Simple and Efficient Batch Verification Techniques for Verifiable Delay Functions

Lior Rotem*

Abstract

We study the problem of *batch verification* for verifiable delay functions (VDFs), focusing on proofs of correct exponentiation (PoCE), which underlie recent VDF constructions. We show how to compile any PoCE into a batch PoCE, offering significant savings in both communication and verification time. Concretely, given any PoCE with communication complexity c, verification time t and soundness error δ, and any pseudorandom function with key length $\mathsf{k_{prf}}$ and evaluation time t_{prf}, we construct:

- A batch PoCE for verifying n instances with communication complexity $m \cdot c + \mathsf{k_{prf}}$, verification time $m \cdot t + n \cdot m \cdot O(t_{\mathsf{op}} + t_{\mathsf{prf}})$ and soundness error $\delta + 2^{-m}$, where λ is the security parameter, m is an adjustable parameter that can take any integer value, and t_{op} is the time required to evaluate the group operation in the underlying group. This should be contrasted with the naïve approach, in which the communication complexity and verification time are $n \cdot c$ and $n \cdot t$, respectively. The soundness of this compiler relies only on the soundness of the underlying PoCE and the existence of one-way functions.
- An improved batch PoCE based on the low order assumption. For verifying n instances, the batch PoCE requires communication complexity $c + \mathsf{k_{prf}}$ and verification time $t + n \cdot (t_{\mathsf{prf}} + \log(s) \cdot O(t_{\mathsf{op}}))$, and has soundness error $\delta + 1/s$. The parameter s can take any integer value, as long as it is hard to find group elements of order less than s in the underlying group. We discuss instantiations in which s can be exponentially large in the security parameter λ.

If the underlying PoCE is constant round and public coin (as is the case for existing protocols), then so are all of our batch PoCEs. This implies that they can be made non-interactive using the Fiat-Shamir transform.

Additionally, for RSA groups with moduli which are the products of two safe primes, we show how to efficiently verify that certain elements are not of order 2. This protocol, together with the second compiler above and any (single-instance) PoCE in these groups, yields an efficient batch PoCE in safe RSA groups. To complete the picture, we also show how to extend Pietrzak's protocol (which is statistically sound in the group QR_N^+ when N is the product of two safe primes) to obtain a statistically-sound PoCE in safe RSA groups.

*School of Computer Science and Engineering, Hebrew University of Jerusalem, Jerusalem 91904, Israel. Email: lior.rotem@cs.huji.ac.il. Supported by the Adams Fellowship Program of the Israel Academy of Sciences and Humanities and by the European Union's Horizon 2020 Framework Program (H2020) via an ERC Grant (Grant No. 714253).

1 Introduction

Verifiable delay functions (VDFs), recently formalized by Boneh et al. [BBB+18], have proven to be extremely useful in a wide array of exciting applications. These include, among others, verifiable randomness beacons (e.g., [LW15] and also [GS98, BCG15, BGZ16, PW18, HYL20, GLO+21]), resource-efficient blockchains [CP19], computational time-stamping (e.g., [CE12, LSS20] and the references therein) and time-based proofs of replication (e.g., [Ler14, ABB+16, Pro17, BDG17, Fis19]). Roughly speaking, a VDF is a function $f : \mathcal{X} \to \mathcal{Y}$ which is defined with respect to a delay parameter T and offers the following sequentiality guarantee: It should not be possible to compute f on a randomly-chosen input in time less than T, even with preprocessing and polynomially-many parallel processors. However, the function should be computable in time polynomial in T. Moreover, the function should be efficiently verifiable: For an input $x \in \mathcal{X}$, it should be possible to produce alongside the output $y \in \mathcal{Y}$, a short proof π asserting that indeed $y = f(x)$. Verifying the proof should be much quicker than computing the function anew.

Proofs of correct exponentiation. The main VDF candidates currently known are based on the "repeated squaring" function in groups of unknown order, such as RSA groups or class groups of imaginary quadratic fields. This function – first introduced for delay purposes by Rivest, Shamir and Wagner [RSW96] – is defined by $x \mapsto x^{2^T}$, where T is the delay parameter and the exponentiation is with respect to the group operation. The recent and elegant works of Wesolowski [Wes19], of Pietrzak [Pie19] have augmented the repeated squaring function with non-interactive proofs, yielding full-fledged candidate VDF constructions (see also the survey of Boneh et al. [BBF18] covering these constructions). Both of these proofs are based on applying the Fiat-Shamir heuristic [FS86] to *succinct proofs of correct exponentiation*. These are protocols in which a (possibly malicious) prover tries to convince a verifier that $y = x^e$, for a joint input consisting of two group elements x and y and an arbitrary (and potentially very large) exponent e.[1] The succinctness of these protocols manifests itself both in their communication complexity, and in the verifier's running time, which is much lesser than the time it would take the verifier to compute x^e on her own. Very recently, in an independent and concurrent work, Block et al. [BHR+21] showed how to generalize Pietrzak's protocol to obtain a proof of correct exponentiation that is information-theoretically secure in any group of unknown order.

Verifying multiple VDF outputs. In many of the applications of VDFs, one might be (and in some cases even likely to be) interested in verifying not only one but many VDF outputs at once. Examples for such scenarios include (but are not restricted to) verifying that a storage service maintains multiple replicas of the same file via a VDF-based proof of replication; verifying the shared randomness produced by a VDF-based randomness beacon during the last several epochs; and verifying the time-stamps of multiple files stamped using a VDF-based time-stamping scheme. Unfortunately, verifying multiple VDF outputs naïvely, by verifying the proof for each of them separately and independently, comes at a premium: If one wishes to verify n individual VDF proofs, each of which is ℓ bits long and takes time t to verify, then verifying all of them using the aforesaid naïve approach results in a total proof size (and hence communication overhead) of $n \cdot \ell$ and verification time of $n \cdot t$.

[1] Often, these protocols offer only computational soundness guarantee. Nevertheless, we use the name *proofs* (rather than *arguments*) throughout, for more concise presentation and for consistency with previous works (e.g., [BBF19]).

Existing approaches. The related and fundamental problem of verifying many exponentiations in cryptographic groups, traces back to the seminal work of Bellare, Garay and Rabin [BGR98], which presented elegant batch verification algorithms. However, their work, motivated by the task of batch verification of signatures, did not consider the setting of an external prover and the efficiency considerations attached to it (i.e., succinctness). Moreover, their more efficient approach and its analysis rely on cyclic groups of prime order, which seem somewhat unlikely to accommodate VDF constructions [RSS20] (see Section 1.2). In the context of VDF verification, Wesolowski[2] recently presented a "batch version" of his proof of correct exponentiation. Alas, the soundness of this batch proof is proven under the adaptive root assumption, which is a new and rather strong assumption in groups of unknown order. In particular this assumption is stronger than the low order assumption which underlies Pietrzak's protocol [BBF18, Pie19],[3] and making this assumption is of course undesirable when starting from the information-theoretically sound protocol of Block et al. [BHR+21]. This state of affairs urges the search for succinct and efficient batch proofs of correct exponentiation which rely on weaker assumptions than the adaptive root assumption.

1.1 Our Contributions

We present simple and efficient batch verification techniques for proofs of correct exponentiation, extending the basic techniques of Bellare, Garay and Rabin to the external-prover setting and to composite-order groups. In conjunction with current VDF candidates based on proofs of correct exponentiation for the repeated squaring function [RSW96, Wes19, Pie19, BHR+21], our techniques immediately give rise to VDFs with batch verification. Our compilers rely on weaker assumptions than currently-known batching techniques for verifiable delay functions, paving the way to a variety of new instantiations.

Batch proofs of correct exponentiation. We define the notion of a *batch proof of correct exponentiation*. This is a protocol in which a prover and a verifier share as input n pairs $(x_1, y_1), \ldots, (x_n, y_n)$ of group elements and an exponent $e \in \mathbb{N}$, and the prover attempts to convince the verifier that $x_i^e = y_i$ for each $i \in [n]$.[4] Loosely speaking, we say that a batch proof of correct exponentiation has soundness error δ if in case $x_i^e \neq y_i$ for some $i \in [n]$, no efficient (malicious) prover can convince a verifier to accept with probability greater than $\delta + \mathsf{negl}(\lambda)$, where $\lambda \in \mathbb{N}$ is the security parameter (see Section 3 for the formal definition).

A general compiler. As our first main contribution, we show how to compile any proof of correct exponentiation into a batch proof of correct exponentiation, offering significant savings in both communication and verification time, relative to the naïve transformation. The soundness of this compiler essentially relies on an information-theoretic argument, which is then derandomized using a pseudorandom function. This makes it a generic compiler which can be applied in any group, as

[2] In the updated longer version of his work [Wes20].

[3] For example, in cyclic groups of prime (and publicly known) order, the low order assumption holds information-theoretically, whereas the adaptive root assumption does not hold. Moreover, even in groups which are believed to be of unknown order, the adaptive root assumption seems to be a stronger one: For instance, for a modulus N which is the product of two safe primes, the low order assumption holds information-theoretically in the group of quadratic residues modulo N. This is in contrast to the adaptive root assumption which is at least as strong as assuming the hardness of factoring N. See Section 2 for details.

[4] Note that our definition of batch proofs of correct exponentiation deals with scenarios in which all instances should be verified with respect to the same exponent. We discuss this point further in Section 3, and leave it as an interesting open question to construct batch proofs for different exponents.

long as the underlying proof of correct exponentiation is sound in this group, hence also making it compatible with the new proof of correct exponentiation of Block et al. [BHR+21].

Theorem 1.1 (informal)**.** *Let $\mathbb{G}$ be a group and assume the existence of a one-way function and of a proof of correct exponentiation in $\mathbb{G}$ with communication complexity $c = c(\lambda, e)$, verification time $t = t(\lambda, e)$ and soundness error $\delta = \delta(\lambda)$, where $\lambda \in \mathbb{N}$ is the security parameter and $e \in \mathbb{N}$ is the exponent. Then, for any $n, m \in \mathbb{N}$, there exists a batch proof of correct exponentiation for n pairs of elements in the group $\mathbb{G}$, with communication complexity $c_{\mathsf{batch}} = c \cdot m + \lambda$, verification time $t_{\mathsf{batch}} = m \cdot t + n \cdot m \cdot \mathsf{poly}(\lambda)$ and soundness error $\delta_{\mathsf{batch}} = \delta + 2^{-m}$.*

An improved compiler based on the low order assumption. Our second main contribution is an improved compiler, whose soundness is based on the low order assumption in groups of unknown order, recently introduced by Boneh et al. [BBF18]. Roughly speaking, for an integer ℓ, the ℓ-low order assumption asserts that one cannot efficiently come up with a group element $z \neq 1$ and an exponent $\omega < \ell$ such that $z^\omega = 1$. This compiler enjoys significant improvements over our general compiler: The communication complexity is now completely independent of the desired soundness guarantee (i.e., one can reduce the soundness error without increasing communication), and the running time of the verifier is also improved. Concretely, we prove the following theorem.

Theorem 1.2 (informal)**.** *Let $\mathbb{G}$ be a group and assume the existence of a one-way function and of a proof of correct exponentiation in $\mathbb{G}$ with communication complexity $c = c(\lambda, e)$, verification time $t = t(\lambda, e)$ and soundness error $\delta = \delta(\lambda)$, where $\lambda \in \mathbb{N}$ is the security parameter and $e \in \mathbb{N}$ is the exponent. Assume that the ℓ-low order assumption holds in $\mathbb{G}$ for an integer $\ell = \ell(\lambda)$. Then, for any $n \in \mathbb{N}$ and $s \leq \ell$, there exists a batch proof of correct exponentiation for n pairs of elements in the group $\mathbb{G}$, with communication complexity $c_{\mathsf{batch}} = c + \lambda$, verification time $t_{\mathsf{batch}} = t + O(n \cdot \log(s) \cdot \mathsf{poly}(\lambda))$, and soundness error $\delta_{\mathsf{batch}} = \delta + 1/s$.*

In Section 6, we also discuss why the low order assumption is necessary for our compiler to yield the soundness guarantees of Theorem 1.2.

Instantiating the compiler. The compiler from Theorem 1.2 relies on the same techniques as those underlying Wesolowski's [Wes20] batch proof (which, as mentioned above, can be traced back to Bellare, Garay and Rabin [BGR98]). However, our compiler is modular and our analysis of its soundness is based solely on the low order assumption (compared to Wesolowski's reliance on the adaptive root assumption). This means that our compiler can be applied to both the protocols of Wesolowski and of Pietrzak, without making any further assumptions beyond those required by their single-instance protocols (and one-way functions for derandomization purposes). Concretely, there are currently three main candidates for families of groups in which the low order assumption is plausible:[5]

- **The groups QR_N and QR_N^+.** The low order assumption holds information-theoretically in the group QR_N of quadratic residues modulo N when N is the product of two safe primes (as well as in the isomorphic group QR_N^+ of *signed* quadratic residues modulo N, in which group membership is efficiently recognizable). It may also rely on the assumption that the low order problem is computationally hard in these groups for other choices of N. The reader is referred to Section 2 for further details regarding these groups.

[5] For a more in-depth discussion see Section 6 and the work of Boneh et al. [BBF18].

- **RSA groups.** The low order assumption cannot hold in the RSA group $\mathbb{Z}_N^*$ since $-1 \in \mathbb{Z}_N$ is always of order two in this group. Boneh et al. [BBF18] suggested to work over the quotient group $\mathbb{Z}_N^*/\{\pm 1\}$ instead. We consider two additional possibilities. One is to settle on a slightly weaker soundness guarantee: If the verifier accepts, it must be the case that $x_i^e \in \{y_i, -y_i\}$ for every $i \in [n]$. Observe, that this requirement indeed seems compatible with many of the applications of VDFs mentioned above.[6] When N is the product of two safe primes, we show (following Seres and Burcsi [SB20]) that this weaker notion of soundness for our compiler is actually implied by the hardness of factoring N. The second option is to compose our compiler with an additional protocol, specifically tailored in order to prove that $y_i \neq -x_i^e$ for each i. We discuss this approach below.
- **Class groups of imaginary quadratic fields.** The security of the low order assumption in these groups is still unclear [BBF18, BKS+20], but at least for now, there are possible parameters for which the low order assumption remains unbroken in these groups.

Strong soundness in RSA groups via proofs of order. As discussed above, when our compiler from Theorem 1.2 is used within RSA groups, we can obtain only a weaker form of soundness. The issue is that a malicious prover can still convince the verifier that $x_i^e = y_i$ for every i, even though there exists an index j for which $y_j/x_j^e = -1$. To remedy this situation we present a protocol that allows the prover to convince the verifier that $\mathsf{order}(y_i/x_i^e) \neq 2$ for every i, and hence in particular $y_i/x_i^e \neq -1$ for every i. The protocol builds on the work of Di Crescenzo et al. [CKK+17], but extends it in a non-trivial manner to save in communication (or proof size, when Fiat-Shamir is applied). It enjoys efficient verification and information-theoretic soundness when the modulus N of the RSA group is the product of two safe primes, and it can be made succinct (i.e., with communication complexity is independent of the number n of pairs of group elements) using a pseudorandom function. Hence, in such groups, executing this protocol in parallel to our compiler from Theorem 1.2 yields a full-fledged sound compiler in RSA groups, without compromising on weaker soundness notions or making strong assumptions. In Theorem 1.3 below, by "safe RSA groups" we mean RSA groups whose modulus is the product of two λ-bit safe primes.

Theorem 1.3 (informal)**.** *Assume the existence of a one-way function and of a proof of correct exponentiation in safe RSA groups with communication complexity $c = c(\lambda, e)$, verification time $t = t(\lambda, e)$ and soundness error $\delta = \delta(\lambda)$, where $\lambda \in \mathbb{N}$ is the security parameter and $e \in \mathbb{N}$ is the exponent. Then, for any $n, m \in \mathbb{N}$ and $s < 2^{\lambda-1}$, there exists a batch proof of correct exponentiation for n pairs of elements in safe RSA groups, with communication complexity $c_{\mathsf{batch}} = c + O(\lambda)$, verification time $t_{\mathsf{batch}} = t + O(n \cdot m \cdot \log(s) \cdot \mathsf{poly}(\lambda))$, and soundness error $\delta_{\mathsf{batch}} = \delta + 1/s + 2^{-m}$.*

A statistically-sound proof of correct exponentiation in RSA groups. To complete the picture, we present a proof of correct exponentiation in standard (safe) RSA groups.[7] The protocol is obtained by extending Pietrzak's protocol [Pie19] with techniques similar to those used to prove Theorem 1.3. The protocol actually achieves statistical soundness in safe RSA groups, with very little overhead in terms of communication and verification time when compared to Pietrzak's protocol.

