. Let $f_{\mathsf{num}}, f_{\mathsf{den}} \in \mathbb{Z}_N[X_1, \ldots, X_k]$ such that $f = f_{\mathsf{num}}/f_{\mathsf{den}}$ is the rational function computed by DF on explicit input (N, T). If all coefficients of f_{num} and of f_{den} are coprime to N, then for every pair $G = (G_0, G_1)$ of polynomial-time generic-ring algorithms, where G_1 is a straight-line program with polynomially-many parallel processors each of which issues at most d sequential rounds of ring-operation queries, it holds that*

1. *If* $\Pr\left[\mathsf{Exp}^{\mathsf{SeqDepth}}_{\mathsf{DF},G}(\lambda, N) = 1\right] > 0$ *then* $d(\lambda) \geq \log\left(\mathsf{deg}(f)\right)$.
2. *If* $\Pr\left[\mathsf{Exp}^{\mathsf{PRDepth}}_{\mathsf{DF},G}(\lambda, N) = 1\right] > 0$ *then* $d(\lambda) \geq \log\left(\mathsf{deg}(f)\right)$.

The depth of repeated squaring. As discussed in Section 1.1, for rather simple polynomials our notions of sequentiality depth and pseudorandomness depth essentially coincides with the logarithm of their degree. This is the case with the repeated squaring function of Rivest, Shamir and Wagner [RSW96], where both notions exactly coincide with the logarithm of its degree. Specifically, consider the repeated squaring function: For a modulus N and a delay parameter $T = T(\lambda)$, the function is defined by $f^{\mathsf{RSW}}_{N,T}(X) = X^{2^T} \pmod N$. Of course, $f^{\mathsf{RSW}}_{N,T}$ may be evaluated using T ring operations, so its sequentiality depth is at most T. Claim 3.5 shows that it is exactly T, since any function computed with less than T ring operations will not be equivalent (as specified by Definition 3.3) to $f^{\mathsf{RSW}}_{N,T}$. Moreover, Claim 3.5 shows that the pseudorandomness depth of repeated squaring is also exactly T.

Arbitrary generic-ring delay functions. We now extend the above notions to arbitrary generic-ring delay functions. Informally, in case that a delay function DF issues equality queries, we consider the straight-line program obtained from DF by setting the responses to all equality queries to be negative, except those which are trivially satisfied. As formally defined below, by a trivially satisfied equality query we mean that the polynomial it induces is the all-zero polynomial modulo N.

More formally, for a generic-ring delay function DF, we denote by $\mathsf{SLP}(\mathsf{DF})$ the straight-line program obtained from DF by setting the responses to all non-trivial equality queries to be negative (and to all trivial queries to be positive). This may be done one query at a time: At each step, consider the first of the equality queries remaining (recall that DF is deterministic), and let (P, Q) and (P', Q') be the pairs

of polynomials associated with it. If $P \cdot Q' - P' \cdot Q \equiv 0 \pmod{N}$, then assume (without querying) that the answer is answered affirmatively, and otherwise assume that it is answered negatively (if any of P, Q, P' and Q' is $\perp$, then treat the query as ignored). Note that this transformation is not necessarily efficient, but it need not be, since it is only used to *define* the notions of sequentiality depth and pseudorandomness depth for arbitrary generic-ring delay functions. Equipped with this notation, Definition 3.6 captures the above informal description.

Definition 3.6. *Let $T = T(\lambda)$, $d_{\mathsf{Seq}} = d_{\mathsf{Seq}}(\lambda)$ and $d_{\mathsf{PR}} = d_{\mathsf{PR}}(\lambda)$ be functions of the security parameter $\lambda \in \mathbb{N}$, and let DF be a generic-ring delay function. We say that DF has sequentiality depth at least (resp. at most) d_{Seq} if $\mathsf{SLP}(\mathsf{DF})$ has sequentiality depth at least (resp. at most) d_{Seq}. We say that DF has pseudorandomness depth at least (resp. at most) d_{PR} if $\mathsf{SLP}(\mathsf{DF})$ has sequentiality depth at least (resp. at most) d_{PR}.*

4 A Sharp Sequentiality Threshold for Straight-Line Delay Functions

In this section we present our sharp threshold for the number of sequential rounds of ring-operation queries that are required for evaluating straight-line delay functions (i.e., rational functions in their input elements). Our lower bound is proven in Section 4.1, and its matching upper bound is proven in Section 4.2.

4.1 From Speeding Up Straight-Line Delay Functions to Factoring

Let DF be a straight-line delay function that has sequentiality depth at least d, for some function $d = d(\lambda)$ of the security parameter (recall Definition 3.3). We prove the following theorem, showing that any generic-ring algorithm that computes DF on a uniform input with a non-negligible probability in less than d sequential rounds of ring-operation queries, can be transformed into a factoring algorithm in the standard model.

Theorem 4.1. *Let $T = T(\lambda)$ and $k_{\mathsf{in}} = k_{\mathsf{in}}(\lambda)$ be functions of the security parameter $\lambda \in \mathbb{N}$, and let DF be a straight-line program delay function receiving k_{in} ring elements as input. Then, for every function $\epsilon = \epsilon(\lambda)$, for every polynomial $p(\cdot)$, and for every pair $G = (G_0, G_1)$ of probabilistic polynomial-time generic-ring algorithms such that G_1 has polynomially many parallel processors each of which issues at most $q_{\mathsf{op}} = q_{\mathsf{op}}(\lambda)$ sequential rounds of ring-operation queries and the sequentiality depth of DF is at least $q_{\mathsf{op}} + 1$, there exists an algorithm A running in time $\mathsf{poly}(\lambda, \log(1/\epsilon))$ for which the following holds: For all sufficiently large $\lambda \in \mathbb{N}$, if*

$$\Pr\left[\mathsf{Exp}^{\mathsf{Seq}}_{\mathsf{DF},G}(\lambda) = 1\right] \geq \frac{1}{p(\lambda)}$$

then

$$\Pr_{\substack{N \leftarrow \mathsf{ModGen}(1^\lambda) \\ (a,b) \leftarrow A(N,T)}} [(N = a \cdot b) \wedge (a, b \in [N-1])] > 1 - \epsilon(\lambda).$$

The proof of Theorem 4.1 makes use of Lemma 4.2 stated below. We first introduce some notation: For an integer $N \in \mathbb{N}$ and for a k-variate polynomial P, we denote by $\alpha_N(P)$ the density of roots of P in $\mathbb{Z}_N$; i.e.,

$$\alpha_N(P) = \Pr_{x_1,\ldots,x_k \leftarrow \mathbb{Z}_N} \left[P(x_1,\ldots,x_k) = 0 \pmod{N} \right].$$

Roughly speaking, Lemma 4.2 states that given any straight-line program whose output is a polynomial P in its input elements, we can construct a standard-model algorithm which succeeds in factoring N with probability which is proportional to $\alpha_N(P)$. Recall that for a straight-line program S, the pair $(P^S_{N,\sigma}, Q^S_{N,\sigma})$ denotes the output of S on explicit input (N,σ), as a pair of polynomials in the input ring elements to S (see Section 2).