[6] For example, when it comes to proofs of replication, if a file is retrievable given an encoding y which is the output of a VDF, then it is also retrievable given $-y$. As an additional example, in the context of verifiable randomness beacons, this weakened soundness guarantee gives a malicious prover the ability to convince the verifier that the shared randomness is $-y$, when in fact it should be y, but no more than that.

[7] This is in contrast to the quotient group $\mathbb{Z}_N^* \setminus \{\pm 1\}$ or the subgroups QR_N and QR_N^+ of $\mathbb{Z}_N^*$.

Theorem 1.4 (informal)**.** *There exists a statistically-sound proof of correct exponentiation in safe RSA groups, whose communication complexity and verification time essentially match those of Pietrzak's protocol.*

Though the statistically-sound proof of correct exponentiation of Block et al. [BHR+21] can also be instantiated in safe RSA groups, their protocol incurs a factor λ overhead in communication complexity when compared to Pietrzak's protocol. In contrast, our protocol only incurs an overhead of factor 2 in communication complexity.

Interpreting our results. We make two clarifications in order to help the reader interpret the above results. Firstly, we emphasize that the parameters m (from Theorems 1.1 and 1.3) and s (from Theorems 1.2 and 1.3) do not scale with the number n of pairs (x_i, y_i) to be verified, and can be fine-tuned at will to achieve the desired tradeoff between the soundness error of the batch protocol on the one hand, and the communication complexity and verifier's running time on the other hand. Secondly, we stress that in all of the above theorems, the polynomials referred to by poly are fixed polynomials that depend only on λ, and do not scale with neither n nor the exponent e. These two points have several important implications:

- The communication overhead incurred by our compilers is completely independent of n.
- The verification time depends linearly on n, but in all of our compilers, we manage to "decouple" the terms which depend on n from the terms which depend on the original verification time t in the underlying proof of correct exponentiation protocol (and hence we also decouple n from the exponent e). This should be contrasted with the naïve solution discussed above, in which the verification time is $t \cdot n$. Moreover, observe that some linear dependency on n seems unavoidable, since merely reading the verifier's input takes time at least n.
- One can set m to be super-logarithmic in the security parameter λ (e.g., by setting $m(\lambda) = \log(\lambda) \cdot \log^*(\lambda)$) in Theorems 1.1 and 1.3, and s to be super-polynomial in Theorems 1.2 and 1.3, to obtain protocols with negligible soundness error, with only slightly greater communication complexities and verification times than those of the underlying proof of correct exponentiation.

Applying the Fiat-Shamir heuristic. All of our compilers add only a single *public coin* message from the verifier to the prover. Consequently, if the Fiat-Shamir heuristic [FS86] can be applied in the random oracle model to the underlying proof of correct exponentiation, then it can be applied to resulted batch proof as well (as long as m and s are set such that the soundness error is negligible). In particular, the Fiat-Shamir heuristic may be applied to the compiled versions (via our compilers) of the protocols of Wesolowski [Wes19], of Pietrzak [Pie19], and of Block et al. [BHR+21] to obtain non-interactive batch proofs of correct exponentiation. See Section 5 and the related work of Lombardi and Vaikuntanathan [LV20] for a more exhaustive discussion on the matter.

Communication in the interactive setting. As illustrated below, we use pseudorandom functions in order to shrink the length of the public coin message from the verifier to the prover. This is necessary only in the interactive setting, since when the Fiat-Shamir heuristic is applied this message is computed locally by the prover (and hence does not affect the proof's length). Indeed, verification of VDFs is typically considered to be non-interactive, but we nevertheless believe that exploring interactive verification of VDFs is interesting and well-justified by applications in which verifiaction is done by a single verifier in an online manner, such as VDF-based proofs of storage. Using an interactive protocol in such settings eliminates the need to rely on the Fiat-Shamir transform.

1.2 Additional Related Work and Open Problems

Batch verification for group exponentiation. The line of works that seems most related to ours was initiated by the seminal work of Bellare, Garay and Rabin [BGR98] (following up on the works of Fiat [Fia89], Naccache et al. [NMV+94] and of Yen and Laih [YL95][8]) and considers the following problem: Let $\mathbb{G}$ be a cyclic group and let g be a generator of the group. The task, given n exponents $x_1, \ldots, x_n$ and n group elements $h_1, \ldots, h_n$, is to verify that $g^{x_i} = h_i$ for each $i \in [n]$. Bellare et al. presented several approaches for solving this problem, exhibiting different savings in terms of computational costs vis-à-vis the naïve solution of raising g to the power of each x_i.

Our compilers are inspired by two elegant techniques of Bellare et al. – the "random subsets" technique and the "random exponents" technique – but there are some key differences between their work and ours. Firstly, we embed these techniques within the framework of succinct proofs of correct exponentiation (which we extend to the batch setting). This setting presents its own set of unique technical challenges, the main one being reducing the communication overhead (or proof size, in the non-interactive setting). This challenge does not arise in the setting considered in their work (and in follow-up works), which is motivated by batch verification of signatures. Secondly, the random exponents technique as proposed by Bellare et al. and its analysis explicitly and inherently assumes that the group at hand is of prime order. Such groups do not seem to enable VDF constructions, as Rotem, Segev and Shahaf [RSS20] recently showed how break the sequentiality of any such construction in the generic-group model. We extend the approach underlying the random exponents technique to composite-order groups.

In a concurrent and independent work, Block et al. [BHR+21] also extended the random subsets technique of Bellare et al. that we use to derive Theorem 1.1, in the context of hidden-order groups (though they did not explicitly observe the connection between the work of Bellare et al. and theirs). Their motivation was to extend Pietrzak's protocol [Pie19] to obtain an information-theoretically sound protocol, while ours is proof batching, but the application of the random subsets technique is quite similar in both cases.

Di Crescenzo et al. [CKK+17] considered the related problem of batch delegation of exponentiation in RSA groups, while extending the random exponents technique of Bellare, Garay and Rabin. Our treatment of RSA groups is also inspired by the techniques of Di Crescenzo et al. but their protocol includes a communication overhead which is linear in the number n of exponentiations to be delegated (and verified) – which in our setting, is exactly what we are trying to avoid. We manage to get rid of this dependency altogether.

A long line of follow-up works succeeded the work of Bellare et al. suggesting various improvements to their techniques in various settings (see for example [BP00, CL06, CY07, CHP07, CL15] and the references therein). An interesting open question is whether some of these techniques can be used in order to improve our results.

Other VDF candidates. This work focuses on VDF candidates which are based on the repeated squaring function in groups of unknown order. Other candidates have also been proposed over the last couple of years. Some of which are based on different assumptions, such as candidates based on super-singular isogenies [FMP+19, Sha19], and the VeeDo VDF candidate in prime fields [Sta20]; while other constructions (e.g., [EFK+20, DGM+20]) achieve various desired properties. An interesting possible direction for future research is to enable batch verification to these candidate VDFs as well, either relying on our techniques or presenting new ones tailored specifically for these candidates.

[8] See also [CHP07] and the many references therein.

Necessity of one-way functions in the interactive setting. Informally put, our protocols use some function f to derandomize a long public coin message from the verifier to the prover. In our proposed instantiations, f is implemented using a cryptographic pseudorandom generator or pseudorandom function (both are known to exist assuming one-way functions [GGM86, Nao91, HIL+99]), and we show that the soundness of this approach is "as good" as the soundness before the derandomization. However, in all of our protocols, we only need the output of f on a uniformly-random input to satisfy some specific statistical property. Hence, an interesting open question is whether our reliance on one-way functions is necessary, or can f be instantiated without them while still offering comparable security guarantees. One possibility is to use a Nissan-Wigderson type pseudorandom generator [NW94, IW97], relying on worst-case assumptions. Another is to use ϵ-biaset sets (see for example [NN93, AGH+92, Ta-17] and the references therein), though this approach seems to inherently yield a slightly worse communication to soundness tradeoff.

1.3 Technical Overview

In this section we provide a high-level overview of the techniques used throughout the paper. In this overview, we ignore various technical subtleties that arise in the full proofs.

The basic Random Subset Compiler. We start by describing the basic idea which underlines the generic compiler guaranteed by Theorem 1.1. Let Π be any (single-instance) proof of correct exponentiation, and let $(x_1, y_1), \ldots, (x_n, y_n)$ and $e \in \mathbb{N}$ be the n pairs of group elements and the exponent that are shared by the prover and the verifier as input. Recall that the prover wishes to convince the verifier that $y_i = x_i^e$ for each $i \in [n]$. The basic technical observation underlying the compiler is that if for some index $i \in [n]$ it holds that $y_i \neq x_i^e$, then with probability at least $1/2$ over the choice of a uniformly random subset $\mathcal{S}$ of $[n]$, it holds that $\prod_{j \in \mathcal{S}} y_j \neq \left(\prod_{j \in \mathcal{S}} x_j\right)^e$. This observation then naturally lends itself to obtain a batch proof of correct exponentiation: First, the verifier simply chooses such a subset $\mathcal{S}$ uniformly at random and sends it to the prover. Then, the verifier and the prover execute Π on shared input $(x' = \prod_{j \in \mathcal{S}} x_j, y' = \prod_{j \in \mathcal{S}} y_j, e)$; that is, the prover uses Π to convince the verifier that indeed $y' = (x')^e$. By the above observation, if Π has soundness error δ, then the compiled batch protocol has soundness error at most $\delta + 1/2$. The reader is referred to Section 4 for a formal description of the compiler and its analysis.

Amplifying soundness and reducing communication. The above compiler suffers from two main drawbacks: The soundness error of the resulted batch protocol is at least $1/2$, and its communication complexity is necessarily linear in the number n of pairs of group elements, since a uniformly chosen subset of $[n]$ has n bits of entropy. Fortunately, this situation is easy to remedy by introducing two simple modifications to the protocol. First, instead of choosing just one subset $\mathcal{S}$ of $[n]$, the verifier chooses m such subsets $\mathcal{S}_1, \ldots, \mathcal{S}_m$ for some integer m which parameterizes the compiler. Then, the verifier and the prover run m parallel executions of the underlying protocol Π, where in the ith execution, they run on shared input $(x'_i = \prod_{j \in \mathcal{S}_i} x_j, y'_i = \prod_{j \in \mathcal{S}_i} y_j, e)$. Suppose that $y_j \neq x_j^e$ for some $j \in [n]$. Using the observation from the previous paragraph, it is straightforward that if $\mathcal{S}_1, \ldots, \mathcal{S}_m$ are chosen independently and uniformly at random from all subsets of $[n]$, then the probability that $y'_i = (x'_i)^e$ for each $i \in [m]$ is at most 2^{-m}. Hence, if Π has soundness error δ, then the compiled batch protocol has soundness error at most $\delta + 2^{-m}$ (regardless of whether the soundness of the underlying protocol Π is amplified via parallel repetition).

In order to attend to the large communication complexity of the compiled protocol (now the verifier has to send $k \cdot n$ bits to the prover), we derandomize the choice of the sets $\mathcal{S}_1, \ldots, \mathcal{S}_m$.

Instead of sampling these sets explicitly and sending their description to the prover, the verifier now samples and sends a short key k to a pseudorandom function PRF. This key can now succinctly represent $\mathcal{S}_1, \ldots, \mathcal{S}_m$; for example by letting $j \in \mathcal{S}_i$ if and only if $\mathsf{PRF}_k(i, j) = 1$, for every $i \in [m]$ and $j \in [n]$. Roughly speaking, the security of the pseudorandom function guarantees that if $y_j \neq x_j^e$ for some $j \in [n]$, then the probability that $y'_i = (x'_i)^e$ for each $i \in [m]$ is at most $2^{-m} + \mathsf{negl}(\lambda)$, where λ is the security parameter. See Section 5 for a formal description and analysis of the strengthened protocol.

The Random Exponents Compiler. We now describe the idea behind our improved compiler based on the low order assumption (Theorem 1.2). The observation is that the basic Random Subset Compiler as described above can be viewed in a more general manner: The verifier chooses $\alpha_1, \ldots, \alpha_n \leftarrow \{0, 1\}$, and then the two parties invoke the underlying protocol Π on joint input $x' = \prod_{i \in [n]} x_i^{\alpha_i}$, $y' = \prod_{i \in [n]} y_i^{\alpha_i}$ and e. The idea is to now let the verifier choose $\alpha_1, \ldots, \alpha_n$ from a large domain; concretely, from the set $[s]$ for some appropriately chosen parameter $s \in \mathbb{N}$. As before, after the verifier chooses $\alpha_1, \ldots, \alpha_n$ and sends them over to the prover, the two parties invoke the underlying protocol Π on shared input (x', y', e) for asserting that $y' = (x')^e$. It should be noted that as in the Random Subset Compiler, the choice of $\alpha_1, \ldots, \alpha_n$ can be derandomized using a pseudorandom function in order to save in communication, without significantly affecting the soundness of the compiler.

The proof of soundness of the compiled protocol now has to rely on the s-low order assumption, which roughly speaking, says that it should be hard to find a group element x and a positive integer $\omega < s$ such that $x^\omega = 1$. We wish to argue that if the s-low order assumption holds in the group at hand and $y_j \neq x_j^e$ for some $j \in [n]$, then enlarging the domain from which $\alpha_1, \ldots, \alpha_n$ are drawn (up to and including $[s]$) proportionally reduces the probability that $y' = (x')^e$. This is done by a reduction, which we now informally describe, to the s-low order assumption. For the formal statement and reduction, we refer the reader to Section 6.

Let $(x_1, y_1), \ldots, (x_n, y_n)$ be n pairs of elements in a group $\mathbb{G}$ such that at least one index i satisfies $y_i \neq x_i^e$, and let i^* be the first such index. Consider the following algorithm A for finding a low order element in $\mathbb{G}$. A first samples $n + 1$ integers $\alpha_1, \ldots, \alpha_{i^*-1}, \alpha_{i^*+1}, \ldots, \alpha_n, \beta, \beta'$ uniformly at random from $[s]$. Then, it checks that $\left(x_{i^*}^{\beta} \cdot \prod_{i \in [n] \setminus \{i^*\}} x_i^{\alpha_i}\right)^e = y_{i^*}^{\beta} \cdot \prod_{i \in [n] \setminus \{i^*\}} y_i^{\alpha_i}$, that $\left(x_{i^*}^{\beta'} \cdot \prod_{i \in [n] \setminus \{i^*\}} x_i^{\alpha_i}\right)^e = y_{i^*}^{\beta'} \cdot \prod_{i \in [n] \setminus \{i^*\}} y_i^{\alpha_i}$, and that $\beta \neq \beta'$. If any of these conditions does not hold, it aborts. Otherwise, if all of these conditions check out, A outputs the group element $z = y_{i^*}/x_{i^*}^e$ together with the exponent $\omega = |\beta - \beta'|$. It is easy to verify that if both of the equalities checked by A hold, then this implies that $z^\omega = 1$, while the inequality checked by A implies that indeed $\omega \neq 0$. Now assume towards contradiction that the probability that $y' = (x')^e$ is at least $1/s + \epsilon$ for some $\epsilon > 0$. Then, a careful analysis shows that the probability that A does not abort is at least ϵ^2. Informally, this implies that if the s-low order assumption holds in $\mathbb{G}$ and the underlying protocol Π has soundness error δ, then the compiled batch protocol has soundness error at most $\delta + 1/s + \mathsf{negl}(\lambda)$.

Strong soundness in safe RSA groups. Recall that, as mentioned in Section 1.1, the s-low order assumption cannot hold in the group $\mathbb{Z}_N^*$ for any $s \geq 2$, since $N - 1$ is always an element of order 2 in the group. Therefore, the Random Exponents Compiler obtains a weaker form of soundness when applied in $\mathbb{Z}_N^*$, guaranteeing only that $y_i = \pm x_i^e$. To counter this problem, we present a protocol for proving that $\mathsf{order}(y_i/x_i^e) \neq 2$ for every i. Our basic approach follows a technique by Di Crescenzo et al. [CKK+17] for proving that $\mathsf{order}(y/x^e) \neq 2$ for $x, y \in \mathbb{Z}_N^*$ and an odd exponent e. In their

protocol, the prover computes $w = x^{(e+1)/2}$ and sends it to the verifier, who then accepts if and only if $w^2 = x \cdot y$. The idea is that if N is the product of two safe primes and $\mathsf{order}(y/x^e) = 2$, then $x \cdot y$ must be a quadratic non-residue modulo N. This is true since, as we prove, all group elements of order 2 in $\mathbb{Z}_N^*$, including y/x^e, are quadratic non-residues modulo N. Now observe that $x \cdot y = (y/x^e) \cdot x^{e+1}$. Since e is odd, x^{e+1} is a quadratic residue modulo N, and we conclude that $x \cdot y$ is a quadratic non-residue modulo N. This means that w^2, which is of course a quadratic residue modulo N, cannot be equal to $x \cdot y$ and the verifier will inevitably reject.

Generalizing this approach to arbitrary exponents is fairly straightforward, by having the prover compute w as $x^{\lceil (e+1)/2 \rceil}$ and then having the verifier check that $w^2 = x^{1+(e+1 \mod 2)} \cdot y$. The more acute issue is that when moving to the batch setting, the naïve way for verifying that $\mathsf{order}(y_i/x_i^e) \neq 2$ for every $i \in [n]$ is by running the above protocol n times in parallel, which results in communication complexity which is linear in n. To avoid this overhead, we combine the ideas of Di Crescenzo et al. with techniques from our Random Subset Compiler. Concretely, in our final protocol, the verifier chooses a key k to a pseudorandom function, to succinctly represent m random subsets $\mathcal{S}_1, \ldots, \mathcal{S}_m$ of $[n]$, and sends k to the prover. The prover then computes $w_j := \prod_{i \in \mathcal{S}_j} x_i^{\lceil (e+1)/2 \rceil}$ for every $j \in [m]$ and sends $w_1, \ldots, w_m$ to the verifier. Finally, the verifier computes $t_j := \prod_{i \in \mathcal{S}_j} x_i^{1+(e+1 \mod 2)} \cdot y_i$ for every $j \in [m]$ and accepts if and only if $t_j = w_j^2$ for all j-s. A careful analysis shows that if $\mathsf{order}(y_i/x_i^e) = 2$ for some $i \in [n]$ and the subsets $\mathcal{S}_1, \ldots, \mathcal{S}_m$ uniformly and independently at random, then each t_j is a quadratic non-residue with probability at least $1/2$. This implies that the verifier will accept with probability at most $2^{-m} + \mathsf{negl}(\lambda)$. We refer the reader to Section 7 for a detailed description of our protocol and a formal analysis of its soundness.

A statistically-sound protocol in safe RSA groups. Our statistically-sound proof of correct exponentiation in RSA groups is obtained by extending the protocol of Pietrzak [Pie19] using techniques similar to those detailed above. We start by recalling Pietrzak's protocol. Suppose that the prover wishes to convince the verifier that $y = x^{2^T}$ for a group $\mathbb{G}$, elements $x, y \in \mathbb{G}$ and an integer T, and assume for ease of presentation that $T = 2^t$ for some $t \in \mathbb{N}$. In the beginning of the protocol, the prover computes $z = x^{2^{T/2}}$ and sends z to the verifier. Now the prover wishes to prove to the verifier that indeed $z = x^{2^{T/2}}$ *and* $y = z^{2^{T/2}}$, since if $y \neq x^{2^T}$ then it must be that $z \neq x^{2^{T/2}}$ or $y \neq z^{2^{T/2}}$. One possibility is to recurse on both claims, until the exponent is small enough for the verifier to verify the claims herself. However, in this manner the number of sub-claims will blowup very quickly, resulting in a lengthy proof and in a long verification time. So instead, Pietrzak's idea is to merge both claims using (implicitly) the random exponents technique of Bellare, Garay and Rabin [BGR98]. The verifier samples a random integer $r \leftarrow [2^\lambda]$ and sends it to the prover, and then the two parties recurse on the (single) instance $(x' = x^r \cdot z, y' = z^r \cdot y, T' = T/2)$. That is, the prover now needs to convince the verifier of the claim $y' = (x')^{T'}$, where T' is half the size of T. Suppose now that all elements in the group are of order at least 2^λ.[9] In this case, if $y \neq x^{2^T}$ then there is at most one value of $r \in [2^\lambda]$ for which $y' = (x')^{T'}$ and hence $\Pr\left[y' = (x')^{T'}\right] \leq 2^{-\lambda}$ over the choice of r. This recursion continues for $\log T = t$ rounds until $T = 1$, in which case the verifier can simply check the relation $y = x^2$ herself using a single squaring in the group.

In order to extend this protocol to the group $\mathbb{Z}_N^*$ for a modulus N that is the product of two safe primes, we use similar techniques to those used above for extending the Random Exponents Compiler. Concretely, consider a round in Piertzak's protocol in which the prover wants to prove

[9] Recall that Boneh et al. [BBF18] proved computational soundness by extending this analysis to the case where there are elements of order less than 2^λ, but these are hard to find. We focus here on statistical soundness, as this is what we will obtain in safe RSA groups.

that $y = x^{2^T}$. In addition to $z = x^{2^{T/2}}$, the prover now also computes $w = x^{2^{T/2-1}+1}$ and sends it to the verifier. The verifier then checks that $x^2 \cdot z = w^2$, and if that is not the case then the verifier rejects immediately. This additional verification is made in each of the $\log T$ rounds of the protocol, and if the verifier does not reject in any of the rounds and the check for $T = 1$ goes through, then the verifier accepts.

To analyze the soundness of our protocol, suppose that all group elements (other than the identity) are either of order 2 or have order at least 2^λ; this is the case for $\mathbb{Z}_N^*$ where N is the product of two large enough safe primes. Let x and y be group elements such that $y \neq x^{2^T}$, and assume that z and w are the group elements which the prover sends to the verifier. If $x^2 \cdot z \neq w^2$, then the verifier will surely reject and we are done, so for the rest of the analysis we assume that $x^2 \cdot z = w^2$. Consider two cases:

- If $z = x^{2^{T/2}}$, then for any $r \in [2^\lambda]$ it holds that $(x^r \cdot z)^{2^{T/2}} = z^r \cdot x^{2^T} \neq z^r \cdot y$. In other words, $\Pr\left[y' = (x')^{T'}\right] = 0$, where $x' = x^r \cdot z$, $y' = z^r \cdot y$, $T' = T/2$ and the probability is taken over $r \leftarrow [2^\lambda]$.
- If $z \neq x^{2^{T/2}}$, then we prove that there is at most one value $r \in [2^\lambda]$ for which $(x^r \cdot z)^{2^{T/2}} = z^r \cdot y$. Assume towards contradiction otherwise; that is, that there are two *distinct* integers $r, r' \in [2^\lambda]$ for which this equality holds. Rearranging, this means that $(x^{2^{T/2}}/z)^d = 1$ for $d = r - r'$. Since $x^{2^{T/2}} \neq z$ and $d < 2^\lambda$, we obtain that the order of $x^{2^{T/2}}/z$ is greater than 1 and lesser than 2^λ, and hence this order must be 2. On the one hand, this means that $x^{2^{T/2}}/z$ is a quadratic non-residue modulo N (recall that all elements of order 2 in a safe RSA group are quadratic non-residues), and hence z is a quadratic non-residue modulo N. But on the other hand, our assumption that $x^2 \cdot z = w^2$ implies that z is a quadratic residue modulo N, arriving at a contradiction.

Over all, we obtain that in each round whose input (x, y, T) satisfies $y \neq x^{2^T}$, the probability that the input (x', y', T') to the next round will satisfy $y' = (x')^{T'}$ is at most $2^{-\lambda}$ over the choice of $r \leftarrow [2^\lambda]$. The soundness of our protocol then follows by taking a union bound over all rounds. We refer the reader to Section 8 for a detailed description of our protocol and a formal analysis of its soundness.

1.4 Paper Organization

The remainder of this paper is organized as follows. First, in Section 2 we present the basic notation, mathematical background and standard cryptographic primitives that are used throughout the paper. In Section 3 we formally define proofs of correct exponentiation and their batch variant. In Section 4 we present a simplified version of our Random Subsets Compiler for general groups; and then in Section 5 we present the necessary amendments required in order to obtain the full-fledged compiler. In Section 6 we present our improved compiler and analyze its security based on the low order assumption; and in Appendix A we give tighter security analyses for the specific cases of QR_N^+ and RSA groups. Finally, in Section 7 we show how to obtain strong soundness for our improved compiler in safe RSA groups; and in Section 8 we present our new proof of correct exponentiation in such groups.

2 Preliminaries

In this section we present the basic notions and standard cryptographic tools that are used in this work. For an integer $n \in \mathbb{N}$ we denote by $[n]$ the set $\{1, \ldots, n\}$. For a set $\mathcal{X}$, we denote by $2^{\mathcal{X}}$ the

power set of $\mathcal{X}$; i.e., the set which contains all subsets of $\mathcal{X}$ (including the empty set and $\mathcal{X}$ itself). For a distribution X we denote by $x \leftarrow X$ the process of sampling a value x from the distribution X. Similarly, for a set $\mathcal{X}$ we denote by $x \leftarrow \mathcal{X}$ the process of sampling a value x from the uniform distribution over $\mathcal{X}$. A function $\nu : \mathbb{N} \to \mathbb{R}^+$ is *negligible* if for any polynomial $p(\cdot)$ there exists an integer N such that for all $n > N$ it holds that $\nu(n) \leq 1/p(n)$.

Pseudorandom Functions. We use the following standard notion of a pseudorandom function. Let $\mathsf{PRF} = (\mathsf{PRF.Gen}, \mathsf{PRF.Eval})$ be a function family over domain $\{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ with range $\{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ and key space $\{\mathcal{K}_\lambda\}_{\lambda\in\mathbb{N}}$, such that:

- $\mathsf{PRF.Gen}$ is a probabilistic polynomial-time algorithm, which takes as input the security parameter $\lambda \in \mathbb{N}$ and outputs a key $K \in \mathcal{K}_\lambda$.
- $\mathsf{PRF.Eval}$ is a deterministic polynomial-time algorithm, which takes as input a key $K \in \mathcal{K}_\lambda$ and a domain element $x \in \mathcal{X}_\lambda$ and outputs a value $y \in \mathcal{Y}_\lambda$.

For ease of notation, for a key $K \in \mathcal{K}_\lambda$, we denote by $\mathsf{PRF}_K(\cdot)$ the function $\mathsf{PRF.Eval}(K, \cdot)$. We also assume without loss of generality that for every $\lambda \in \mathbb{N}$, it holds that $\mathcal{K}_\lambda = \{0,1\}^\lambda$ and that $\mathsf{PRF.Gen}(1^\lambda)$ simply samples K from $\{0,1\}^\lambda$ uniformly at random. Using these conventions, the following definition captures the standard notion of a pseudorandom function family.

Definition 2.1. A function family $\mathsf{PRF} = (\mathsf{PRF.Gen}, \mathsf{PRF.Eval})$ is *pseudorandom* if for every probabilistic polynomial-time algorithm D, there exists a negligible function $\nu(\cdot)$ such that

$$\mathsf{Adv}_{\mathsf{PRF},\mathsf{D}}(\lambda) \stackrel{\mathsf{def}}{=} \left| \Pr_{K\leftarrow\{0,1\}^\lambda}\left[\mathsf{D}(1^\lambda)^{\mathsf{PRF}_K(\cdot)} = 1\right] - \Pr_{f\leftarrow\mathcal{F}_\lambda}\left[\mathsf{D}(1^\lambda)^{f(\cdot)} = 1\right]\right| \leq \nu(\lambda),$$

for all sufficiently large $\lambda \in \mathbb{N}$, where $\mathcal{F}_\lambda$ is the set of all functions mapping $\mathcal{X}_\lambda$ into $\mathcal{Y}_\lambda$.

RSA groups and the factoring assumption. We will use to following formalization in order to reason about ensembles of RSA moduli and the hardness of finding their factorizations. Let ModGen be a probabilistic polynomial-time algorithm, which takes as input the security parameter $\lambda \in \mathbb{N}$, and outputs a bi-prime modulus $N = p \cdot q$ and possibly additional parameters pp.

Definition 2.2. The factoring assumption holds with respect to modulus generation algorithm ModGen if for every probabilistic polynomial time algorithm A, there exists a negligible function $\nu(\cdot)$ such that

$$\Pr\left[\begin{array}{l|l} p' \cdot q' = N & (N, \mathsf{pp}) \leftarrow \mathsf{ModGen}(1^\lambda) \\ p', q' \in \{2, \ldots, N-1\} & (p', q') \leftarrow \mathsf{A}(N, \mathsf{pp}) \end{array}\right] \leq \nu(\lambda),$$

for all sufficiently large $\lambda \in \mathbb{N}$.

The following simple lemma (see for example [Bon99]) states that it is easy to find a factorization of an RSA modulus N given a non-trivial square root of unity in the RSA group $\mathbb{Z}_N^*$.

Lemma 2.3. *There exist a deterministic algorithm* A*, such that for every pair* (p, q) *of primes and every group element* $x \in \mathbb{Z}_N^*$ *for which* $x^2 = 1$ *and* $x \notin \{1, -1\}$*, where* $N = p \cdot q$*, it holds that* $\mathsf{A}(N, x)$ *outputs* p *and* q*. Moreover,* A *runs in time polynomial in* $\log(N)$*.*

Using safe primes. We will sometimes focus on the case in which the RSA modulus N is the product of two *safe* primes. That is, $N = p' \cdot q'$, such that p' and q' are primes and there exist primes p and q for which $p' = 2p + 1$ and $q' = 2q + 1$. In this case, the order of the RSA group $\mathbb{Z}_N^*$ is $\varphi(N) = 4 \cdot p \cdot q$, where $\varphi(\cdot)$ is Euler's totient function. Looking ahead, this fact induces a relatively simple subgroup structure which will prove useful in Sections 7, 8 and A.2.

The group QR_N^+. Another group of interest in this work is the group QR_N of quadratic residues modulo N, where N is an RSA modulus generated by the modulus generation algorithm ModGen. This is the group defined by

$$QR_N \stackrel{\mathsf{def}}{=} \left\{x^2 \mod N \ : \ x \in \mathbb{Z}_N^*\right\}.$$

The order of the group QR_N is $\varphi(N)/4$. If N is the product of two safe primes $p' = 2p+1$ and $q' = 2q+1$, this means the the order of QR_N is $p \cdot q$.

We will also consider the group QR_N^+ of *signed* quadratic residues modulo N, defined by

$$QR_N^+ \stackrel{\mathsf{def}}{=} \{|x| \ : \ x \in QR_N\},$$

where the absolute value operator $|\cdot|$ is with respect to the representation of $\mathbb{Z}_N^*$ elements as elements in $\{-(N-1)/2, \ldots, (N-1)/2\}$. This is because membership in QR_N^+ can be decided in polynomial time[10] and we will implicitly use this fact in Section A.1. The map $|\cdot|$ acts as an isomorphism from QR_N to QR_N^+, and hence QR_N^+ is also of order $\varphi(N)/4$. For a more in-depth discussion on the use of QR_N^+ instead of QR_N see [FS00, HK09, Pie19].

Working over general groups. Some of the results in this paper are more general, and do not assume working over a specific group. In these cases, the algorithm ModGen will be replaced by a group generation algorithm GGen. This is a probabilistic polynomial-time algorithm which takes in as input the security parameter and outputs a description of a group $\mathbb{G}$, and possibly additional public parameters pp. All groups in this paper are assumed to be abelian and we will not note this explicitly hereinafter. We will also implicitly assume that for all groups considered in this paper, their group operation is implementable in time polynomial in the security parameter λ.

The low order assumption. We will rely on the following formalization of the low order assumption, put forth by Boneh et al. [BBF18] as a prerequesite for instantiating Pietrzak's protocol [Pie19] in general groups. For a group $\mathbb{G}$, let $1_{\mathbb{G}}$ denote the identity element of the group.

Definition 2.4. Let GGen be a group generation algorithm, and let $d = d(\lambda)$ be an integer function of the security parameter $\lambda \in \mathbb{N}$. We say that the d-low order assumption holds with respect to GGen if for every probabilistic polynomial-time algorithm A, there exists a negligible function $\nu(\cdot)$ such that

$$\mathsf{Adv}^{\mathsf{LowOrd}}_{\mathsf{GGen},d,\mathsf{A}}(\lambda) \stackrel{\mathsf{def}}{=} \Pr\left[\mathsf{LowOrd}^{\mathsf{GGen}}_{d,\mathsf{A}}(\lambda) = 1\right] \leq \nu(\lambda)$$

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

1. $\mathbb{G} \leftarrow \mathsf{GGen}(1^\lambda)$.
2. $(x, \omega) \leftarrow \mathsf{A}(\mathbb{G})$.
3. Output 1 if $x \neq 1_{\mathbb{G}}$; $\omega < d$; and $x^\omega = 1_{\mathbb{G}}$. Otherwise, output 0.

Pietrzak observed (although not in this terminology) that the d-low order assumption holds information-theoretically in the group QR_N^+, whenever N is the product of two safe primes $p' = 2p+1$ and $q' = 2q+1$, and $d \leq \min\{p, q\}$.

[10] This is the case since, as observed by Fischlin and Schnorr [FS00], $QR_N^+ = \mathbb{J}_N^+$, where $\mathbb{J}_N$ is the group of elements with Jacobi symbol $+1$ and $\mathbb{J}_N^+ \stackrel{\mathsf{def}}{=} \mathbb{J}_N / \pm 1$. Hence, deciding whether an integer x is in QR_N^+ amounts to checking that $x \geq 0$ and that its Jacobi symbol is $+1$.