Lemma 4.2. *Let $k = k(\lambda), t = t(\lambda), w = w(\lambda), \ell = \ell(\lambda)$ and $q = q(\lambda)$ be functions of the security parameter $\lambda \in \mathbb{N}$. For any generic-ring straight-line program S which takes as input k ring elements, a modulus N and an additional explicit ℓ-bit string, and runs in time t with w parallel processors, while making at most q sequential rounds of ring-operation queries, there exists an algorithm A_S which runs in time $O\left(t + \lambda^5 \cdot k^3 + w^3 \cdot q^3\right)$, such that the following holds: For every $\lambda \in \mathbb{N}$, for every N which is outputted with positive probability by $\mathsf{ModGen}(1^\lambda)$ and for every bit-string $\sigma \in \{0,1\}^\ell$ which S may receive as an additional explicit input, if $P^S_{N,\sigma} \not\equiv 0 \pmod{N}$ then*

$$\Pr_{(a,b) \leftarrow A_S(N,\sigma)} \left[\begin{matrix} N = a \cdot b \\ a, b \in [N-1] \end{matrix} \right] \geq \frac{\alpha_N(P^S_{N,\sigma}) - (k-1) \cdot 2^{-\lambda+1}}{\left(1 - 2^{-\lambda}\right)^{k-1} \cdot 8k \cdot (2\lambda \cdot k + w \cdot q)}.$$

We first prove Theorem 4.1 assuming Lemma 4.2 and then turn to prove Lemma 4.2. We start by giving a high-level overview of the proof, which ignores many of the technical difficulties arising in the formal analysis. Given $G = (G_0, G_1)$, our factoring algorithm A operates in three stages. In the first stage, it invokes G_0 in order to sample a state st,[7] samples random coins ρ for G_1, and initializes a data structure η which will be used in order to keep track of the likely response pattern to G_1's equality queries. The second stage proceeds in iterations – one per each equality query made by G_1. Each such equality query naturally induces a polynomial when fixing st, ρ and the responses to all previous equality queries according to the information in η. In the ith iteration, A tries to factor N using the factoring algorithm guaranteed by Lemma 4.2 for the polynomial induced by the ith equality query. If unsuccessful, A updates η with the likely response to the ith query, by checking it on a uniformly sampled input. In the third stage, A considers the polynomial induced by an equality between the output of DF and the output of G_1 when it is ran on the state st and with random coins ρ, and when the responses to its equality queries are in accordance with the learned η. Our algorithm then tries to factor N using the factoring

[7] For the sake of this high-level overview, assume that st does not include any implicit ring elements. In the full proof, this assumption is lifted by noting that since A is the one that runs G_0 it has explicit knowledge of the integer values of these elements.

algorithm guaranteed by Lemma 4.2 for the straight-line program computing this induced polynomial.

The analysis considers two cases. In the first case, there exists an equality query which is "balanced" in the sense that it is affirmatively answered with probability which is sufficiently bounded away from both 0 and 1, conditioned on all previous queries being answered with the more likely response. When this is the case, we show that in the iteration which corresponds to the first such query in the second stage of A, it succeeds in factoring N with high probability since: (1) with high probability the information in η indeed reflects the likely responses to previous queries; and (2) the polynomial induced by the this query is non-trivial and has a high rate of roots. In the second case, all equality queries of G_1 are sufficiently non-balanced so that the success probability of G is not reduced by too much when conditioning on all of these queries being answered in the more likely manner. If this is the case, then whenever the information in η is consistent with the likely responses (which happens with high enough probability), the rate of roots of the polynomial considered in the third stage of A is proportional to the success probability of G. We use the fact that G_1 makes less ring-operation queries than the sequentiality depth of DF to argue that this polynomial is non-trivial. We proceed to the formal proof.

Proof of Theorem 4.1. Let $G = (G_0, G_1)$ be a pair of generic-ring algorithms as in the statement of Theorem 4.1. Let $q_{\mathsf{eq}} = q_{\mathsf{eq}}(\lambda)$ and $w = w(\lambda)$ denote the bound on the number of equality queries made by G_1, and the number of parallel processors of G_1, respectively, and let $r = r(\lambda)$ be a bound on the number of random coins used by G_1. For a modulus N, a state st outputted by $G_0^{\mathbb{Z}_N}(N, T)$, an index $i \in [q_{\mathsf{eq}}]$, random coins $\rho \in \{0,1\}^r$ and a binary string η of length at most q_{eq} bits, we define a related polynomial $f[G_1, N, \mathsf{st}, i, \rho, \eta]$. This is the polynomial obtained from G_1 by running it on explicit input (N, T, st) and randomness ρ up to (and not including) the ith equality query, while setting the reply to each of the first $i-1$ equality queries of G_1 according to η: The reply to the jth equality query is positive if and only if the jth bit of η is 1. Let (P, Q) and (P', Q') be the pairs of polynomials corresponding to the two ring elements compared in the jth equality query in this computation. Then, we define $f[G_1, N, \mathsf{st}, i, \rho, \eta] = P \cdot Q' - P' \cdot Q$. Finally, for $\rho \in \{0,1\}^r$ and $\eta \in \{0,1\}^{q_{\mathsf{eq}}}$, denote by $\mathsf{SLP}(G_1)$ the straight-line program obtained from G_1 in the following manner: On explicit input $(N, T, \mathsf{st}, \rho, \eta)$, the program $\mathsf{SLP}(G_1)$ runs G_1 on explicit input (N, T, st) and randomness ρ, while setting the responses to all equality queries according to the bits of η. Consider the following standard-model factoring algorithm A_G:

Algorithm A_G

Input: An integer N sampled by $\mathsf{ModGen}(1^\lambda)$, and a delay parameter $T \in \mathbb{N}$.

1. Sample $\mathsf{st} \leftarrow G_0^{\mathbb{Z}_N}(N, T)$, and $\rho \leftarrow \{0,1\}^r$.
2. Initialize η_0 to be the empty string.
3. For $i = 1, \ldots, q_{\mathsf{eq}}$:

(a) Let $f_i^{(N,\mathsf{st},\rho,\eta_{i-1})}(\mathbf{X}) = f[G_1, N, \mathsf{st}, i, \rho, \eta_{i-1}](\mathbf{X})$, let S_i be the straight-line program that on explicit input $(N, T, \mathsf{st}, \rho, \eta_{i-1})$ computes the pair $\left(f_i^{(N,\mathsf{st},\rho,\eta_{i-1})}(\mathbf{X}), 1\right)$, and let A_{S_i} be the corresponding factorization algorithm guaranteed by Lemma 4.2. Run $A_{S_i}(N, T, \mathsf{st}, \rho, \eta_{i-1})$ to obtain (a_i, b_i). If $a_i, b_i \in [N-1]$ and $a_i \cdot b_i = N$, output (a_i, b_i) and terminate.
(b) Sample $x_1, \ldots, x_{k_{\mathsf{in}}} \leftarrow \mathbb{Z}_N$.
(c) If $f_i^{(N,\mathsf{st},\rho,\eta_{i-1})}(x_1, \ldots, x_{k_{\mathsf{in}}}) = 0$, set $\eta_i := \eta_{i-1} \| 1$ and otherwise, set $\eta_i := \eta_{i-1} \| 0$.

4. Let $\eta = \eta_{q_{\mathsf{eq}}}$, let $S = \mathsf{SLP}(G_1)$ and let $f_{\mathsf{out}}^{(N,\mathsf{st},\rho,\eta)} = P_{N,T,\mathsf{st},\rho,\eta}^{S} \cdot Q_{N,T}^{\mathsf{DF}} - Q_{N,T,\mathsf{st},\rho,\eta}^{S} \cdot P_{N,T}^{\mathsf{DF}}$. Let S_{out} be the straight-line program that on explicit input $(N, T, \mathsf{st}, \rho, \eta)$ computes $\left(f_{\mathsf{out}}^{(N,\mathsf{st},\rho,\eta)}(\mathbf{X}), 1\right)$ and let $A_{S_{\mathsf{out}}}$ be the corresponding factorization algorithm guaranteed by Lemma 4.2. Run $A_{S_{\mathsf{out}}}(N, T, \mathsf{st}, \rho, \eta)$ to obtain (a, b). If $a, b \in [N-1]$ and $a \cdot b = N$, output (a, b). Otherwise, output $\perp$.