In cases where $\mathbb{G}$ is naturally embedded in some ring R and $-1_{\mathbb{G}} \in \mathbb{G}$ (that is, the additive inverse of the multiplicative identity is an element of the group),[11] we can consider a weakening of Definition 2.4, requiring that the adversary is unable to come up with a low order element other than $\pm 1_{\mathbb{G}}$.

Definition 2.5. Let GGen be a group generation algorithm, and let $d = d(\lambda)$ be an integer function of the security parameter $\lambda \in \mathbb{N}$. We say that the *weak* d-low order assumption holds with respect to GGen if for every probabilistic polynomial-time algorithm A, there exists a negligible function $\nu(\cdot)$ such that

$$\mathsf{Adv}^{\mathsf{WeakLO}}_{\mathsf{GGen},d,\mathsf{A}}(\lambda) \stackrel{\mathsf{def}}{=} \Pr\left[\mathsf{WeakLO}^{\mathsf{GGen}}_{d,\mathsf{A}}(\lambda) = 1\right] \leq \nu(\lambda)$$

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

1. $\mathbb{G} \leftarrow \mathsf{GGen}(1^\lambda)$.
2. $(x, \omega) \leftarrow \mathsf{A}(\mathbb{G})$.
3. Output 1 if $x \notin \{1_{\mathbb{G}}, -1_{\mathbb{G}}\}$; $\omega < d$; and $x^\omega = 1_{\mathbb{G}}$. Otherwise, output 0.

Seres and Burcsi [SB20] recently proved (as a special case) that in RSA groups with a modulus N which is the product of two safe primes $p' = 2p+1$ and $q' = 2q+1$, the weak d-low order assumption for $d \leq \min\{p, q\}$ is equivalent to factoring N.

3 Succinct Proofs of Correct Exponentiation

In this section we review the notion of succinct proofs of correct exponentiation. First, in Section 3.1, we define proofs of correct exponentiation for a single instance and then, in Section 3.2, we extend the definition to account for the task of batch verification.

3.1 The Basic Definition

Loosely speaking, a proof of correct exponentiation is a protocol executed by two parties, a prover and a verifier, with a common input (x, y, e), where x and y are elements in some group $\mathbb{G}$ and e is an integer. The goal of the prover is to convince the verifier that $y = x^e$. Of course, the verifier can just compute x^e and compare the result to y on her own, but we will be interested in protocols in which the verifier works much less than that. Concretely, we are typically interested in protocols in which the the verifier runs in time $\ll \mathsf{poly}(\log(e), \lambda)$, which is the time it will take the verifier to compute x^e on her own, assuming that the group operation is implementable in time polynomial in the security parameter $\lambda \in \mathbb{N}$.

More formally, a proof of correct exponentiation (PoCE) is a triplet $\pi = (\mathsf{GGen}, \mathsf{P}, \mathsf{V})$ of probabilistic polynomial-time algorithms, where GGen is a group generation algorithm (recall Section 2), P is the prover and V is the verifier. We denote by $\langle \mathsf{P}(\mathsf{aux}), \mathsf{V} \rangle (\mathsf{input})$ the random variable corresponding to the output of V when the joint input to P and to V is input and P additionally receives the private auxiliary information aux. In case P receives no auxiliary information, we write $\langle \mathsf{P}, \mathsf{V} \rangle (\mathsf{input})$. The properties which should be satisfied by a PoCE are defined in the following definition.

Definition 3.1. Let $\delta = \delta(\lambda)$ be a function of the security parameter $\lambda \in \mathbb{N}$, and let $t = t(\lambda, e)$ and $c = c(\lambda, e)$ be functions of $\lambda \in \mathbb{N}$ and of the exponent $e \in \mathbb{N}$. A triplet $\pi = (\mathsf{GGen}, \mathsf{P}, \mathsf{V})$ of probabilistic polynomial-time algorithms is said to be a (δ, c, t)-proof of correct exponentiation (PoCE) if the following conditions hold:

[11] This is indeed the case for RSA groups, which are embedded in the ring $\mathbb{Z}_N$.

1. **Completeness:** For every $\lambda \in \mathbb{N}$, for every $(\mathbb{G}, \mathsf{pp})$ in the support of $\mathsf{GGen}(1^\lambda)$ and for every input $(x, y, e) \in \mathbb{G}^2 \times \mathbb{N}$ such that $x^e = y$, it holds that

$$\Pr\left[\langle \mathsf{P}, \mathsf{V}\rangle\left(\mathbb{G}, \mathsf{pp}, x, y, e\right) = 1\right] = 1,$$

where the probability is over the randomness of P and of V.

2. **δ-Soundness:** For every pair $\mathsf{P}^* = (\mathsf{P}_1^*, \mathsf{P}_2^*)$ of probabilistic polynomial-time algorithms, there exists a negligible function $\nu(\cdot)$ that

$$\mathsf{Adv}^{\mathsf{PoCE}}_{\pi,\mathsf{P}^*}(\lambda) \stackrel{\mathsf{def}}{=} \Pr\left[\begin{array}{l|l} \langle \mathsf{P}_2^*(\mathsf{st}), \mathsf{V}\rangle\left(\mathbb{G}, \mathsf{pp}, x, y, e\right) = 1 & (\mathbb{G}, \mathsf{pp}) \leftarrow \mathsf{GGen}(1^\lambda) \\ x^e \neq y & (x, y, e, \mathsf{st}) \leftarrow \mathsf{P}_1^*(\mathbb{G}, \mathsf{pp}) \end{array}\right] \leq \delta(\lambda) + \nu(\lambda)$$

for all sufficiently large $\lambda \in \mathbb{N}$. We will sometimes write $\mathsf{Adv}^{\mathsf{PoCE}}_{\pi,\mathsf{P}^*}$ instead of $\mathsf{Adv}^{\mathsf{PoCE}}_{\pi,\mathsf{P}^*}(\lambda)$ when there is no ambiguity.

3. **Succinctness:** For every $\lambda \in \mathbb{N}$, for every $(\mathbb{G}, \mathsf{pp})$ in the support of $\mathsf{GGen}(1^\lambda)$ and for every input $(x, y, e) \in \mathbb{G}^2 \times \mathbb{N}$, it holds that: The total length of all messages exchanged between P and V in a random execution of the protocol on joint input $(\mathbb{G}, \mathsf{pp}, x, y, e)$ is at most $c(\lambda, e)$ with probability 1, where the probability is over the randomness of P and of V.

4. **Efficient verification:** For every $\lambda \in \mathbb{N}$, for every $(\mathbb{G}, \mathsf{pp})$ in the support of $\mathsf{GGen}(1^\lambda)$ and for every input $(x, y, e) \in \mathbb{G}^2 \times \mathbb{N}$, it holds that: The running time of V in a random execution of the protocol on joint input $(\mathbb{G}, \mathsf{pp}, x, y, e)$ is at most $t(\lambda, e)$ with probability 1, where the probability is over the randomness of P and of V.

3.2 Batch Proofs of Correct Exponentiation

We now turn to define *batch* proofs of correct exponentiation. In such proofs, the joint input is composed of $2n$ group elements $x_1, \ldots, x_n, y_1, \ldots, y_n$ and an exponent e, for some $n \in \mathbb{N}$. The prover now wishes to convince the verifier that $x_i^e = y_i$ for each of the $i \in [n]$. The definition is a natural extension of Definition 3.1, except that now the communication complexity and the running time of the verifier may both scale with the integer n. It might also make sense to consider the case where the soundness error δ is also a function of n, but this will not be the case in our protocols, and hence we do not account for this case in our definition. The formal definition below uses the same notation as did Definition 3.1.

Definition 3.2. Let $\delta = \delta(\lambda)$ be a function of the security parameter $\lambda \in \mathbb{N}$, and let $t = t(\lambda, e, n)$ and $c = c(\lambda, e, n)$ be function of λ, of the exponent $e \in \mathbb{N}$ and of $n \in \mathbb{N}$. A triplet $\pi = (\mathsf{GGen}, \mathsf{P}, \mathsf{V})$ of probabilistic polynomial-time algorithms is said to be a (δ, c, t)-batch proof of correct exponentiation (BPoCE) if the following conditions hold:

1. **Completeness:** For every integers $\lambda, n \in \mathbb{N}$, every $(\mathbb{G}, \mathsf{pp})$ in the support of $\mathsf{GGen}(1^\lambda)$ and every input $(\vec{x} = (x_1, \ldots, x_n), \vec{y} = (y_1, \ldots, y_n), e) \in \mathbb{G}^n \times \mathbb{G}^n \times \mathbb{N}$ such that $x_i^e = y_i$ for every $i \in [n]$, it holds that

$$\Pr\left[\langle \mathsf{P}, \mathsf{V}\rangle\left(\mathbb{G}, \mathsf{pp}, \vec{x}, \vec{y}, e\right) = 1\right] = 1,$$

where the probability is over the randomness of P and of V.

2. **δ-Soundness:** For every pair $\mathsf{P}^* = (\mathsf{P}_1^*, \mathsf{P}_2^*)$ of probabilistic polynomial-time algorithms, there exists a negligible function $\nu(\cdot)$ such that

$$\mathsf{Adv}^{\mathsf{BPoCE}}_{\pi,\mathsf{P}^*}(\lambda) \stackrel{\mathsf{def}}{=} \Pr\left[\begin{array}{l|l} \langle \mathsf{P}_2^*(\mathsf{st}), \mathsf{V}\rangle\left(\mathbb{G}, \mathsf{pp}, \vec{x}, \vec{y}, e\right) = 1 & (\mathbb{G}, \mathsf{pp}) \leftarrow \mathsf{GGen}(1^\lambda) \\ \exists i \in [n], x_i^e \neq y_i & (n, \vec{x}, \vec{y}, e, \mathsf{st}) \leftarrow \mathsf{P}_1^*(\mathbb{G}, \mathsf{pp}) \end{array}\right] \leq \delta(\lambda) + \nu(\lambda)$$

for all sufficiently large $\lambda \in \mathbb{N}$, where $\vec{x} = (x_1, \ldots, x_n)$ and $\vec{y} = (y_1, \ldots, y_n)$. We will sometimes write $\mathsf{Adv}^{\mathsf{BPoCE}}_{\pi,\mathsf{P}^*}$ instead of $\mathsf{Adv}^{\mathsf{BPoCE}}_{\pi,\mathsf{P}^*}(\lambda)$ when there is no ambiguity.

3. **Succinctness:** For every $\lambda, n \in \mathbb{N}$, for every $(\mathbb{G}, \mathsf{pp})$ in the support of $\mathsf{GGen}(1^\lambda)$ and for every input $(\vec{x}, \vec{y}, e) \in \mathbb{G}^n \times \mathbb{G}^n \times \mathbb{N}$, it holds that: The total length of all messages exchanged between P and V in a random execution of the protocol on joint input $(\mathbb{G}, \mathsf{pp}, \vec{x}, \vec{y}, e)$ is at most $c(\lambda, e, n)$ with probability 1, where the probability is over the randomness of P and of V.

4. **Efficient verification:** For every $\lambda, n \in \mathbb{N}$, for every $(\mathbb{G}, \mathsf{pp})$ in the support of $\mathsf{GGen}(1^\lambda)$ and for every input $(\vec{x}, \vec{y}, e) \in \mathbb{G}^n \times \mathbb{G}^n \times \mathbb{N}$, it holds that: The running time of V in a random execution of the protocol on joint input $(\mathbb{G}, \mathsf{pp}, \vec{x}, \vec{y}, e)$ is at most $t(\lambda, e, n)$ with probability 1, where the probability is over the randomness of P and of V.

On using a single exponent. The above definition considers the setting of a single exponent for all n pairs of group elements; that is, the joint input includes a single exponent $e \in \mathbb{N}$ for which the prover contends that $x_i^e = y_i$ for all $i \in [n]$. Note that this setting is indeed in line with the motivation described in Section 1 of batch verification of many VDF outputs based on the repeated squaring function. This is the case, since in this scenario the exponent e is determined by the delay parameter T. In the examples mentioned in Section 1, a scenario in which all outputs were computed with respect to the same delay parameter is reasonable. It might still be of interest, both theoretically and for specific applications (see for example [BBF19]), to construct batch proofs of correct exponentiation with different exponents, and we leave it is as an interesting open question.

4 Warm-Up: The Random Subset Compiler

In this section we present a simplified version of our general compiler, which we call "The Random Subset Compiler" following Bellare, Garay and Rabin [BGR98]. This simplified version is based on a technique introduced by Bellare et al. for related, yet distinct, purposes (recall Section 1.2). In our context of proofs of correct exponentiation, this technique introduces quite a large communication overhead and a considerable amount of additional soundness error. Nevertheless, we start off with this simplified version as it already captures the main ideas behind the full-fledged compiler. Then, in Section 5 we show how to simultaneously amplify the soundness guarantees of our compiler and considerably reducing the communication overhead.

Let $\delta = \delta(\lambda)$ be a function of the security parameter $\lambda \in \mathbb{N}$, and let $c = c(\lambda, e)$ and $t = t(\lambda, e)$ be functions of λ and of the exponent $e \in \mathbb{N}$. Our compiler uses as a building block any (δ, c, t)-PoCE (recall Definition 3.1) $\pi = (\mathsf{GGen}, \mathsf{P}, \mathsf{V})$ and produces a protocol $\mathsf{Batch}_1(\pi) = (\mathsf{GGen}, \mathsf{P}_{\mathsf{Batch}}, \mathsf{V}_{\mathsf{Batch}})$, which is a (δ', c', t')-BPoCE for $\delta' = \delta + 1/2$ and for related functions $c' = c'(\lambda, e, n)$ and $t' = t'(\lambda, e, n)$.

The Protocol $\mathsf{Batch}_1(\pi)$

Joint input: Public parameters $(\mathbb{G}, \mathsf{pp})$ generated by $\mathsf{GGen}(1^\lambda)$, vectors $\vec{x} = (x_1, \ldots, x_n)$ and $\vec{y} = (y_1, \ldots, y_n)$ of group elements, and an exponent $e \in \mathbb{N}$.

1. $\mathsf{V}_{\mathsf{Batch}}$ samples $\mathcal{I} \leftarrow 2^{[n]}$ and sends $\mathcal{I}$ to $\mathsf{P}_{\mathsf{Batch}}$.
2. Both $\mathsf{V}_{\mathsf{Batch}}$ and $\mathsf{P}_{\mathsf{Batch}}$ compute $x_{\mathsf{prod}} = \prod_{i \in \mathcal{I}} x_i$ and $y_{\mathsf{prod}} = \prod_{i \in \mathcal{I}} y_i$.
3. $\mathsf{V}_{\mathsf{Batch}}$ and $\mathsf{P}_{\mathsf{Batch}}$ execute the protocol π on joint input $(\mathbb{G}, \mathsf{pp}, x_{\mathsf{prod}}, y_{\mathsf{prod}}, e)$, where $\mathsf{V}_{\mathsf{Batch}}$ plays the role of V and $\mathsf{P}_{\mathsf{Batch}}$ plays the role of P. Let $b \in \{0, 1\}$ be the output of V in this execution.
4. $\mathsf{V}_{\mathsf{Batch}}$ outputs b.

Theorem 4.1 below establishes the completeness, soundness, succinctness and verifier efficiency of $\mathsf{Batch}_1(\pi)$. It uses the following notation: We let $t_{\mathsf{op}}(\lambda)$ denote a bound on the time required to apply the binary group operation on two group elements, in a group $\mathbb{G}$ generated by $\mathsf{GGen}(1^\lambda)$.

Theorem 4.1. *Assume that π is a (δ, c, t)-PoCE. Then, $\mathsf{Batch}_1(\pi)$ is a (δ', c', t')-BPoCE, where for every $\lambda, e, n \in \mathbb{N}$:*

- $\delta'(\lambda) = \delta(\lambda) + 1/2$.
- $c'(\lambda, n, e) = c(\lambda, e) + n$.
- $t'(\lambda, n, e) = t(\lambda, e) + O(n \cdot t_{\mathsf{op}}(\lambda))$.

We start by presenting our main technical lemma, which we will use in the proof of Theorem 4.1 as well as in subsequent sections.

Lemma 4.2. *Let $\mathbb{G}$ be a group. For every integers $n, e \in \mathbb{N}$ and vectors $\vec{x}, \vec{y} \in \mathbb{G}^n$ the following holds: If there exists an index $i \in [n]$ such that $x_i^e \neq y_i$, then*

$$\Pr_{\mathcal{I} \leftarrow 2^{[n]}} \left[\left(\prod_{i \in \mathcal{I}} x_i \right)^e \neq \prod_{i \in \mathcal{I}} y_i \right] \geq \frac{1}{2}.$$

Proof of Lemma 4.2. For a subset $\mathcal{I} \subseteq [n]$, we say that $\mathcal{I}$ is *biased* if $\left(\prod_{i \in \mathcal{I}} x_i\right)^e \neq \prod_{i \in \mathcal{I}} y_i$, and otherwise we say that $\mathcal{I}$ is *balanced*. Denote by $\mathcal{S}_{\mathsf{Balanced}}$ and by $\mathcal{S}_{\mathsf{Biased}}$ the set of all balanced subsets of $[n]$ and the set of all biased subsets of $[n]$, respectively.

Suppose that there exists an index $i \in [n]$ such that $x_i^e \neq y_i$, and let i^* be an arbitrary such index (e.g., the minimal index for which the inequality holds). We wish to show that $|\mathcal{S}_{\mathsf{Balanced}}| \leq |\mathcal{S}_{\mathsf{Biased}}|$, as this will conclude the proof of the lemma. To this end, consider a partition $\mathcal{P}$ of $2^{[n]}$ to 2^{n-1} pairs as follows:

$$\mathcal{P} = \left\{ (\mathcal{I}, \mathcal{I} \cup \{i^*\}) \ : \ i^* \notin \mathcal{I} \right\}.$$

In each pair $(\mathcal{I}, \mathcal{I} \cup \{i^*\})$ in $\mathcal{P}$, at most one subset of $\mathcal{I}$ and $\mathcal{I} \cup \{i^*\}$ can be balanced. This is the case since if $\mathcal{I} \in \mathcal{S}_{\mathsf{Balanced}}$, then it must be that $\mathcal{I} \cup \{i^*\} \in \mathcal{S}_{\mathsf{Biased}}$ since

$$\begin{aligned}
\prod_{i \in \mathcal{I} \cup \{i^*\}} x_i^e &= (x_i^*)^e \cdot \prod_{i \in \mathcal{I}} x_i^e \\
&= (x_i^*)^e \cdot \prod_{i \in \mathcal{I}} y_i && (4.1) \\
&\neq y_{i^*} \cdot \prod_{i \in \mathcal{I}} y_i && (4.2) \\
&= \prod_{i \in f(\mathcal{I})} y_i,
\end{aligned}$$

where Eq. (4.1) holds because $\mathcal{I}$ is balanced, and (4.2) holds due to the assumption that $x_{i^*}^e \neq y_{i^*}$. Since at most one subset in each pair of the 2^{n-1} pairs in $\mathcal{P}$ is balanced, it holds that $|\mathcal{S}_{\mathsf{Balanced}}| \leq |\mathcal{S}_{\mathsf{Biased}}|$ concluding the proof of Lemma 4.2. ■

We are now ready to prove Theorem 4.1, establishing the completeness, soundness, verifier efficiency and succinctness of our protocol π_{Batch}.

Proof of Theorem 4.1. Completeness follows immediately from the completeness of π and the fact that if $x_i^e = y_i$ for every $i \in [n]$, then it also holds that $\left(\prod_{i\in\mathcal{I}} x_i\right)^e = \prod_{i\in\mathcal{I}} y_i$ for any choice of $\mathcal{I} \in 2^{[n]}$.

We now turn to prove that $\mathsf{Batch}_1(\pi)$ satisfies δ'-soundness for $\delta' = \delta + 1/2$. Let $\mathsf{P}^*_{\mathsf{Batch}} = (\mathsf{P}^*_{\mathsf{Batch},1}, \mathsf{P}^*_{\mathsf{Batch},2})$ be a malicious prover attempting to break the soundness of $\mathsf{Batch}_1(\pi)$. Consider the following pair $\mathsf{P}^* = (\mathsf{P}^*_1, \mathsf{P}^*_2)$ attempting to break the soundness of π. On input $(\mathbb{G}, \mathsf{pp})$ generated by $\mathsf{GGen}(1^\lambda)$, the algorithm P^*_1 is defined as follows:

1. Invoke $(n, \vec{x}, \vec{y}, e, \mathsf{st}) \leftarrow \mathsf{P}^*_{\mathsf{Batch},1}(\mathbb{G}, \mathsf{pp})$, where $\vec{x} = (x_1, \ldots, x_n)$ and $\vec{y} = (y_1, \ldots, y_n)$.
2. Sample $\mathcal{I} \leftarrow 2^{[n]}$.
3. Compute $x_{\mathsf{prod}} = \prod_{i\in\mathcal{I}} x_i$ and $y_{\mathsf{prod}} = \prod_{i\in\mathcal{I}} y_i$.
4. Set $\mathsf{st}' = (\mathsf{st}, \vec{x}, \vec{y}, \mathcal{I})$.
5. Output $(x_{\mathsf{prod}}, y_{\mathsf{prod}}, e, \mathsf{st}')$.

Then, the algorithm P^*_2, running on private input st' and interacting with the verifier V on joint input $(\mathbb{G}, \mathsf{pp}, x_{\mathsf{prod}}, y_{\mathsf{prod}}, e)$, is defined as follows:

1. Parse st' as $(\mathsf{st}, \vec{x}, \vec{y}, \mathcal{I})$.
2. Invoke $\mathsf{P}^*_{\mathsf{Batch},2}$ on input st and simulate to $\mathsf{P}^*_{\mathsf{Batch},2}$ an execution of $\mathsf{Batch}_1(\pi)$ on joint input $(\mathbb{G}, \mathsf{pp}, \vec{x}, \vec{y}, e)$:
   (a) Send $\mathcal{I}$ to $\mathsf{P}^*_{\mathsf{Batch},2}$ as the first message of the verifier in $\mathsf{Batch}_1(\pi)$.
   (b) Let V play the role of $\mathsf{V}_{\mathsf{Batch}}$ in all subsequent rounds, by relaying all messages from $\mathsf{P}^*_{\mathsf{Batch},2}$ to V and vice versa.

We now turn to bound $\mathsf{Adv}^{\mathsf{PoCE}}_{\pi,\mathsf{P}^*}$. To that end, we define the following events:

- Let $\mathsf{NotEqual}$ denote the event in which for some $i \in [n]$, it holds that $x_i^e \neq y_i$, where $n, \vec{x}, \vec{y}$ and e are outputted by $\mathsf{P}^*_{\mathsf{Batch},1}$ in Step 1 of P^*_1, $\vec{x} = (x_1, \ldots, x_n)$ and $\vec{y} = (y_1, \ldots, y_n)$.
- Let $\mathsf{BiasedSet}$ be the event in which
$$\left(\prod_{i\in\mathcal{I}} x_i\right)^e \neq \prod_{i\in\mathcal{I}} y_i,$$
where $n, \vec{x}, \vec{y}$ and e are as before and $\mathcal{I}$ is the random subset sampled by P^*_1 in Step 2.
- Let PWin be the event in which P^* wins; that is, V outputs 1 and $\mathsf{BiasedSet}$ holds.
- Let $\mathsf{PBatchWin}$ be the event in which $\mathsf{P}^*_{\mathsf{Batch}}$ wins in the simulation of $\mathsf{Batch}_1(\pi)$ by P^*: The simulated $\mathsf{V}_{\mathsf{Batch}}$ outputs 1 and $\mathsf{NotEqual}$ holds.

Equipped with this notation, it holds that

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{PoCE}}_{\pi,\mathsf{P}^*} &= \Pr\left[\mathsf{PWin}\right] \\
&\geq \Pr\left[\mathsf{PWin} | \mathsf{PBatchWin} \wedge \mathsf{BiasedSet}\right] \cdot \Pr\left[\mathsf{PBatchWin} \wedge \mathsf{BiasedSet}\right] \\
&= \Pr\left[\mathsf{PBatchWin} \wedge \mathsf{BiasedSet}\right], \qquad (4.3)
\end{aligned}$$

where Eq. (4.3) holds since conditioned on $\mathsf{PBatchWin}$, it holds that $\mathsf{V}_{\mathsf{Batch}}$ (in the simulation of $\mathsf{Batch}_1(\pi)$) outputs 1. This implies that V outputs 1, and hence that $\Pr\left[\mathsf{PWin} | \mathsf{PBatchWin} \wedge \mathsf{BiasedSet}\right] = 1$. By total probability,

$$\begin{aligned}
\Pr\left[\mathsf{PBatchWin} \wedge \mathsf{BiasedSet}\right] &= \Pr\left[\mathsf{PBatchWin}\right] - \Pr\left[\mathsf{PBatchWin} \wedge \overline{\mathsf{BiasedSet}}\right] \\
&= \mathsf{Adv}^{\mathsf{BPoCE}}_{\mathsf{Batch}_1(\pi),\mathsf{P}^*_{\mathsf{Batch}}} - \Pr\left[\mathsf{PBatchWin} \wedge \overline{\mathsf{BiasedSet}}\right] \qquad (4.4) \\
&\geq \mathsf{Adv}^{\mathsf{BPoCE}}_{\mathsf{Batch}_1(\pi),\mathsf{P}^*_{\mathsf{Batch}}} - \Pr\left[\mathsf{NotEqual} \wedge \overline{\mathsf{BiasedSet}}\right] \qquad (4.5) \\
&\geq \mathsf{Adv}^{\mathsf{BPoCE}}_{\mathsf{Batch}_1(\pi),\mathsf{P}^*_{\mathsf{Batch}}} - \Pr\left[\overline{\mathsf{BiasedSet}} \middle| \mathsf{NotEqual}\right], \qquad (4.6)
\end{aligned}$$

where Eq. (4.4) holds since P^* perfectly simulates $\mathsf{Batch}_1(\pi)$ to $\overline{\mathsf{P}^*_{\mathsf{Batch}}}$; and Eq. (4.5) is true since $\mathsf{PBatchWin}$ is contained in $\mathsf{NotEqual}$, and hence $\mathsf{PBatchWin} \wedge \overline{\mathsf{BiasedSet}}$ is contained in $\mathsf{NotEqual} \wedge \overline{\mathsf{BiasedSet}}$.

We are left with bounding $\Pr\left[\overline{\mathsf{BiasedSet}} \middle| \mathsf{NotEqual}\right]$. Indeed, Lemma 4.2 immediately implies that

$$\Pr\left[\overline{\mathsf{BiasedSet}} \middle| \mathsf{NotEqual}\right] \leq \frac{1}{2}. \tag{4.7}$$

Taking Eq. (4.3), (4.6) and (4.7) together and rearranging, we get that

$$\mathsf{Adv}^{\mathsf{BPoCE}}_{\mathsf{Batch}_1(\pi),\mathsf{P}^*_{\mathsf{Batch}}} \leq \mathsf{Adv}^{\mathsf{PoCE}}_{\pi,\mathsf{P}^*} + \frac{1}{2},$$

which implies – since π satisfies δ-soundness – that there exists a negligible function $\nu(\cdot)$ such that

$$\mathsf{Adv}^{\mathsf{BPoCE}}_{\mathsf{Batch}_1(\pi),\mathsf{P}^*_{\mathsf{Batch}}} \leq \delta(\lambda, e) + \frac{1}{2} + \nu(\lambda),$$

for all sufficiently large $\lambda \in \mathbb{N}$.

We have proved that $\mathsf{Batch}_1(\pi)$ satisfies δ'-soundness for $\delta' = \delta + 1/2$. To conclude the proof, we are left with bounding the verifier's running time t' and the communication complexity c' of $\mathsf{Batch}_1(\pi)$. As for the running time of $\mathsf{V}_{\mathsf{Batch}}$: The verifier samples a random subset $\mathcal{I}$, computes $\prod_{i\in\mathcal{I}} x_i$ and $\prod_{i\in\mathcal{I}} y_i$ and participates in a single execution of π. Since the products computed by $\mathsf{V}_{\mathsf{Batch}}$ include at most n group elements each, it follows that her running time is $t' = O(n \cdot t_{\mathsf{op}}(\lambda)) + t$. The communication in $\mathsf{Batch}_1(\pi)$ includes the subset $\mathcal{I}$, which can be encoded using n bits, and all messages exchanged in a single execution of π. Therefore, the total communication is $c' = c + n$. This concludes the proof of Theorem 4.1. ■

5 Amplifying Soundness and Reducing Communication

In this section, we address the two main drawbacks of the compiler from Section 4; namely, its large soundness error, and the fact that the communication complexity is linearly dependent on the number n of pairs (x_i, y_i). In order to do so, we introduce an improved compiler, which differs from the one found in Section 4 in two respects. First, the verifier now chooses m random subsets $\mathcal{I}_1, \ldots, \mathcal{I}_m \subseteq [n]$ for some integer m which is a parameter of the protocol, and the parties invoke m parallel executions of the underlying protocol π on the m inputs which are induced by these subsets. Note that sending the representation of m random subsets of $[n]$ requires that the verifier sends additional $m \cdot n$ bits to the prover. To avoid this communication overhead, we let the verifier succinctly represent these m subsets via a single key to pseudorandom function, thus reducing the additive communication overhead to just λ bits, where $\lambda \in \mathbb{N}$ is the security parameter. As we will show, this essentially does not harm the soundness guaranteed by the protocol.

We now turn to formally present our compiler. Let $\delta = \delta(\lambda)$ a be a function of the security parameter $\lambda \in \mathbb{N}$, and let $c = c(\lambda, e)$ and $t = t(\lambda, e)$ be functions of λ and of the exponent $e \in \mathbb{N}$. Our compiler is parameterized by an integer $m \in \mathbb{N}$, and uses the following building blocks:

- A (δ, c, t)-PoCE $\pi = (\mathsf{GGen}, \mathsf{P}, \mathsf{V})$ (recall Definition 3.1).
- A pseudorandom function family PRF, such that for every $\lambda \in \mathbb{N}$ and for every key $K \in \{0,1\}^\lambda$, the function PRF_K maps inputs in $\{0,1\}^{\lceil \log(m \cdot n) \rceil}$ to outputs in $\{0,1\}$. For ease of notation, for integers $j \in [m]$ and $i \in [n]$, we will write $\mathsf{PRF}_K(j \| i)$, with the intention (but without noting it explicitly) that the input to the function is a bit string representing the integers j and i.

The compiler produces a protocol $\mathsf{Batch}^m_2(\pi) = (\mathsf{GGen}, \mathsf{P}_{\mathsf{Batch}}, \mathsf{V}_{\mathsf{Batch}})$, which is a (δ', c', t')-BPoCE for $\delta' = \delta + 2^{-m}$ and for related functions $c' = c'(\lambda, e, n)$ and $t' = t'(\lambda, e, n)$.

The Protocol $\mathsf{Batch}_2^m(\pi)$

Joint input: Public parameters $(\mathbb{G}, \mathsf{pp})$ generated by $\mathsf{GGen}(1^\lambda)$, vectors $\vec{x} = (x_1, \ldots, x_n)$ and $\vec{y} = (y_1, \ldots, y_n)$ of $\mathbb{G}$ elements, and an exponent $e \in \mathbb{N}$.

1. $\mathsf{V}_{\mathsf{Batch}}$ samples $K \leftarrow \{0,1\}^\lambda$ and sends K to $\mathsf{P}_{\mathsf{Batch}}$.
2. For $j = 1, \ldots, m$:
   (a) Both $\mathsf{V}_{\mathsf{Batch}}$ and $\mathsf{P}_{\mathsf{Batch}}$ compute $\mathcal{I}_j = \{i \in [n] \ : \ \mathsf{PRF}_K(j \| i) = 1\}$.
   (b) Both $\mathsf{V}_{\mathsf{Batch}}$ and $\mathsf{P}_{\mathsf{Batch}}$ compute $u_j = \prod_{i \in \mathcal{I}_j} x_i$ and $w_j = \prod_{i \in \mathcal{I}_j} y_i$.
3. $\mathsf{V}_{\mathsf{Batch}}$ and $\mathsf{P}_{\mathsf{Batch}}$ execute in parallel m executions of the protocol π, where in the j-th execution, the joint input is $(\mathsf{pp}, u_j, w_j, e)$. In each execution, $\mathsf{V}_{\mathsf{Batch}}$ plays the role of V and $\mathsf{P}_{\mathsf{Batch}}$ plays the role of P. For each $j \in [m]$, let $b_j \in \{0,1\}$ be the output of V in the j-th execution.
4. $\mathsf{V}_{\mathsf{Batch}}$ outputs $b := \bigwedge_{j=1}^m b_j$.

Theorem 5.1 establishes the completeness, soundness, succinctness and verifier efficiency of $\mathsf{Batch}_2^m(\pi)$. Recall that we denote by $t_{\mathsf{op}} = t_{\mathsf{op}}(\lambda)$ the time required to apply the binary group operation on two group elements, in a group $\mathbb{G}$ generated by $\mathsf{GGen}(1^\lambda)$. We also denote by $t_{\mathsf{prf}} = t_{\mathsf{prf}}(\lambda, m, n)$ the time required to compute $\mathsf{PRF}_K(z)$ for $K \in \{0,1\}^\lambda$ and $z \in \{0,1\}^{\lceil \log(m \cdot n) \rceil}$.