Denote $\Pr\left[\mathsf{Exp}_{\mathsf{DF},G}^{\mathsf{Seq}}(\lambda) = 1\right]$ by β. We show that the probability that A_G outputs a valid factorization of N is at least $\Omega\left(\beta^2/\mathsf{poly}(\lambda)\right)$. Then, repeating the attack described by A_G for $\Omega\left(\ln(1/\epsilon) \cdot \beta^{-2} \cdot \mathsf{poly}(\lambda)\right)$ iterations yields Theorem 4.1.

For a modulus N, an index $i \in [q_{\mathsf{eq}}]$, a state st passed by G_0, randomness $\rho \in \{0,1\}^r$ and a string $\eta \in \{0,1\}^{i-1}$, we say that the polynomial $f_i^{(N,\mathsf{st},\rho,\eta)}$ is *heavy* if $\Pr_{\mathbf{x}}\left[f_i^{(N,\mathsf{st},\rho,\eta)}(\mathbf{x}) = 0\right] \geq 1 - \beta/(4 \cdot q_{\mathsf{eq}})$, and we say that it is *light* if $\Pr_{\mathbf{x}}\left[f_i^{(N,\mathsf{st},\rho,\eta)}(\mathbf{x}) = 0\right] \leq \beta/(4 \cdot q_{\mathsf{eq}})$. Otherwise, we say that it is *balanced*. For the same parameters, we also define an i-character string $\eta^*_{N,\mathsf{st},\rho,i} \in \{0,1,\perp\}^i$ recursively; we let $\eta^*_{N,\mathsf{st},\rho,0}$ be the empty string, and for $i \in [q_{\mathsf{eq}}]$ we define:

$$\eta^*_{N,\mathsf{st},\rho,i} = \begin{cases} \eta^*_{N,\mathsf{st},\rho,i-1} \| \perp, & \text{if } \perp \in \eta^*_{N,\mathsf{st},\rho,i-1} \text{ or } f_i^{(N,\mathsf{st},\rho,\eta^*_{N,\mathsf{st},\rho,i-1})} \text{ is balanced} \\ \eta^*_{N,\mathsf{st},\rho,i-1} \| 0, & \text{if } \perp \notin \eta^*_{N,\mathsf{st},\rho,i-1} \text{ and } f_i^{(N,\mathsf{st},\rho,\eta^*_{N,\mathsf{st},\rho,i-1})} \text{ is light} \\ \eta^*_{N,\mathsf{st},\rho,i-1} \| 1, & \text{if } \perp \notin \eta^*_{N,\mathsf{st},\rho,i-1} \text{ and } f_i^{(N,\mathsf{st},\rho,\eta^*_{N,\mathsf{st},\rho,i-1})} \text{ is heavy} \end{cases}$$

Denote $\eta^*_{N,\mathsf{st},\rho} = \eta^*_{N,\mathsf{st},\rho,i_{\mathsf{eq}}}$, and denote by Bal the event in which $\eta^*_{N,\mathsf{st},\rho}$ contains a $\perp$ symbol. We will prove the bound on A_G's success probability separately for the following two cases.

Case 1: $\Pr[\mathsf{Bal}] \geq \beta/2$. Let Factor be the event in which $A_G(N,T)$ successfully outputs a factorization of N. By total probability, it holds that

$$\Pr[\mathsf{Factor}] \geq \Pr[\mathsf{Factor}|\mathsf{Bal}] \cdot \Pr[\mathsf{Bal}] \geq \frac{\beta}{2} \cdot \Pr[\mathsf{Factor}|\mathsf{Bal}], \tag{1}$$

and so we wish to bound $\Pr[\mathsf{Factor}|\mathsf{Bal}]$. For $i \in [q_{\mathsf{eq}}]$, let E_i and E_i^* be the random variables corresponding to η_i in the execution of A_G and to $\eta^*_{N,\mathsf{st},\rho,i}$ described above. Let i^* be the minimal index in which $\eta^*_{N,\mathsf{st},\rho}$ has a $\perp$ symbol (if there are no $\perp$ symbols, then $i^* = 0$; note that i^* is also a random variable), and

let Typ be the event in which $E_{i^*-1} = E^*_{i^*-1}$. Then,

$$\begin{aligned}\Pr\left[\mathsf{Factor}|\mathsf{Bal}\right] &\geq \Pr\left[\mathsf{Factor}|\mathsf{Bal}\wedge\mathsf{Typ}\right]\cdot\Pr\left[\mathsf{Typ}|\mathsf{Bal}\right] \\ &\geq \Pr\left[\mathsf{Factor}|\mathsf{Bal}\wedge\mathsf{Typ}\right]\cdot\left(1-\frac{\beta}{4}\right) \qquad (2)\\ &\geq \frac{3}{4}\cdot\Pr\left[\mathsf{Factor}|\mathsf{Bal}\wedge\mathsf{Typ}\right] \qquad (3)\end{aligned}$$

where (2) follow by union bound on all indices up to i^* and the fact that it is always the case that $i^* \leq q_{\mathsf{eq}}$.

Let $\mathsf{Factor}(i^*)$ denote the event in which $A_{S_{i^*}}(N, T, \mathsf{st}, \rho, \eta_{i^*-1})$ successfully outputs a factorization of N. It holds that

$$\Pr\left[\mathsf{Factor}|\mathsf{Bal}\wedge\mathsf{Typ}\right] \geq \Pr\left[\mathsf{Factor}(i^*)|\mathsf{Bal}\wedge\mathsf{Typ}\right]. \qquad (4)$$

To complete the analysis of Case 1, we wish to bound $\Pr\left[\mathsf{Factor}(i^*)|\mathsf{Bal}\wedge\mathsf{Typ}\right]$, and to this end, we would like to invoke Lemma 4.2. In order to so, we need to argue two things: (1) That the (first) polynomial outputted by S_{i^*} – meaning, the polynomial $f_{i^*}^{(N,\mathsf{st},\rho,\eta_{i^*-1})}$ – is non-trivial modulo N; and (2) That this polynomial has many roots modulo N in $\mathbb{Z}_N$. This is indeed the case, since assuming both Bal and Typ occur, it holds that $E_{i^*-1} = E^*_{i^*-1}$ and hence the polynomial $f_{i^*}^{(N,\mathsf{st},\rho,\eta_{i^*-1})}$ is equal to the polynomial $f_{i^*}^{(N,\mathsf{st},\rho,\eta^*_{N,\mathsf{st},\rho,i^*-1})}$. But since the i^*th bit of $\eta^*_{\mathsf{st},\rho}$ is $\perp$, it means that $\alpha_N\left(f_{i^*}^{(N,\mathsf{st},\rho,\eta^*_{N,\mathsf{st},\rho,i^*-1})}\right) > \beta/(4\cdot q_{\mathsf{eq}})$. Hence, by Lemma 4.2,

$$\begin{aligned}\Pr\left[\mathsf{Factor}(i^*)|\mathsf{Bal}\wedge\mathsf{Typ}\right] &\geq \frac{\alpha_N\left(f_{i^*}^{(N,\mathsf{st},\rho,\eta^*_{N,\mathsf{st},\rho,i^*-1})}\right)-(k_{\mathsf{in}}-1)\cdot 2^{-\lambda+1}}{8\cdot\left(1-2^{-\lambda}\right)^{k_{\mathsf{in}}-1}\cdot k_{\mathsf{in}}\cdot(2\lambda\cdot k_{\mathsf{in}}+w\cdot q_{\mathsf{op}})} \\ &\geq \frac{\beta-(k_{\mathsf{in}}-1)\cdot 2^{-\lambda+3}\cdot q_{\mathsf{eq}}}{32\cdot q_{\mathsf{eq}}\cdot\left(1-2^{-\lambda}\right)^{k_{\mathsf{in}}-1}\cdot k_{\mathsf{in}}\cdot(2\lambda\cdot k_{\mathsf{in}}+w\cdot q_{\mathsf{op}})}. \qquad (5)\end{aligned}$$