Theorem 5.1. *Assume that* PRF *is a pseudorandom function and that* π *is a* (δ, c, t)*-PoCE. Then,* $\mathsf{Batch}_2^m(\pi)$ *is a* (δ', c', t')*-BPoCE, where:*

- $\delta'(\lambda) = \delta(\lambda) + 2^{-m}$.
- $c'(\lambda, n, e) = m \cdot c(\lambda, e) + \lambda$.
- $t'(\lambda, n, e) = m \cdot t(\lambda, e) + \lambda + O\left(m \cdot n \cdot (t_{\mathsf{op}} + t_{\mathsf{prf}})\right)$.

Before proving Theorem 5.1, a couple of remarks are in order.

Applying the Fiat-Shamir heuristic. If the Fiat-Shamir heuristic [FS86] can be applied to π in the random oracle model, then it can also be applied to $\mathsf{Batch}_2^m(\pi)$ as well, as long as $m = \omega(\log(\lambda))$ (and hence 2^{-m} is a negligible function of the security parameter $\lambda \in \mathbb{N}$). This is the case since our compiler only adds a single *public coin* message from the verifier to the prover. In particular, the Fiat-Shamir heuristic may be applied whenever π is a constant-round public-coin protocol with negligible soundness error, which is indeed the case for the protocol of Wesolowski [Wes19]. It should be noted that even though the protocol of Pietrzak [Pie19] is not constant-round, the Fiat-Shamir heuristic may still be applied to it in the random oracle model, and so it can also be applied to the compiled version thereof using our compiler.

Replacing PRF with a pseudorandom generator. Our use of PRF enables us to handle cases in which n is not a-priori bounded and can be chosen by the malicious prover (this is in line with Definition 3.2). However, in many cases it makes sense to consider values of n which *are* a-priori bounded. In such cases, we can replace our use of the pseudorandom function with a pseudorandom generator PRG mapping seeds of length λ to outputs of length $m \cdot n$.[12] Instead of sampling a key K, the verifier will now sample a seed $s \leftarrow \{0,1\}^\lambda$ to PRG and send it over to the prover. Then, both the prover and the verifier can compute $y = \mathsf{PRG}(s)$ and parse y as the natural encoding of m subsets $\mathcal{I}_1, \ldots, \mathcal{I}_m$ of $[n]$ (i.e., for each $j \in [m]$, the vector $(y_{(j-1)\cdot n+1}, \ldots, y_{j \cdot n})$ is the characteristic vector of $\mathcal{I}_j$). In practice, PRG can be efficiently implemented via a cryptographic hash function (e.g., SHA).

[12] We implicitly assume here that m and n are both polynomially-bounded functions of the security parameter.

Proof of Theorem 5.1. We now turn to the proof of Theorem 5.1. We start by analyzing the communication complexity and the verifier's running time. Per the running time of the verifier: It samples a random key $K \leftarrow \{0,1\}^\lambda$, taking time λ; makes $m \cdot n$ invocation of PRF, taking time $m \cdot n \cdot t_{\mathsf{prf}}$; computes $2m$ products of at most n group elements each, taking time $O(m \cdot n \cdot t_{\mathsf{op}})$; and participates in m executions of π, which takes time $m \cdot t$. It follows that her running time is $t' = t + \lambda + O(m \cdot n \cdot (t_{\mathsf{op}} + t_{\mathsf{prf}}))$. The communication in $\mathsf{Batch}_2^m(\pi)$ includes the key K and all messages exchanged in m executions of π, resulting in a total communication complexity of $c' = m \cdot c + \lambda$. The δ'-soundness of $\mathsf{Batch}_2^m(\pi)$ follows immediately from the following lemma and the pseudorandomness of PRF.

Lemma 5.2. *For every pair $\mathsf{P}^*_{\mathsf{Batch}} = (\mathsf{P}^*_{\mathsf{Batch},1}, \mathsf{P}^*_{\mathsf{Batch},2})$ of probabilistic polynomial time algorithms, there exist a probabilistic polynomial-time algorithm D and a negligible function $\nu(\cdot)$ such that*

$$\mathsf{Adv}^{\mathsf{BPoCE}}_{\mathsf{Batch}_2^m(\pi),\mathsf{P}^*_{\mathsf{Batch}}} \leq \delta + 2^{-m} + \mathsf{Adv}_{\mathsf{PRF},\mathsf{D}}(\lambda) + \nu(\lambda)$$

for all sufficiently large $\lambda \in \mathbb{N}$.

Proof. Let $\mathsf{P}^*_{\mathsf{Batch}} = (\mathsf{P}^*_{\mathsf{Batch},1}, \mathsf{P}^*_{\mathsf{Batch},2})$ be a pair of probabilistic polynomial-time algorithms trying to break the soundness of $\mathsf{Batch}_2^m(\pi)$. Define the following events:

- Let $\mathsf{NotEqual}$ denote the event in which for some $i \in [n]$, it holds that $x_i^e \neq y_i$, where $(\mathbb{G}, \mathsf{pp}) \leftarrow \mathsf{GGen}(1^\lambda)$, $(n, \vec{x}, \vec{y}, e, \mathsf{st}) \leftarrow \mathsf{P}^*_{\mathsf{Batch},1}(\mathbb{G}, \mathsf{pp})$, $\vec{x} = (x_1, \ldots, x_n)$ and $\vec{y} = (y_1, \ldots, y_n)$.
- Let $\mathsf{BiasedSet}$ be the event in which there exists some $j^* \in [m]$ such that

$$\left(\prod_{i \in \mathcal{I}_{j^*}} x_i \right)^e \neq \prod_{i \in \mathcal{I}_{j^*}} y_i,$$

  where $n, \vec{x}, \vec{y}$ and e are as before, $K \leftarrow \{0,1\}^\lambda$ and $\mathcal{I}_1, \ldots, \mathcal{I}_m$ are chosen using $\mathsf{PRF}_K(\cdot)$ as in Step 2(a) of $\mathsf{Batch}_2^m(\pi)$.
- Let $\mathsf{PBatchWin}$ be the event in which $\mathsf{P}^*_{\mathsf{Batch}}$ wins; that is, $\mathsf{V}_{\mathsf{Batch}}$ outputs 1 and $\mathsf{NotEqual}$ holds.

By definition, it holds that

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{BPoCE}}_{\mathsf{Batch}_2^m(\pi),\mathsf{P}^*_{\mathsf{Batch}}} &= \Pr[\mathsf{PBatchWin}] \\
&= \Pr[\mathsf{PBatchWin} \wedge \mathsf{NotEqual}] \qquad (5.1) \\
&= \Pr[\mathsf{PBatchWin} \wedge \mathsf{NotEqual} \wedge \mathsf{BiasedSet}] \\
&\quad + \Pr\left[\mathsf{PBatchWin} \wedge \mathsf{NotEqual} \wedge \overline{\mathsf{BiasedSet}}\right] \\
&\leq \Pr[\mathsf{PBatchWin} \wedge \mathsf{BiasedSet}] + \Pr\left[\mathsf{NotEqual} \wedge \overline{\mathsf{BiasedSet}}\right], \qquad (5.2)
\end{aligned}$$

where Eq. (5.1) follows from the fact that $\mathsf{PBatchWin} \subseteq \mathsf{NotEqual}$. The lemma then follows from the following two claims. Claim 5.3 below uses the pseudorandomness of PRF in order to bound the probability of the event $\mathsf{NotEqual} \wedge \overline{\mathsf{BiasedSet}}$.

Claim 5.3. *There exists a probabilistic polynomial-time algorithm D such that*

$$\mathsf{Adv}_{\mathsf{PRF},\mathsf{D}}(\lambda) \geq \Pr\left[\mathsf{NotEqual} \wedge \overline{\mathsf{BiasedSet}}\right] - 2^{-m}$$

for every $\lambda \in \mathbb{N}$.

Claim 5.4 reduces the event $\mathsf{PBatchWin} \wedge \mathsf{BiasedSet}$ to the soundness of the underlying PoCE protocol π.

Claim 5.4. *There exists a pair* $\mathsf{P}^* = (\mathsf{P}_1^*, \mathsf{P}_2^*)$ *of probabilistic polynomial-time algorithms such that*

$$\mathsf{Adv}^{\mathsf{PoCE}}_{\pi,\mathsf{P}^*} \geq \Pr\left[\mathsf{PBatchWin} \wedge \mathsf{BiasedSet}\right]$$

for every $\lambda \in \mathbb{N}$.

We first conclude the proof of Lemma 5.2 and then prove Claims 5.3 and 5.4. Taken together with Eq. (5.2) and rearranging, the two claims indeed imply the existence of probabilistic polynomial-time algorithm D and of a pair $\mathsf{P}^* = (\mathsf{P}_1^*, \mathsf{P}_2^*)$ of probabilistic polynomial-time algorithms such that

$$\mathsf{Adv}^{\mathsf{BPoCE}}_{\mathsf{Batch}_2^m(\pi),\mathsf{P}^*_{\mathsf{Batch}}} \leq \mathsf{Adv}_{\mathsf{PRF},\mathsf{D}}(\lambda) + \mathsf{Adv}^{\mathsf{BPoCE}}_{\pi,\mathsf{P}^*} + 2^{-m}$$

for every $\lambda \in \mathbb{N}$. By the δ-soundness of π, this implies that there exists a negligible function $\nu(\cdot)$, such that

$$\mathsf{Adv}^{\mathsf{BPoCE}}_{\mathsf{Batch}_2^m(\pi),\mathsf{P}^*_{\mathsf{Batch}}} \leq \delta + 2^{-m} + \mathsf{Adv}_{\mathsf{PRF},\mathsf{D}}(\lambda) + \nu(\lambda)$$

for all sufficiently large $\lambda \in \mathbb{N}$, concluding the proof of Lemma 5.2. ■

We now turn to prove the two claims, starting with Claim 5.3.

Proof of Claim 5.3. Consider the probabilistic polynomial-time algorithm D, which on input 1^λ and oracle access to an oracle $\mathcal{O}$ is defined as follows:

1. Sample $(\mathbb{G}, \mathsf{pp}) \leftarrow \mathsf{GGen}(1^\lambda)$.
2. Invoke $(n, \vec{x}, \vec{y}, e, \mathsf{st}) \leftarrow \mathsf{P}^*_{\mathsf{Batch},1}(\mathbb{G}, \mathsf{pp})$.
3. If for each $i \in [n]$ it holds that $x_i^e = y_i$, then output 0 and terminate.
4. For $j = 1, \ldots, m$:
   (a) Initialize $\mathcal{I}_j := \emptyset$.
   (b) For $i = 1, \ldots, n$, query the oracle for $\mathcal{O}(j \| i)$ and if the answer is 1 then update $\mathcal{I}_j := \mathcal{I}_j \cup \{i\}$.
   (c) Compute $u_j := \prod_{i \in \mathcal{I}_j} x_i$ and $w_j := \prod_{i \in \mathcal{I}_j} y_i$.
5. If for every $j \in [m]$ it holds that $u_j^e = w_j$, then output 1; and otherwise, output 0.

We now proceed to bound $\mathsf{Adv}_{\mathsf{PRF},\mathsf{D}}(\lambda)$ by considering two cases:

- If D has oracle access to $\mathsf{PRF}_K(\cdot)$ for $K \leftarrow \{0,1\}^\lambda$: In this case the values of $(\vec{x}, \vec{y}, (\mathcal{I}_j)_{j \in [m]})$ are distributed identically to as they are distributed in a random execution of $\mathsf{Batch}_2^m(\pi)$. Moreover, D outputs 1 if there exists an $i^* \in [n]$ for which $x_{i^*}^e \neq y_{i^*}$ yet for all $j \in [m]$ it holds that $\left(\prod_{i \in \mathcal{I}_j} x_i\right)^e = \prod_{i \in \mathcal{I}_j} y_i$. Hence, by definition of the events $\mathsf{NotEqual}$ and $\mathsf{BiasedSet}$, it holds that

$$\Pr_{K \leftarrow \{0,1\}^\lambda}\left[\mathsf{D}(1^\lambda)^{\mathsf{PRF}_K(\cdot)} = 1\right] = \Pr\left[\mathsf{NotEqual} \wedge \overline{\mathsf{BiasedSet}}\right] \tag{5.3}$$

  for every $\lambda \in \mathbb{N}$.

- If D has oracle access to $f(\cdot)$ for $f \leftarrow \mathcal{F}_\lambda$ (where $\mathcal{F}_\lambda$ denotes the set of all functions from $\{0,1\}^{\lceil \log(m \cdot n) \rceil}$ to $\{0,1\}$): In this case, the values of $(\vec{x}, \vec{y})$ are distributed identically to as they are distributed in a random execution of $\mathsf{Batch}_2^m(\pi)$. However, the subsets $\mathcal{I}_1, \ldots, \mathcal{I}_m$ are uniformly and independently sampled from $2^{[n]}$. Hence,

$$\begin{aligned}
\Pr_{f \leftarrow \mathcal{F}_\lambda}\left[\mathsf{D}(1^\lambda)^{f(\cdot)} = 1\right] &= \Pr\left[\mathsf{NotEqual} \wedge \left(\forall j \in [m] : \left(\prod_{i \in \mathcal{I}_j} x_i\right)^e = \prod_{i \in \mathcal{I}_j} y_i\right)\right] \\
&\leq \Pr\left[\forall j \in [m] : \left(\prod_{i \in \mathcal{I}_j} x_i\right)^e = \prod_{i \in \mathcal{I}_j} y_i \middle| \mathsf{NotEqual}\right] \\
&\leq 2^{-m}, \qquad (5.4)
\end{aligned}$$

for every $\lambda \in \mathbb{N}$, where $\mathcal{I}_1, \ldots, \mathcal{I}_m \leftarrow 2^{[n]}$, and Eq. (5.4) follows from Lemma 4.2 and the fact that $\mathcal{I}_1, \ldots, \mathcal{I}_m$ are sampled independently.

From Eq. (5.3) and (5.4), it follows that

$$\begin{aligned}
\mathsf{Adv}_{\mathsf{PRF},\mathsf{D}}(\lambda) &= \left| \Pr_{K \leftarrow \{0,1\}^\lambda}\left[\mathsf{D}(1^\lambda)^{\mathsf{PRF}_K(\cdot)} = 1\right] - \Pr_{f \leftarrow \mathcal{F}_\lambda}\left[\mathsf{D}(1^\lambda)^{f(\cdot)} = 1\right] \right| \\
&\geq \Pr_{K \leftarrow \{0,1\}^\lambda}\left[\mathsf{D}(1^\lambda)^{\mathsf{PRF}_K(\cdot)} = 1\right] - \Pr_{f \leftarrow \mathcal{F}_\lambda}\left[\mathsf{D}(1^\lambda)^{f(\cdot)} = 1\right] \\
&\geq \Pr\left[\mathsf{NotEqual} \wedge \overline{\mathsf{BiasedSet}}\right] - 2^{-m}
\end{aligned}$$

for every $\lambda \in \mathbb{N}$, concluding the proof of Claim 5.3. ∎

We now conclude this section by proving Claim 5.4.

Proof of Claim 5.4. Consider the following pair $\mathsf{P}^* = (\mathsf{P}_1^*, \mathsf{P}_2^*)$ of probabilistic polynomial-time algorithms, attempting to break the soundness of π. On input $(\mathbb{G}, \mathsf{pp})$ generated by $\mathsf{GGen}(1^\lambda)$, the algorithm P_1^* is defined as follows:

1. Invoke $(n, \vec{x}, \vec{y}, e, \mathsf{st}) \leftarrow \mathsf{P}^*_{\mathsf{Batch},1}(\mathbb{G}, \mathsf{pp})$, where $\vec{x} = (x_1, \ldots, x_n)$ and $\vec{y} = (y_1, \ldots, y_n)$.
2. Sample $K \leftarrow \{0,1\}^\lambda$.
3. For $j = 1, \ldots, m$:
   (a) Compute $\mathcal{I}_j = \{i \in [n] \ : \ \mathsf{PRF}_K(j \| i) = 1\}$.
   (b) Compute $u_j = \prod_{i \in \mathcal{I}_j} x_i$ and $w_j = \prod_{i \in \mathcal{I}_j} y_i$.
4. Find the first index $j^* \in [m]$ for which $u_{j^*}^e \neq w_{j^*}$; and if no such index exists, output $\perp$ and terminate.
5. Set $\mathsf{st}' = (\mathsf{st}, \vec{x}, \vec{y}, K, j^*, \vec{u}, \vec{w})$, where $\vec{u} = (u_1, \ldots, u_m)$ and $\vec{w} = (w_1, \ldots, w_m)$.
6. Output $(u_{j^*}, w_{j^*}, e, \mathsf{st}')$.