Combining inequalities (1), (3) and (5) concludes the analysis of Case 1.

Case 2: $\Pr\left[\mathsf{Bal}\right] < \beta/2$. In this case, it holds that

$$\begin{aligned}\Pr\left[\mathsf{Exp}^{\mathsf{Seq}}_{\mathsf{DF},G}(\lambda)=1\middle|\overline{\mathsf{Bal}}\right] &\geq \Pr\left[\left(\mathsf{Exp}^{\mathsf{Seq}}_{\mathsf{DF},G}(\lambda)=1\right)\wedge\overline{\mathsf{Bal}}\right] \\ &= \beta-\Pr\left[\left(\mathsf{Exp}^{\mathsf{Seq}}_{\mathsf{DF},G}(\lambda)=1\right)\wedge\mathsf{Bal}\right] \\ &\geq \beta-\Pr\left[\mathsf{Bal}\right] \\ &> \frac{\beta}{2}. \qquad (6)\end{aligned}$$

Let AllTyp be the event in which $\eta = \eta^*_{N,\mathsf{st},\rho}$, and let $\mathsf{FactorOut}$ denote the event in which $A_{S_{\mathsf{out}}}(N, T, \mathsf{st}, \rho, \eta)$ successfully outputs a factorization of N. Then,

$$\begin{aligned}
\Pr[\mathsf{Factor}] &\geq \Pr[\mathsf{FactorOut}] \\
&\geq \Pr\left[\mathsf{FactorOut} \middle| \overline{\mathsf{Bal}} \wedge \mathsf{AllTyp}\right] \cdot \Pr\left[\overline{\mathsf{Bal}}\right] \cdot \Pr\left[\mathsf{AllTyp} \middle| \overline{\mathsf{Bal}}\right] \\
&> \Pr\left[\mathsf{FactorOut} \middle| \overline{\mathsf{Bal}} \wedge \mathsf{AllTyp}\right] \cdot \left(1 - \frac{\beta}{2}\right) \cdot \left(1 - \frac{\beta}{4}\right) \qquad (7) \\
&\geq \frac{3}{8} \cdot \Pr\left[\mathsf{FactorOut} \middle| \overline{\mathsf{Bal}} \wedge \mathsf{AllTyp}\right], \qquad (8)
\end{aligned}$$

where (7) follows from union bound over $i \in [q_{\mathsf{eq}}]$.

We again wish to invoke Lemma 4.2, so we wish to argue that conditioned on $\overline{\mathsf{Bal}} \wedge \mathsf{AllTyp}$, the polynomial $f^{(N,\mathsf{st},\rho,\eta)}_{\mathsf{out}} = P^S_{N,T,\mathsf{st},\rho,\eta} \cdot Q^{\mathsf{DF}}_{N,T} - Q^S_{N,T,\mathsf{st},\rho,\eta} \cdot P^{\mathsf{DF}}_{N,T}$ which the straight-line program S (from Step 4 of the algorithm A_G) computes is non-trivial modulo N with overwhelming probability. Assume that the contrary is true; i.e., that $f^{(N,\mathsf{st},\rho,\eta)}_{\mathsf{out}} \equiv 0 \pmod N$ with non-negligible probability conditioned on $\overline{\mathsf{Bal}} \wedge \mathsf{AllTyp}$. But, conditioned on $\overline{\mathsf{Bal}} \wedge \mathsf{AllTyp}$, it holds that $\eta = \eta^*_{N,\mathsf{st},\rho}$, and the responses pattern induced by $\eta^*_{N,\mathsf{st},\rho}$ to G_1's equality queries occurs with probability at least $3/4$ in $\mathsf{Exp}^{\mathsf{Seq}}_{\mathsf{DF},G}(\lambda)$. This means that the straight-line program S corresponds to a valid execution of G_1, and thus makes at most q_{op} operation queries, when given as input the ring elements in st implicitly (as elements in the oracle table $\mathbf{B}$). Consider the pair of algorithms (B_0, S), where B_0 computes st, ρ and η as in the definition of A_G and passes them to S, where the ring elements in st are passed implicitly. By Definition 3.3, this is a contradiction to the fact that DF has sequentiality depth at least $q_{\mathsf{op}} + 1$. Hence, for all sufficiently large $\lambda \in \mathbb{N}$, the polynomial $f^{(N,\mathsf{st},\rho,\eta)}_{\mathsf{out}}$ is non-trivial modulo N with all but negligible probability, and there exists an negligible function $\nu(\cdot)$ such that by Lemma 4.2,

$$\begin{aligned}
&\Pr\left[\mathsf{factor}(N, \mathsf{out}) \middle| \overline{\mathsf{Bal}} \wedge \mathsf{AllTyp}\right] \\
&\qquad \geq \frac{\alpha_N\left(f^{(N,\mathsf{st},\rho,\eta)}_{\mathsf{out}}\right) - (k_{\mathsf{in}} - 1) \cdot 2^{-\lambda+1} - \nu(\lambda)}{8 \cdot (1 - 2^{-\lambda})^{k_{\mathsf{in}}-1} \cdot k_{\mathsf{in}} \cdot (2\lambda \cdot k_{\mathsf{in}} + w \cdot q_{\mathsf{op}})} \\
&\qquad = \frac{\alpha_N\left(f^{(N,\mathsf{st},\rho,\eta^*_{N,\mathsf{st},\rho})}_{\mathsf{out}}\right) - (k_{\mathsf{in}} - 1) \cdot 2^{-\lambda+1} - \nu(\lambda)}{8 \cdot (1 - 2^{-\lambda})^{k_{\mathsf{in}}-1} \cdot k_{\mathsf{in}} \cdot (2\lambda \cdot k_{\mathsf{in}} + w \cdot q_{\mathsf{op}})} \qquad (9)
\end{aligned}$$

We are left with bounding $\alpha_N\left(f^{(N,\mathsf{st},\rho,\eta^*_{N,\mathsf{st},\rho})}_{\mathsf{out}}\right)$ for N, st and ρ for which $\overline{\mathsf{Bal}}$ holds. Consider the experiment $\mathsf{Exp}^{\mathsf{Seq}}_{\mathsf{DF},G}(\lambda)$, and let Con be the event in which the all equality queries made by G_1 in this experiment are answered consistently with $\eta^*_{N,\mathsf{st},\rho}$. Conditioned on $\overline{\mathsf{Bal}}$ and on Con, the output of G_1 is exactly $\left(P^S_{N,\mathsf{st},\rho,\eta^*_{N,\mathsf{st},\rho}}, Q^S_{N,\mathsf{st},\rho,\eta^*_{N,\mathsf{st},\rho}}\right)$, and hence for every $\mathbf{x} \in (\mathbb{Z}_N)^{k_{\mathsf{in}}}$ for which G_1 successfully evaluates the function, it is also the case that $f^{N,\mathsf{st},\rho,\eta^*_{N,\mathsf{st},\rho}}_{\mathsf{out}}(\mathbf{x}) = 0 \pmod N$. Since the input N given to A_G is sampled as in $\mathsf{Exp}^{\mathsf{Seq}}_{\mathsf{DF},G}(\lambda)$, and the

st and ρ are sampled by A_G as in $\mathsf{Exp}^{\mathsf{Seq}}_{\mathsf{DF},G}(\lambda)$, this means that for N, st and ρ for which $\overline{\mathsf{Bal}}$ holds, it holds that

$$\begin{aligned}
&\alpha_N\left(f_{\mathsf{out}}^{N,\mathsf{st},\rho,\eta^*_{N,\mathsf{st},\rho}}\right)\\
&\quad\geq \Pr\left[\mathsf{Exp}^{\mathsf{Seq}}_{\mathsf{DF},G}(\lambda)=1\middle|\overline{\mathsf{Bal}}\wedge\mathsf{Con}\right]\\
&\quad\geq \Pr\left[\left(\mathsf{Exp}^{\mathsf{Seq}}_{\mathsf{DF},G}(\lambda)=1\right)\wedge\mathsf{Con}\middle|\overline{\mathsf{Bal}}\right]\\
&\quad= \Pr\left[\mathsf{Exp}^{\mathsf{Seq}}_{\mathsf{DF},G}(\lambda)=1\middle|\overline{\mathsf{Bal}}\right]-\Pr\left[\left(\mathsf{Exp}^{\mathsf{Seq}}_{\mathsf{DF},G}(\lambda)=1\right)\wedge\overline{\mathsf{Con}}\middle|\overline{\mathsf{Bal}}\right]\\
&\quad\geq \frac{\beta}{2}-\Pr\left[\overline{\mathsf{Con}}\middle|\overline{\mathsf{Bal}}\right] \qquad (10)\\
&\quad\geq \frac{\beta}{4}. \qquad (11)
\end{aligned}$$

Inequality (10) follows from (6) and inequality (11) follows by union bound over all $i\in[q_{\mathsf{eq}}]$. Combining inequalities (8), (9) and (11) concludes the analysis of case 2.

We complete the proof by analyzing the running time of A_G. To that end, we use the following proposition, which states that any single-output straight-line program S can be converted into a related straight-line program S' which computes the pairs $(P^S_{N,\sigma},1)$ and $(Q^S_{N,\sigma},1)$ in two different oracle queries, using roughly the same running time, parallelism and query complexity as S. In other words, one can "decouple" the numerator from the denominator which a straight-line program computes, with very little overhead. An almost identical proposition (in the univariate, single processor setting) was proven in [Jag07] and was also used in [AM09].

Proposition 4.3 *Let $w=w(\lambda),q=q(\lambda),t=t(\lambda)$ and $\ell=\ell(\lambda)$ be functions of the security parameter $\lambda\in\mathbb{N}$, let $N\leftarrow\mathsf{ModGen}(1^\lambda)$ and let $\sigma\in\{0,1\}^\ell$. For any single-output straight-line program S which runs in time t with w parallel processors, each of which making at most q sequential oracle queries, there exits a straight-line program S' which runs in time $O(t)$ with $3w$ parallel processors, each of which making at most $2q$ sequential oracle queries, and there exist indices $i_1,i_2\in[3w]$ and $j_1,j_2\in[2q]$, such that $(P^{S'}_{N,\sigma}[i_1,j_1],Q^{S'}_{N,\sigma}[i_1,j_1])=(P^S_{N,\sigma},1)$ and $(P^{S'}_{N,\sigma}[i_2,j_2],Q^{S'}_{N,\sigma}[i_2,j_2])=(Q^S_{N,\sigma},1)$.*

We turn to the runtime analysis. Steps 1 and 2 of A_G take $\mathsf{poly}(\lambda)$ time. The dominant part in each iteration of Step 3 is Step (a): The straight-line program S_i runs in time $\mathsf{poly}(\lambda)$ and makes at most $O(\mathsf{poly}(\lambda)+q_{\mathsf{op}})$ ring-operation queries per processor. This is by: (1) first, converting the explicit elements in st (of which there are at most $\mathsf{poly}(\lambda)$) to elements in the table (each conversion takes at most 2λ queries), and then (2) running the straight-line program guaranteed by Proposition 4.3 for the straight-line program which runs G_1 until the ith equality query (answering all equality queries up to that point according to η) and then outputting z_1-z_2 where z_1 and z_2 are the ring elements being compared. Hence,

the factorization algorithm A_{S_i} runs in time polynomial in λ by Lemma 4.2. Similarly, the factorization algorithm $A_{S_{\mathsf{out}}}$ from Step 5 runs in time polynomial in λ as well. This concludes the proof of Theorem 4.1. ■

We now conclude this section by presenting the proof of Lemma 4.2. In order to prove Lemma 4.2, we reduce the case of straight-line programs with multiple input elements and multiple parallel processors, to single-processor straight-line programs receiving just one ring element as input. In the latter setting, Aggarwal and Maurer [AM09] proved the following special case of Lemma 4.2.[8]

Lemma 4.4 ([AM09]). *Let $t = t(\lambda), q = q(\lambda)$ and $\ell = \ell(\lambda)$ be functions of the security parameter $\lambda \in \mathbb{N}$. For any straight-line program S which takes as input a single ring element and an explicit bit-string in $\{0,1\}^\ell$, and runs in time t with a single processor making at most q ring-operation queries, there exists an algorithm A_S which runs in time $O(t + q^3 \cdot \lambda^2)$, such that the following holds: For every $\lambda \in \mathbb{N}$, for every N which is outputted with positive probability by $\mathsf{ModGen}(1^\lambda)$ and for every $\sigma \in \{0,1\}^\ell$ which S may receive as an explicit input, if $P^S_{N,\sigma} \not\equiv 0 \pmod N$ then*

$$\Pr_{(a,b)\leftarrow A_S(N,\sigma)}\left[\begin{matrix} N = a \cdot b \\ a, b \in [N-1] \end{matrix}\right] \geq \frac{\alpha_N\left(P^S_{N,\sigma}(x)\right)}{8q}.$$

Equipped with this lemma, we turn to prove the general case of Lemma 4.2.

Proof of Lemma 4.2. Let $k = k(\lambda), t = t(\lambda), w = w(\lambda), \ell = \ell(\lambda)$ and $q = q(\lambda)$ be functions of the security parameter $\lambda \in \mathbb{N}$, and let S be a straight-line program which receives as input k ring elements, a modulus N and an additional ℓ-bit string, and runs in time t with w parallel processors, each making at most q oracle queries. Consider the following algorithm A_S:

Algorithm A_S

Input: An integer N sampled by $\mathsf{ModGen}(1^\lambda)$, and an ℓ-bit string σ.