Then, the algorithm P_2^*, running on private input st' and interacting with the verifier V on joint input $(\mathbb{G}, \mathsf{pp}, u_{j^*}, w_{j^*}, e)$, is defined as follows:

1. Parse st' as $(\mathsf{st}, \vec{x}, \vec{y}, K, j^*, \vec{u}, \vec{w})$.
2. Invoke $\mathsf{P}^*_{\mathsf{Batch},2}$ on input st and simulate to $\mathsf{P}^*_{\mathsf{Batch},2}$ an execution of $\mathsf{Batch}_2^m(\pi)$ on joint input $(\mathbb{G}, \mathsf{pp}, \vec{x}, \vec{y}, e)$:
   (a) Send K to $\mathsf{P}^*_{\mathsf{Batch},2}$ as the first message of the verifier in π_{Batch}.

(b) Simulate to $\mathsf{P}^*_{\mathsf{Batch},2}$ the m parallel executions of π (Step 3 in $\mathsf{Batch}_2^m(\pi)$) as follows:
- In the j^*-th execution, let V play the role of the verifier, by relaying all messages from $\mathsf{P}^*_{\mathsf{Batch},2}$ to V and vice versa.
- For each $j \in [m] \setminus \{j^*\}$, in the j-th execution P_2^* honestly plays the role of the verifier on joint input $(\mathbb{G}, \mathsf{pp}, u_j, w_j, e)$.

Let PWin be the event in which P^* wins; that is, V outputs 1 and $\mathsf{BiasedSet}$ holds. Recall that by the definition of $\mathsf{Batch}_2^m(\pi)$, the verifier $\mathsf{V}_{\mathsf{Batch}}$ outputs 1 only if each of the verifiers in the m parallel executions outputs 1, including the verifier in the j^* execution. Hence,

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{PoCE}}_{\pi,\mathsf{P}^*} &= \Pr[\mathsf{PWin}] \\
&= \Pr[\mathsf{V} \text{ outputs } 1 \wedge \mathsf{BiasedSet}] \\
&\geq \Pr[\mathsf{V}_{\mathsf{Batch}} \text{ outputs } 1 \wedge \mathsf{BiasedSet}] \\
&= \Pr[\mathsf{PBatchWin} \wedge \mathsf{BiasedSet}] \qquad (5.5)
\end{aligned}$$

for every $\lambda \in \mathbb{N}$, where Eq. (5.5) follows from the fact that $\mathsf{BiasedSet} \subseteq \mathsf{NotEqual}$ and $\mathsf{PBatchWin} = (\mathsf{V}_{\mathsf{Batch}} \text{ outputs } 1) \wedge \mathsf{NotEqual}$. ∎

6 An Improved Compiler From the Low Order Assumption

In this section we present an improved compiler, which enjoys significant communication improvements over our general compiler from Section 5. Concretely, the communication complexity of the resulted protocol is completely independent of the additional soundness error (the verification time, though also improved, still depends on it).[13] The cost is that this compiler, unlike the previous one, relies on an algebraically-structured computational assumption – the low order assumption (recall Definition 2.4). However, this caveat does not seem overly restrictive when to compiler is applied to either the protocol of Pietrzak or to that of Wesolowski [Pie19, Wes19], both of which rely either on this assumption or stronger ones. Our compiler is inspired by an approach presented by Bellare, Garay and Rabin [BGR98] (which also implicitly underlies the batch proof of Wesolowski [Wes20]), while introducing some new ideas for the setting of succinct BPoCE (see Sections 1.2 and 1.3 for details).

6.1 The Compiler

We now present the compiler. Let GGen be a group generation algorithm (recall Section 2). Let $\delta = \delta(\lambda)$ a be a function of the security parameter $\lambda \in \mathbb{N}$, and let $c = c(\lambda, e)$ and $t = t(\lambda, e)$ be functions of λ and of the exponent $e \in \mathbb{N}$. Our compiler is parameterized by an integer s, and uses the following building blocks:

- A (δ, c, t)-PoCE $\pi = (\mathsf{GGen}, \mathsf{P}, \mathsf{V})$.
- A pseudorandom function family PRF, such that for every $\lambda \in \mathbb{N}$ and for every key $K \in \{0,1\}^\lambda$, the function PRF_K maps inputs in $\{0,1\}^{\lceil \log(n) \rceil}$ to outputs in $[s]$.[14] For ease of notation, for an integer $i \in [n]$, we will write $\mathsf{PRF}_K(i)$, with the intention (but without noting it explicitly) that the input to the function is a bit string representing the integer i.

[13] Recall that in the compiler from Section 5, in order to obtain an additive soundness loss of 2^{-m}, the communication had to grow linearly with m.

[14] Given any efficient algorithm Samp for sampling from the uniform distribution over $[s]$ using $r = r(s)$ random coins, PRF can be implemented by invoking a PRF mapping $\{0,1\}^{\lceil \log(n) \rceil}$ into $\{0,1\}^r$ and then applying Samp using the output of the PRF as random coins.

The compiler produces a protocol $\mathsf{Batch}_3^s(\pi) = (\mathsf{GGen}, \mathsf{P}', \mathsf{V}')$, which is a (δ', c', t')-BPoCE for related functions $\delta' = \delta'(\lambda)$, $c' = c'(\lambda, e, n)$ and $t' = t'(\lambda, e, n)$.

The Protocol $\mathsf{Batch}_3^s(\pi)$

Joint input: Public parameters $(\mathbb{G}, \mathsf{pp})$ generated by $\mathsf{GGen}(1^\lambda)$, vectors $\vec{x} = (x_1, \ldots, x_n)$ and $\vec{y} = (y_1, \ldots, y_n)$ of group elements, and an exponent $e \in \mathbb{N}$.

1. V' samples $K \leftarrow \{0,1\}^\lambda$ and sends K to P'.
2. For $i = 1, \ldots, n$: Both V' and P' compute $\alpha_i := \mathsf{PRF}_K(i)$.
3. Both V' and P' compute $x = \prod_{i \in [n]} x_i^{\alpha_i}$ and $y = \prod_{i \in [n]} y_i^{\alpha_i}$.
4. V' and P' execute the protocol π on joint input $(\mathbb{G}, \mathsf{pp}, x, y, e)$, where V' plays the role of V and P' plays the role of P. Let $b \in \{0,1\}$ be the output of V in this execution.
5. V' outputs b.

Similarly to our discussion in Section 5, if the Fiat-Shamir heuristic [FS86] can be applied to π in the random oracle, then it can also be applied to $\mathsf{Batch}_3^s(\pi)$. Moreover, if the number n of pairs (x_i, y_i) is a-priori bounded, then the use of PRF can be replaced by a pseudorandom generator in a similar manner to what was done in Section 5.

Theorem 6.1 establishes the completeness, soundness, succinctness and verifier efficiency of $\mathsf{Batch}_3^s(\pi)$, in cases where the low order assumption holds with respect to GGen. Recall that we denote by $t_{\mathsf{op}} = t_{\mathsf{op}}(\lambda)$ the time required to apply the binary group operation on two group elements in $\mathbb{G}$ that is generated by $\mathsf{GGen}(1^\lambda)$. We also denote by $t_{\mathsf{prf}} = t_{\mathsf{prf}}(\lambda, s, n)$ the time required to compute $\mathsf{PRF}_K(z)$ for $K \in \{0,1\}^\lambda$ and $z \in \{0,1\}^{\lceil \log(n) \rceil}$.

Theorem 6.1. *Assume that* PRF *is a pseudorandom function, that* π *is a* (δ, c, t)*-PoCE, and that the* s*-low order assumption holds with respect to* GGen*. Then,* $\mathsf{Batch}_3^s(\pi)$ *is a* (δ', c', t')*-BPoCE, where:*

- $\delta'(\lambda) = \delta(\lambda) + 1/s$.
- $c'(\lambda, n, e) = c(\lambda, e) + \lambda$.
- $t'(\lambda, n, e) = t(\lambda, e) + \lambda + n \cdot t_{\mathsf{prf}} + O(n \cdot \log(s) \cdot t_{\mathsf{op}})$.

Instantiating the compiler. Basing the compiler on the general low order assumption gives rise to several possible instantiations. In particular:

- **The groups QR_N and QR_N^+.** The low order assumption holds unconditionally in the group QR_N of quadratic residues modulo N when N is the product of two safe primes, as well as in the (isomorphic) group QR_N^+ of *signed* quadratic residues modulo N (recall Section 2). Concretely, if $N = (2p+1) \cdot (2q+1)$ for prime p and q, then QR_N and QR_N^+ contain no elements of order less than $\min\{p, q\}$. In the context of VDFs, this was observed by Pietrzak [Pie19] and by Boneh et al. [BBF18]. However, it is also plausible that the assumption holds computationally in the groups QR_N and QR_N^+ when the factors of N are not safe primes.

- **RSA groups.** It is tempting to instantiate our compiler within the RSA group $\mathbb{Z}_N^*$ as perhaps the best-understood group of unknown order. Alas, the low order assumption cannot hold in $\mathbb{Z}_N^*$ since $-1 \in \mathbb{Z}_N$ is always of order two in this group. One possible ramification, suggested by Boneh et al. is to work over the quotient group $\mathbb{Z}_N^*/\{\pm 1\}$. Another possibility is to settle on a slightly weaker soundness guarantee for BPoCEs, which allows a malicious prover to convince the verifier that $y_i = x_i^e$ for every i, even though for some indices $y_i = -x_i^e$. This weakened soundness guarantee is defined in Appendix A and can be shown to follow from the weak low

order assumption (Definition 2.5), using essentially the same proof as the proof of Theorem 6.1. Moreover, Seres and Burcsi [SB20] have shown that when N is the product of two safe primes, breaking the weak low order assumption in $\mathbb{Z}_N^*$ is equivalent to factoring the modulus N.[15] A third option is to compose our compiler with an additional protocol, specifically dedicated to proving that $y_i \neq -x_i^e$ for each i. We present and analyze such a protocol in Section 7.

- **Class groups of imaginary quadratic fields.** These groups were suggested in the context of VDFs by Wesolowski [Wes19] as candidate groups of unknown order. The security of the low order assumption in these groups is still unclear [BBF18, BKS+20]; but at least until proven otherwise, it is possible that our compiler can be instantiated in a sub-family of these groups as well. See the recent work of Belabas et al. [BKS+20] for further details on the possible choice of parameters for such groups.

On the tightness of the reduction. In Section 6.2 we prove the soundness of our compiler based on the low order assumption. This general reduction, however, suffers from a cubic security loss: Given a prover which breaks the soundness of the resulting BPoCE with advantage $\delta + 1/s + \epsilon$, we construct an adversary breaking the low order assumption with advantage $O(\epsilon^3)$. Coming up with a tight reduction to the general low order assumption seems to be beyond current techniques. Hence, instead, in Appendix A, we give specific proofs for the soundness of our compiler in the groups QR_N^+ and $\mathbb{Z}_N^*$. In the former, our proof is information-theoretic, while in the latter, it relies on a tight reduction to the factoring assumption.

Necessity of the low order assumption. We note that our reliance on the s-low-order assumption in Theorem 6.1 is necessary. To see why that is, suppose that we work in a group $\mathbb{G}$ in which the assumption does not hold; that is, given the group description it is easy to find a group element z and an integer $\omega < s$ such that $z^\omega = 1_\mathbb{G}$. In this case, the attacker can output an instance $((x_i, y_i)_{i\in[n]}, e)$ such that n and e are arbitrary integers, $x_1, \ldots, x_n$ are arbitrary group elements, $y_i = x_i^e$ for every $i \in \{2, \ldots, n\}$ and $y_1 = z \cdot x_1^e$. The verifier V' will incorrectly accept whenever the group elements x and y computed by P' and V' in Step 3 of the protocol satisfy $y = x^e$. This occurs when the exponents $\alpha_1, \ldots, \alpha_n$ satisfy $(\prod_{i=1}^n x_i^{\alpha_i})^e = \prod_{i=1}^n y_i^{\alpha_i}$. By the choices made by the attacker, this equality holds whenever $z^{\alpha_1} = 1_\mathbb{G}$, which happens with probability at least $1/s$.

Proof of Theorem 6.1. We first analyze the communication complexity and the running time of the verifier, and then in Section 6.2, we base the soundness of $\mathsf{Batch}_3^s(\pi)$ on the low order assumption. As for the running time of verifier: It samples a random key $K \leftarrow \{0,1\}^\lambda$, taking time λ; makes n invocation of PRF, taking time $n \cdot t_{\mathsf{prf}}$; raises n elements to exponents which are bounded by s, which takes time $O(n \cdot \log(s) \cdot t_{\mathsf{op}})$; computes the product of n group elements, taking time $(n-1) \cdot t_{\mathsf{op}}$); and participates in a single execution of π, which takes time t. It follows that her running time is $t' = t + \lambda + n \cdot t_{\mathsf{prf}} + O(n \cdot (\log(s) + 1) \cdot t_{\mathsf{op}})$. The communication in $\mathsf{Batch}_3^s(\pi)$ includes the key K and all messages exchanged in a single execution of π, resulting in a total communication of $c' = c + \lambda$.

6.2 Soundness Analysis Based on the Low Order Assumption

The proof of soundness for $\mathsf{Batch}_3^s(\pi)$ follows the same outline as did the corresponding proof in Section 5, and is by reduction to the δ-soundness of π, to the pseudorandomness of PRF and to the low order assumption with respect to GGen. Since the reduction and its analysis are extremely similar

[15] Their proof can also be used, essentially unchanged, to show that (the strong variant of) the low order assumption in $\mathbb{Z}_N^*$ $\{\pm 1\}$ is equivalent to factoring N.

to those presented in Section 5, we forgo presenting them explicitly here, and instead concentrate on the main differences.

Concretely, the only major difference between the soundness analysis of $\mathsf{Batch}_3^s(\pi)$ and the analysis of $\mathsf{Batch}_2^m(\pi)$ in Section 5, is that instead of relying on Lemma 4.2 in order to lower bound the probability that $x^e \neq y$ (where x and y are computed from $\vec{x}, \vec{y}$ as defined in Step 3 of $\mathsf{Batch}_3^s(\pi)$), we rely on Lemma 6.2 and Corollary 6.3 found below. Loosely, relying on the low order assumption with respect to GGen, Lemma 6.2 and its corollary assert that if there is some $i \in [n]$ for which $x_i^e \neq y_i$, then with probability at most $1/s + \mathsf{negl}(\lambda)$ over the choice of $\alpha_1, \ldots, \alpha_n \leftarrow [s]$, it holds that $x^e = y$.[16]

Lemma 6.2. *Let $\mathbb{G}$ be a group. For every integers $n, e \in \mathbb{N}$, any integer s, any number $\epsilon \in (0,1)$ and any vectors $\vec{x}, \vec{y} \in \mathbb{G}^n$ the following holds: If there exists an index $i \in [n]$ such that $x_i^e \neq y_i$ and*

$$\Pr_{\alpha_1,\ldots,\alpha_n \leftarrow [s]}\left[\left(\prod_{i\in[n]} x_i^{\alpha_i}\right)^e = \prod_{i\in[n]} y_i^{\alpha_i}\right] > \frac{1}{s} + \epsilon,$$

then there exists an algorithm A which receives as input $\vec{x}, \vec{y}$ and e, runs in time $\mathsf{poly}(\lambda, n, \log(e))$, and with probability at least ϵ^2 outputs $(u, \omega) \in \mathbb{G} \times \mathbb{N}$ such that $u \neq 1_{\mathbb{G}}$, $1 < \omega < s$ and $u^\omega = 1_{\mathbb{G}}$.

Corollary 6.3 below follows from Lemma 6.2 and from the definition of the low order assumption (Definition 2.4).

Corollary 6.3. *Let GGen be a group generation algorithm and let $s = s(\lambda)$ be a function of the security parameter λ. If the s-low order assumption holds with respect to GGen, then for any probabilistic polynomial-time algorithm P_0^*, there exists a negligible function $\nu(\cdot)$ such that*

$$\Pr\left[\begin{array}{l}\exists i \in [n], x_i^e \neq y_i \\ \left(\prod_{i\in[n]} x_i^{\alpha_i}\right)^e = \prod_{i\in[n]} y_i^{\alpha_i}\end{array}\middle|\begin{array}{l}(\mathbb{G}, \mathsf{pp}) \leftarrow \mathsf{GGen}(1^\lambda) \\ (n, \vec{x}, \vec{y}, e) \leftarrow \mathsf{P}_0^*(\mathbb{G}, \mathsf{pp}) \\ \alpha_1, \ldots, \alpha_n \leftarrow [s]\end{array}\right] \leq \frac{1}{s} + \nu(\lambda),$$

for all sufficiently large $\lambda \in \mathbb{N}$.