1. Sample $i \leftarrow [k]$, and sample $\mathbf{x} = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_k) \leftarrow (\mathbb{Z}_N \setminus \{0\})^{k-1}$.
2. For every $j \in [k] \setminus \{i\}$, compute $g_i = \gcd(x_j, N)$, and if $g_j \notin \{1, N\}$, then output $(g_j, N/g_j)$ and terminate.
3. Let $f_{\mathbf{x}} := P^S_{N,\sigma}(x_1, \ldots, x_{i-1}, X, x_{i+1}, \ldots, x_k)$ be the uni-variate polynomial in the indeterminate X obtained from $P^S_{N,\sigma}$ by fixing X_j to be x_j for each $j \in [k] \setminus \{i\}$, let $S_{f_{\mathbf{x}}}$ be the single-processor straight-line which on explicit input (N, σ) outputs $(f_{\mathbf{x}}(X), 1)$, and let $A_{S_{f_{\mathbf{x}}}(N)}$ be the factoring algorithm guaranteed by Lemma 4.4. Invoke $(a, b) \leftarrow A_{S_{f_{\mathbf{x}}}}(N, \sigma)$ and output (a, b).

Let N be the input modulus to A_S, and let $a^*, b^* \in \{0,1\}^\lambda$ be its prime factors and assume that $P^S_{N,\sigma} \not\equiv 0 \pmod N$ (as otherwise the lemma trivially

[8] The lemma of Aggarwal and Maurer is stated in [AM09] in the terminology of their graph-based language for generic-ring algorithms, and without explicitly considering additional bit-string inputs (alongside the implicit access to ring elements). However, Lemma 4.4 as stated here follows directly from their proof.

holds). Let $\mathsf{success}$ be the event in which A_S outputs the correct factors of N, and denote the event in which X_i (where i is the index chosen by A_S in Step 1) has non-zero degree in $P^S_{N,\sigma}$ by $\mathsf{nonzero}$. Observe that since $P^S_{N,\sigma} \not\equiv 0 \pmod N$, then the probability of $\mathsf{nonzero}$ is at least $1/k$. By total probability it holds that

$$\Pr\left[\mathsf{sucess}\right] \geq \Pr\left[\mathsf{sucess}|\mathsf{nonzero}\right] \cdot \frac{1}{k}.$$

Denote by hit the event in which A_S terminates in Step 2 (hence, $\overline{\mathsf{hit}}$ is the event in which it terminates in Step 3). Since hit and $\mathsf{nonzero}$ are independent events, it holds that

$$\begin{aligned}\Pr\left[\mathsf{success}|\mathsf{nonzero}\right] &= \Pr\left[\mathsf{success}|\mathsf{hit} \wedge \mathsf{nonzero}\right]\Pr\left[\mathsf{hit}\right] \\ &\quad + \Pr\left[\mathsf{success}\middle|\overline{\mathsf{hit}} \wedge \mathsf{nonzero}\right] \cdot \Pr\left[\overline{\mathsf{hit}}\right] \\ &= 1 \cdot \Pr\left[\mathsf{hit}\right] + \Pr\left[\mathsf{success}\middle|\overline{\mathsf{hit}} \wedge \mathsf{nonzero}\right] \cdot (1 - \Pr\left[\mathsf{hit}\right]) \\ &\geq \Pr\left[\mathsf{success}\middle|\overline{\mathsf{hit}} \wedge \mathsf{nonzero}\right].\end{aligned}$$

We now wish to lower bound $\Pr\left[\mathsf{success}\middle|\overline{\mathsf{hit}} \wedge \mathsf{nonzero}\right]$. We observe that since $P^S_{N,\sigma} \not\equiv 0 \pmod N$, then conditioned on $\overline{\mathsf{hit}}$ and $\mathsf{nonzero}$ it is also the case that $f_{\mathbf{x}} \not\equiv 0 \pmod N$. To see why that is, assume that $\overline{\mathsf{hit}}$ and $\mathsf{nonzero}$ hold, and assume towards contradiction that $f_{\mathbf{x}} \equiv 0 \pmod N$. In this case, since X_i has non-zero degree in $P^S_{N,\sigma}$, there exists in $P^S_{N,\sigma}$ a monomial of the form $c \cdot X_{i_1} \cdots X_{i_m} \cdot X_i^{\delta}$ (where $c \in \mathbb{Z}$, $m \in \mathbb{N}$, $\delta > 0$ and $i_j \in [k]$ for every $j \in [m]$) such that c is not divisible by N. Assume without loss of generality that c is not divisible by a^* (if c is divisible by a^*, then it is not divisible by b^* and the proof is symmetric). But since $f \equiv 0 \pmod N$ it holds that $c \cdot x_{i_1} \cdots x_{i_m}$ is divisible by N. Finally, since $0 < x_{i_1}, \ldots, x_{i_m} < N$, there exists at least one $h \in [m]$ such that x_{i_h} is divisible by a^*. Therefore, $\gcd(x_{i_h}, N) = a^*$ and A_S outputs (a^*, b^*) in Step 2 with probability 1, in contradiction to the fact that we are conditioning on $\overline{\mathsf{hit}}$. Moreover, the single-processor straight-line program $S_{f_{\mathbf{x}}}$ makes at most $2\lambda \cdot k + w \cdot q$ oracle queries: 2λ queries to obtain each element in $\mathbf{x}$, and then $w \cdot q$ operations to compute the polynomial $f_{\mathbf{x}}$ (by a serialization of the multi-processor program S). Hence, if $P^S_{N,\sigma} \not\equiv 0 \pmod N$ then by Lemma

4.4 it holds that

$$\begin{aligned}
\Pr[\mathsf{success}] &\geq \frac{1}{k} \cdot \Pr\left[\mathsf{success} \middle| \overline{\mathsf{hit}} \wedge \mathsf{nonzero}\right] \\
&\geq \frac{\Pr_{\substack{\mathbf{x} \leftarrow (\mathbb{Z}_N \setminus \{0\})^{k-1} \\ x \leftarrow \mathbb{Z}_N}} \left[P_N^{A_{S_{f_{\mathbf{x}}}(N)}}(x) = 0 \pmod N\right]}{8k \cdot (2\lambda \cdot k + w \cdot q)} \qquad (12) \\
&= \frac{\Pr_{\substack{\mathbf{x} \leftarrow (\mathbb{Z}_N \setminus \{0\})^{k-1} \\ x \leftarrow \mathbb{Z}_N}} \left[f_{\mathbf{x}}(x) = 0 \pmod N\right]}{8k \cdot (2\lambda \cdot k + w \cdot q)} \\
&= \frac{\Pr_{\substack{\mathbf{x} \leftarrow (\mathbb{Z}_N)^{k-1} \\ x \leftarrow \mathbb{Z}_N}} \left[f_{\mathbf{x}}(x) = 0 \pmod N \;\middle|\; \forall x_j \in \mathbf{x}, x_j \neq 0\right]}{8k \cdot (2\lambda \cdot k + w \cdot q)} \\
&\geq \frac{\Pr_{\substack{\mathbf{x} \leftarrow (\mathbb{Z}_N)^{k-1} \\ x \leftarrow \mathbb{Z}_N}} \left[f_{\mathbf{x}}(x) = 0 \pmod N\right] - \Pr_{\mathbf{x} \leftarrow (\mathbb{Z}_N)^{k-1}} \left[\exists x_j \in \mathbf{x}, x_j = 0\right]}{\left(\Pr_{\mathbf{x} \leftarrow (\mathbb{Z}_N)^{k-1}} \left[\forall x_j \in \mathbf{x}, x_j \neq 0\right]\right) \cdot 8k \cdot (2\lambda \cdot k + w \cdot q)} \\
&\geq \frac{\Pr_{\substack{\mathbf{x} \leftarrow (\mathbb{Z}_N)^{k-1} \\ x \leftarrow \mathbb{Z}_N}} \left[f_{\mathbf{x}}(x) = 0 \pmod N\right] - (k-1) \cdot 2^{-\lambda+1}}{\left(1 - 2^{-\lambda}\right)^{k-1} \cdot 8k \cdot (2\lambda \cdot k + w \cdot q)} \qquad (13) \\
&\geq \frac{\alpha_N(P^S_{N,\sigma}) - (k-1) \cdot 2^{-\lambda+1}}{\left(1 - 2^{-\lambda}\right)^{k-1} \cdot 8k \cdot (2\lambda \cdot k + w \cdot q)}, \qquad (14)
\end{aligned}$$

where (12) follows from Lemma 4.4, (13) holds since $2^{\lambda-1} \leq N < 2^{\lambda}$ and (14) follows from the definition of $f_{\mathbf{x}}$.

We conclude by analyzing the running time of A_S. Steps 1 and 2 can be executed in time $O(k \cdot \lambda)$. The significant step is Step 3: $S_{f_{\mathbf{x}}}$ runs in time t and makes at most $2\lambda \cdot k + w \cdot q$ oracle queries. Hence, by Lemma 4.4, invoking $A_{S_{f_{\mathbf{x}}}}$ in Step 3 can be done in time $O\left(t + (2\lambda \cdot k + w \cdot q)^3 \cdot \lambda^2\right) = O\left(t + \lambda^5 \cdot k^3 + w^3 \cdot q^3\right)$. This concludes the proof of Lemma 4.2. ■

4.2 A Matching Upper Bound

In this section we prove a matching upper bound to the lower bound from Section 4.1. Roughly speaking, Theorem 4.5 states that for $q_{\mathsf{op}} = q_{\mathsf{op}}(\lambda)$, if DF has sequentiality depth at most q_{op}, then there is a generic attack which evaluates DF (according to Definition 3.1) while issuing at most q_{op} rounds of ring-operation queries (after a preprocessing stage), or else factoring is easy.

Theorem 4.5. *Let $T = T(\lambda)$ and $q_{\mathsf{op}} = q_{\mathsf{op}}(\lambda)$ be functions of the security parameter $\lambda \in \mathbb{N}$, and let DF be a straight-line delay function whose sequentiality depth is at most q_{op}. Then, there exist a pair $G = (G_0, G_1)$ of generic-ring algorithms where G_1 has polynomially-many parallel processors each of which issues at most q_{op} rounds of ring-operation queries, a standard-model probabilistic polynomial-time algorithm A, and a polynomial $p(\cdot)$, such that at least one of the following holds for infinitely many values of $\lambda \in \mathbb{N}$:*

1. $\Pr\left[\mathsf{Exp}^{\mathsf{Seq}}_{\mathsf{DF},G}(\lambda)=1\right] \geq 1/(2\cdot p(\lambda))$.
2. $\Pr_{\substack{N\leftarrow\mathsf{ModGen}(1^\lambda)\\(a,b)\leftarrow A(N,T)}}\left[(N=a\cdot b)\wedge(a,b\in[N-1])\right] > 1/(2\cdot p(\lambda))$.

Proof. Since DF has sequentiality depth at most q_{op}, it means that there exists a pair $G=(G_0,G_1)$ of polynomial-time generic-ring algorithms, where G_1 is a straight-line program with polynomially-many parallel processors making at most q_{op} rounds of operation queries, and there exists a polynomial $p(\cdot)$ such that

$$\Pr\left[\mathsf{Exp}^{\mathsf{SeqDepth}}_{\mathsf{DF},G}(\lambda)=1\right] > \frac{1}{p(\lambda)},$$

for infinitely many values of $\lambda\in\mathbb{N}$. Let $k_{\mathsf{in}}=k_{\mathsf{in}}(\lambda)$ be the number of ring elements which DF receives as input, and consider the following standard-model factoring algorithm A.

Algorithm A

Input: An integer N sampled by $\mathsf{ModGen}(1^\lambda)$, and the delay parameter $T\in\mathbb{N}$.

1. Sample $\mathbf{x}=(x_1,\ldots,x_{k_{\mathsf{in}}})\leftarrow(\mathbb{Z}_N)^{k_{\mathsf{in}}}$.
2. Compute $y=Q^{\mathsf{DF}}_{N,T}(x_1,\ldots,x_{k_{\mathsf{in}}})$, and compute $a=\gcd(y,N)$. If $a\notin\{1,N\}$, then output $(a,N/a)$ and terminate.
3. Run $G_0^{\mathbb{Z}_N}(N,T)$ to obtain a state $\mathsf{st}=(\mathsf{st}_0,\mathsf{st}_1,\ldots,\mathsf{st}_\ell)$, where $\mathsf{st}_1,\ldots,\mathsf{st}_\ell$ are ring elements.
4. Compute $z=Q^{G_1}_{N,T,\mathsf{st}_0}(\mathsf{st}_1,\ldots,\mathsf{st}_\ell,x_1,\ldots,x_{k_{\mathsf{in}}})$, and compute $b=\gcd(z,N)$. If $b\notin\{1,N\}$, then output $(b,N/b)$ and terminate.
5. Output $\perp$.

Denote by Factor the event in which A outputs a valid factorization of N, and denote by Inv the event in which both $y=Q^{\mathsf{DF}}_{N,T}(x_1,\ldots,x_{k_{\mathsf{in}}})$ and $z=Q^{G_1}_{N,T,\mathsf{st}_0}(\mathsf{st}_1,\ldots,\mathsf{st}_\ell,x_1,\ldots,x_{k_{\mathsf{in}}})$ are invertible modulo N. Observe that conditioned on $\mathsf{Exp}^{\mathsf{SeqDepth}}_{\mathsf{DF},G}(\lambda)=1$ and on Inv, it holds that

$$\frac{Q^{\mathsf{DF}}_{N,T}(x_1,\ldots,x_{k_{\mathsf{in}}})}{P^{\mathsf{DF}}_{N,T}(x_1,\ldots,x_{k_{\mathsf{in}}})}=\frac{P^{G_1}_{N,T,\mathsf{st}_0}(\mathsf{st}_1,\ldots,\mathsf{st}_\ell,x_1,\ldots,x_{k_{\mathsf{in}}})}{Q^{G_1}_{N,T,\mathsf{st}_0}(\mathsf{st}_1,\ldots,\mathsf{st}_\ell,x_1,\ldots,x_{k_{\mathsf{in}}})}.$$

In other words, G_1 successfully outputs the output of DF.

On the other hand, conditioned on $\mathsf{Exp}^{\mathsf{SeqDepth}}_{\mathsf{DF},G}(\lambda)=1$ and on $\overline{\mathsf{Inv}}$, at least one of y and z are not invertible modulo N. In this case, $a=\gcd(y,N)$ or $b=\gcd(z,N)$ are a prime factor of N and A outputs a valid factorization of N. Hence, by total probability, for infinitely many values of $\lambda\in\mathbb{N}$ it holds that

$$\begin{aligned}\Pr\left[\mathsf{Exp}^{\mathsf{SeqDepth}}_{\mathsf{DF},G}(\lambda)=1\right] &= \Pr\left[\mathsf{Exp}^{\mathsf{SeqDepth}}_{\mathsf{DF},G}(\lambda)=1\wedge\mathsf{Inv}\right]\\ &\quad+\Pr\left[\mathsf{Exp}^{\mathsf{SeqDepth}}_{\mathsf{DF},G}(\lambda)=1\wedge\overline{\mathsf{Inv}}\right]\\ &\leq \Pr\left[\mathsf{Exp}^{\mathsf{Seq}}_{\mathsf{DF},G}(\lambda)=1\right]+\Pr\left[\mathsf{Factor}\right]. \qquad (15)\end{aligned}$$

Therefore, at least one of the addends in (15) is greater than $1/(2p(\lambda))$ for infinitely many values of $\lambda\in\mathbb{N}$, concluding the proof. ■

References

[AM09] D. Aggarwal and U. Maurer. Breaking RSA generically is equivalent to factoring. In *Advances in Cryptology – EUROCRYPT '09*, pages 36–53, 2009.

[BBB+18] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch. Verifiable delay functions. In *Advances in Cryptology – CRYPTO '18*, pages 757–788, 2018.

[BBF18] D. Boneh, B. Bünz, and B. Fisch. A survey of two verifiable delay functions. Cryptology ePrint Archive, Report 2018/712, 2018.

[BCG15] J. Bonneau, J. Clark, and S. Goldfeder. On bitcoin as a public randomness source. Cryptology ePrint Archive, Report 2015/1015, 2015.

[BGJ+16] N. Bitansky, S. Goldwasser, A. Jain, O. Paneth, V. Vaikuntanathan, and B. Waters. Time-lock puzzles from randomized encodings. In *Proceedings of the 7th Conference on Innovations in Theoretical Computer Science*, pages 345–356, 2016.

[BGZ16] I. Bentov, A. Gabizon, and D. Zuckerman. Bitcoin beacon. arXiv:605.04559, 2016.

[BL96] D. Boneh and R. J. Lipton. Algorithms for black-box fields and their application to cryptography. In *Advances in Cryptology – CRYPTO '96*, pages 283–297, 1996.

[Bro05] D. R. L. Brown. Breaking RSA may be as difficult as factoring. Cryptology ePrint Archive, Report 2005/380, 2005.

[BV98] D. Boneh and R. Venkatesan. Breaking RSA may not be equivalent to factoring. In *Advances in Cryptology – EUROCRYPT '98*, pages 59–71, 1998.

[BW88] J. Buchmann and H. C. Williams. A key-exchange system based on imaginary quadratic fields. *Journal of Cryptology*, 1(2):107–118, 1988.

[CP18] B. Cohen and K. Pietrzak. Simple proofs of sequential work. In *Advances in Cryptology – EUROCRYPT '18*, pages 451–467, 2018.

[DGM+19] N. Döttling, S. Garg, G. Malavolta, and P. N. Vasudevan. Tight verifiable delay functions. Cryptology ePrint Archive, Report 2019/659, 2019.

[DK02] I. Damgård and M. Koprowski. Generic lower bounds for root extraction and signature schemes in general groups. In *Advances in Cryptology – EUROCRYPT '02*, pages 256–271, 2002.

[DN92] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *Advances in Cryptology – CRYPTO '92*, pages 139–147, 1992.

[EFK+20] N. Ephraim, C. Freitag, I. Komargodski, and R. Pass. Continuous verifiable delay functions. In *Advances in Cryptology – EUROCRYPT '20*, pages 125–154, 2020.

[FKL18] G. Fuchsbauer, E. Kiltz, and J. Loss. The algebraic group model and its applications. In *Advances in Cryptology – CRYPTO '18*, pages 33–62, 2018.

[FMP+19] L. D. Feo, S. Masson, C. Petit, and A. Sanso. Verifiable delay functions from supersingular isogenies and pairings. In *Advances in Cryptology – ASIACRYPT '19*, pages 248–277, 2019.

[GW11] C. Gentry and D. Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing*, pages 99–108, 2011.

[Jag07] T. Jager. Generic group algorithms. Master's Thesis, Ruhr Universität Bochum, 2007.

[JNT07] A. Joux, D. Naccache, and E. Thomé. When e-th roots become easier than factoring. In *Advances in Cryptology – ASIACRYPT '07*, pages 13–28, 2007.

[JR10] T. Jager and A. Rupp. The semi-generic group model and applications to pairing-based cryptography. In *Advances in Cryptology – ASIACRYPT '10*, pages 539–556, 2010.

[JS08] T. Jager and J. Schwenk. On the equivalence of generic group models. In *Proceedings of the 2nd International Conference on Provable Security*, pages 200–209, 2008.

[JS13] T. Jager and J. Schwenk. On the analysis of cryptographic assumptions in the generic ring model. *Journal of Cryptology*, 26(2):225–245, 2013.

[Kil92] J. Kilian. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 723–732, 1992.

[KLX20] J. Katz, J. Loss, and J. Xu. On the security of time-locked puzzles and timed commitments. Cryptology ePrint Archive, Report 2020/730, 2020.

[LR06] G. Leander and A. Rupp. On the equivalence of RSA and factoring regarding generic ring algorithms. In *Advances in Cryptology – ASIACRYPT '06*, pages 241–251, 2006.

[LW15] A. K. Lenstra and B. Wesolowski. A random zoo: sloth, unicorn, and trx. Cryptology ePrint Archive, Report 2015/366, 2015.

[Mau05] U. Maurer. Abstract models of computation in cryptography. In *Proceedings of the 10th IMA International Conference on Cryptography and Coding*, pages 1–12, 2005.

[Mic94] S. Micali. CS proofs. In *Proceedings of the 35th Annual IEEE Symposium on the Foundations of Computer Science*, pages 436–453, 1994.

[MMV11] M. Mahmoody, T. Moran, and S. P. Vadhan. Time-lock puzzles in the random oracle model. In *Advances in Cryptology – CRYPTO '11*, pages 39–50, 2011.

[MMV13] M. Mahmoody, T. Moran, and S. P. Vadhan. Publicly verifiable proofs of sequential work. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, pages 373–388, 2013.

[MSW19] M. Mahmoody, C. Smith, and D. J. Wu. A note on the (im)possibility of verifiable delay functions in the random oracle model. Cryptology ePrint Archive, Report 2019/663, 2019.

[MW98] U. M. Maurer and S. Wolf. Lower bounds on generic algorithms in groups. In *Advances in Cryptology – EUROCRYPT '98*, pages 72–84, 1998.

[Nec94] V. I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55(2):91–101, 1994.

[Pie19] K. Pietrzak. Simple verifiable delay functions. In *Proceedings of the 10th Conference on Innovations in Theoretical Computer Science*, pages 60:1–60:15, 2019.

[PW18] C. Pierrot and B. Wesolowski. Malleability of the blockchain's entropy. *Cryptography and Communications*, 10(1):211–233, 2018.

[RSS20] L. Rotem, G. Segev, and I. Shahaf. Generic-group delay functions require hidden-order groups. In *Advances in Cryptology – EUROCRYPT '20*, pages 155–180, 2020.

[RSW96] R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto, 1996.

[Sha19] B. Shani. A note on isogeny-based hybrid verifiable delay functions. Cryptology ePrint Archive, Report 2019/205, 2019.

[Sho97] V. Shoup. Lower bounds for discrete logarithms and related problems. In *Advances in Cryptology – EUROCRYPT '97*, pages 256–266, 1997.

[Wes19] B. Wesolowski. Efficient verifiable delay functions. In *Advances in Cryptology – EUROCRYPT '19*, pages 379–407, 2019.