Proof of Corollary 6.3. Let P_0^* be a probabilistic polynomial-time algorithm as in the statement of the corollary, and assume towards contradiction that there exists a polynomial $f(\cdot)$ such that

$$\Pr\left[\begin{array}{l}\exists i \in [n], x_i^e \neq y_i \\ \left(\prod_{i\in[n]} x_i^{\alpha_i}\right)^e = \prod_{i\in[n]} y_i^{\alpha_i}\end{array}\middle|\begin{array}{l}(\mathbb{G}, \mathsf{pp}) \leftarrow \mathsf{GGen}(1^\lambda) \\ (n, \vec{x}, \vec{y}, e) \leftarrow \mathsf{P}_0^*(\mathbb{G}, \mathsf{pp}) \\ \alpha_1, \ldots, \alpha_n \leftarrow [s]\end{array}\right] > \frac{1}{s} + \frac{1}{f(\lambda)}, \tag{6.1}$$

for infinitely many values of $\lambda \in \mathbb{N}$. Consider the following polynomial-time algorithm B which receives as input $(\mathbb{G}, \mathsf{pp}) \leftarrow \mathsf{GGen}(1^\lambda)$ and attempts to find a low order element of $\mathbb{G}$:

1. Invoke $(n, \vec{x}, \vec{y}, e) \leftarrow \mathsf{P}_0^*(\mathbb{G}, \mathsf{pp})$.
2. Invoke $(u, \omega) \leftarrow \mathsf{A}(\vec{x}, \vec{y}, e, N)$, where A is the algorithm guaranteed by Lemma 6.2.
3. Output (u, ω).

[16] Observe that in $\mathsf{Batch}_3^s(\pi)$, the exponents $\alpha_1, \ldots, \alpha_n$ are not chosen uniformly at random from $[s]$, but using the pseudorandom function PRF. This is handled in exactly the same manner in which it was handled in the proof of Claim 5.3.

Say that a tuple $(\mathbb{G}, \mathsf{pp}, n, \vec{x}, \vec{y}, e)$ is *good* if there exists some $i \in [n]$ such that $x_i^e \neq y_i$ and in addition it holds that

$$\Pr_{\alpha_1,\ldots,\alpha_n \leftarrow [s]}\left[\left(\prod_{i\in[n]} x_i^{\alpha_i}\right)^e = \prod_{i\in[n]} y_i^{\alpha_i}\right] > \frac{1}{s} + \frac{1}{2f(\lambda)}.$$

By the assumption (6.1) and total probability it holds that

$$\Pr\left[(\mathbb{G}, \mathsf{pp}, n, \vec{x}, \vec{y}, e) \text{ is good}\right] \geq \frac{1}{2f(\lambda)},$$

for infinitely many values of $\lambda \in \mathbb{N}$, where the probability is taken over $(\mathbb{G}, \mathsf{pp}) \leftarrow \mathsf{GGen}(1^\lambda)$ and $(n, \vec{x}, \vec{y}, e) \leftarrow \mathsf{P}_0^*(\mathbb{G}, \mathsf{pp})$.

Observe that Lemma 6.2 implies that conditioned on the event in which the tuple $(\mathbb{G}, \mathsf{pp}, n, \vec{x}, \vec{y}, e)$ is good (where $(\mathbb{G}, \mathsf{pp})$ is sampled by $\mathsf{GGen}(1^\lambda)$ as the input to B and $(n, \vec{x}, \vec{y}, e)$ is sampled by B in Step 1), with probability at least $1/(4 \cdot f^2(\lambda))$ it holds that $u \neq 1_\mathbb{G}$, $1 < \omega < s$ and $u^\omega = 1_\mathbb{G}$. That is,

$$\Pr\left[\mathsf{LowOrd}_{s,\mathsf{A}}^{\mathsf{GGen}}(\lambda) = 1 \middle| (\mathbb{G}, \mathsf{pp}, n, \vec{x}, \vec{y}, e) \text{ is good}\right] \geq \frac{1}{4 \cdot f^2(\lambda)},$$

for every $\lambda \in \mathbb{N}$, where $(\mathbb{G}, \mathsf{pp}) \leftarrow \mathsf{GGen}(1^\lambda)$ and $(n, \vec{x}, \vec{y}, e) \leftarrow \mathsf{P}_0^*(\mathbb{G}, \mathsf{pp})$. Hence, by total probability, it follows that

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{LowOrd}}_{\mathsf{GGen},s,\mathsf{A}}(\lambda) &= \Pr\left[\mathsf{LowOrd}_{s,\mathsf{A}}^{\mathsf{GGen}}(\lambda) = 1\right] \\
&\geq \Pr\left[\mathsf{LowOrd}_{s,\mathsf{A}}^{\mathsf{GGen}}(\lambda) = 1 \middle| (\mathbb{G}, \mathsf{pp}, n, \vec{x}, \vec{y}, e) \text{ is good}\right] \cdot \Pr\left[(\mathbb{G}, \mathsf{pp}, n, \vec{x}, \vec{y}, e) \text{ is good}\right] \\
&\geq \frac{1}{2 \cdot f(\lambda)} \cdot \frac{1}{4 \cdot f^2(\lambda)} \\
&= \frac{1}{8 \cdot f^3(\lambda)},
\end{aligned}$$

for infinitely many values of $\lambda \in \mathbb{N}$, in contradiction to the assumption that the s-low order assumption holds with respect to GGen. ■

We now complete the analysis by proving Lemma 6.2

Proof of Lemma 6.2. Suppose that there exists an index $i \in [n]$ such that $x_i^e \neq y_i$, and let i^* be the minimal index for which this holds. Let $\epsilon \in (0, 1)$ and assume that

$$\Pr_{\alpha_1,\ldots,\alpha_n \leftarrow [s]}\left[\left(\prod_{i\in[n]} x_i^{\alpha_i}\right)^e = \prod_{i\in[n]} y_i^{\alpha_i}\right] > \frac{1}{s} + \epsilon. \tag{6.2}$$

Consider the following algorithm A, which on input $(\mathbb{G}, \mathsf{pp})$ is defined by:

1. Find the minimal index i^* for which $x_{i^*}^e \neq y_i$.
2. Compute $z = y_{i^*}/x_{i^*}^e$.
3. Sample $\alpha_1, \ldots, \alpha_{i^*-1}, \alpha_{i^*+1}, \ldots, \alpha_n, \beta, \beta' \leftarrow [s]$.
4. Check that:
   - $\left(x_{i^*}^\beta \cdot \prod_{i\in[n]\setminus\{i^*\}} x_i^{\alpha_i}\right)^e = y_{i^*}^\beta \cdot \prod_{i\in[n]\setminus\{i^*\}} y_i^{\alpha_i}$;
   - $\left(x_{i^*}^{\beta'} \cdot \prod_{i\in[n]\setminus\{i^*\}} x_i^{\alpha_i}\right)^e = y_{i^*}^{\beta'} \cdot \prod_{i\in[n]\setminus\{i^*\}} y_i^{\alpha_i}$; and

- $\beta \neq \beta'$.

If any of the checks fail, output $\perp$ and terminate. Otherwise, set $\omega = |\beta - \beta'|$.

5. Output (z, ω).

We now turn to analyze the success probability of A. We argue that if A reaches Step 5, then it succeeds with probability 1. Assume that A reaches Step 5, and assume without loss of generality that $\beta > \beta'$ (by the third conidition of Step 4, it holds that $\beta \neq \beta'$; the proof for the case $\beta' > \beta$ is completely symmetric). By the first two conditions of Step 4, it holds that

$$x_{i^*}^{e\cdot(\beta-\beta')} = y_{i^*}^{\beta-\beta'}.$$

Rearranging the above expression, it holds that

$$\left(\frac{y_{i^*}}{x_{i^*}^e}\right)^{\beta-\beta'} = 1. \tag{6.3}$$

Moreover, it is the case that $\omega = \beta - \beta' < s$; and by the assumption that $x_{i^*}^e \neq y_{i^*}$ it follows that $z = y_{i^*}/x_{i^*}^e \neq 1_{\mathbb{G}}$. Therefore, by (6.3), it must also be that $\omega \neq 1$. Hence, we have shown that whenever A reaches Step 5, it indeed outputs (z, ω) such that $z \neq 1_{\mathbb{G}}$, $z^\omega = 1_{\mathbb{G}}$ and $1 < \omega < s$, as required.

We now turn to bound the probability that A reaches Step 5. For a vector $\vec{\alpha}_{-i^*} = (\alpha_1, \ldots, \alpha_{i^*-1}, \alpha_{i^*+1}, \ldots, \alpha_n)$, denote by $\mathcal{B}_{\vec{\alpha}_{-i^*}} \subseteq [s]$ the set of β-s for which the first condition in Step 4 of algorithm A holds; that is

$$\mathcal{B}_{\vec{\alpha}_{-i^*}} = \left\{ \beta \in [s] \middle| \left(x_{i^*}^\beta \cdot \prod_{i \in [n]\setminus\{i^*\}} x_i^{\alpha_i} \right)^e = y_{i^*}^\beta \cdot \prod_{i \in [n]\setminus\{i^*\}} y_i^{\alpha_i} \right\}.$$

Using this notation, algorithm A reaches Step 5 whenever the event $\beta \in \mathcal{B}_{\vec{\alpha}_{-i^*}} \wedge \beta' \in \mathcal{B}_{\vec{\alpha}_{-i^*}} \wedge \beta \neq \beta'$ occurs. Hence, the probability that A reaches Step 5 is

$$\begin{aligned}
\Pr\left[\begin{array}{c} \beta \in \mathcal{B}_{\vec{\alpha}_{-i^*}} \\ \beta' \in \mathcal{B}_{\vec{\alpha}_{-i^*}} \\ \beta \neq \beta' \end{array}\right] &= \Pr\left[\begin{array}{c} \beta \in \mathcal{B}_{\vec{\alpha}_{-i^*}} \\ \beta \neq \beta' \end{array} \middle| \beta' \in \mathcal{B}_{\vec{\alpha}_{-i^*}} \right] \cdot \Pr\left[\beta' \in \mathcal{B}_{\vec{\alpha}_{-i^*}}\right] \\
&= \left(1 - \Pr\left[\begin{array}{c} \beta \notin \mathcal{B}_{\vec{\alpha}_{-i^*}} \\ \vee \\ \beta = \beta' \end{array} \middle| \beta' \in \mathcal{B}_{\vec{\alpha}_{-i^*}} \right]\right) \cdot \Pr\left[\beta' \in \mathcal{B}_{\vec{\alpha}_{-i^*}}\right] \\
&\geq \left(\Pr\left[\beta \in \mathcal{B}_{\vec{\alpha}_{-i^*}} \middle| \beta' \in \mathcal{B}_{\vec{\alpha}_{-i^*}}\right] - \Pr\left[\beta = \beta' \middle| \beta' \in \mathcal{B}_{\vec{\alpha}_{-i^*}}\right]\right) \cdot \Pr\left[\beta' \in \mathcal{B}_{\vec{\alpha}_{-i^*}}\right] \quad (6.4) \\
&= \Pr\left[\begin{array}{c} \beta \in \mathcal{B}_{\vec{\alpha}_{-i^*}} \\ \beta' \in \mathcal{B}'_{\vec{\alpha}_{-i^*}} \end{array}\right] - \Pr\left[\begin{array}{c} \beta' \in \mathcal{B}_{\vec{\alpha}_{-i^*}} \\ \beta = \beta' \end{array}\right] \\
&= \Pr\left[\begin{array}{c} \beta \in \mathcal{B}_{\vec{\alpha}_{-i^*}} \\ \beta' \in \mathcal{B}'_{\vec{\alpha}_{-i^*}} \end{array}\right] - \frac{1}{s} \cdot \Pr\left[\, \beta' \in \mathcal{B}_{\vec{\alpha}_{-i^*}} \,\right], \quad (6.5)
\end{aligned}$$

where $\vec{\alpha}_{-i^*} \leftarrow [s]^{n-1}$, $\beta, \beta' \leftarrow [s]$, and Eq. (6.4) holds by union bound. By total probability, the

expression in Eq. (6.5) is equal to

$$\begin{aligned}
&\frac{1}{s^{n-1}} \cdot \sum_{\vec{\alpha}_{-i^*} \in [s]^{n-1}} \left(\Pr_{\beta,\beta'} \left[\begin{matrix} \beta \in \mathcal{B}_{\vec{\alpha}_{-i^*}} \\ \beta' \in \mathcal{B}'_{\vec{\alpha}_{-i^*}} \end{matrix} \right] - \frac{1}{s} \cdot \Pr_{\beta'} \left[\beta' \in \mathcal{B}_{\vec{\alpha}_{-i^*}} \right] \right) \\
&\qquad = \frac{1}{s^{n-1}} \cdot \sum_{\vec{\alpha}_{-i^*} \in [s]^{n-1}} \left(\Pr_{\beta} \left[\beta \in \mathcal{B}_{\vec{\alpha}_{-i^*}} \right] \right)^2 - \frac{1}{s} \cdot \sum_{\vec{\alpha}_{-i^*} \in [s]^{n-1}} \frac{1}{s^{n-1}} \cdot \Pr_{\beta} \left[\beta \in \mathcal{B}_{\vec{\alpha}_{-i^*}} \right] \\
&\qquad = \frac{1}{s^{n-1}} \cdot \sum_{\vec{\alpha}_{-i^*} \in [s]^{n-1}} \left(\Pr_{\beta} \left[\beta \in \mathcal{B}_{\vec{\alpha}_{-i^*}} \right] \right)^2 - \frac{1}{s} \cdot \Pr_{\vec{\alpha}_{-i^*},\beta} \left[\beta \in \mathcal{B}_{\vec{\alpha}_{-i^*}} \right] \\
&\qquad \geq \left(\Pr_{\vec{\alpha}_{-i^*},\beta} \left[\beta \in \mathcal{B}_{\vec{\alpha}_{-i^*}} \right] \right)^2 - \frac{1}{s} \cdot \Pr_{\vec{\alpha}_{-i^*},\beta} \left[\beta \in \mathcal{B}_{\vec{\alpha}_{-i^*}} \right] \qquad (6.6) \\
&\qquad \geq \epsilon^2, \qquad (6.7)
\end{aligned}$$

where Eq. (6.6) follows from Jensen's Inequality, and Eq. (6.7) follows from the assumption in Eq. (6.2), and the fact that the expression in Eq. (6.6) is positive and increasing in $\Pr_{\vec{\alpha}_{-i^*},\beta} \left[\beta \in \mathcal{B}_{\vec{\alpha}_{-i^*}} \right]$ as long as $\Pr_{\vec{\alpha}_{-i^*},\beta} \left[\beta \in \mathcal{B}_{\vec{\alpha}_{-i^*}} \right] > 1/s$ (which is indeed the case by the assumption). Hence, the total success probability of A in outputting (z, ω) satisfying the conditions in the Lemma is at least ϵ^2, as required. ■

7 Achieving Strong Soundness in RSA Groups

As discussed in Section 6, only the *weak* low order assumption may hold in the group $\mathbb{Z}_N^*$; and hence, when our compiler from Section 6 is used for batch proof of correct exponentiation within this group, we can obtain only a weaker form of soundness. Recall that the issue was that a malicious prover could still convince the verifier that $x_i^e = y_i$ for every i, even though there exists an index j for which $y_j = -x_j^e$ (or several such indices; where subtraction is with respect to the ring $\mathbb{Z}_N$). Therefore, in this section we present a protocol through which the prover can prove to the verifier that this is not the case. In fact, this protocol provides the following information-theoretic soundness guarantee: Whenever the verifier accepts, it is necessarily the case that y_i/x_i^e is a group element of order different than two. When N is the product of two safe primes, the only low order elements in $\mathbb{Z}_N^*$ have order exactly two.

The protocol $\mathsf{OrderCheck}$ is parameterized by an integer m and is defined below. It is inspired by the work of Di Crescenzo et al. [CKK+17], though their approach incurs a communication overhead which is linear in n – exactly what we are trying to avoid. Moreover, their approach is restricted to exponents which are coprime to the group's order (and in particular, exponents which are odd integers); where in the context of VDFs, the exponent is typically a power of 2. We manage to lift both of these restrictions.

The Protocol $\mathsf{OrderCheck}_m = (\mathsf{V}, \mathsf{P})$

Joint input: Public parameters (N, pp) generated by $\mathsf{ModGen}(1^\lambda)$, vectors $\vec{x} = (x_1, \ldots, x_n)$ and $\vec{y} = (y_1, \ldots, y_n)$ of elements in $\mathbb{Z}_N^*$, and an exponent $e \in \mathbb{N}$.

1. V samples $\mathcal{I}_1, \ldots, \mathcal{I}_m \leftarrow 2^{[n]}$ and sends $\mathcal{I}$ to P.
2. P computes $u_i := x_i^{\lceil (e+1)/2 \rceil}$ for each $i \in [n]$ and $w_j := \prod_{i \in \mathcal{I}_j} u_i$ for each $j \in [m]$.
3. P sends $w_1, \ldots, w_m$ to V.

4. V computes $z_i := x_i^{1+(e+1 \mod 2)} \cdot y_i$ for each $i \in [n]$ and $t_j := \prod_{i \in \mathcal{I}_j} z_i$ for each $j \in [m]$.

5. V outputs 1 if $t_j = w_j^2$ for every $j \in [m]$. Otherwise, V outputs 0.

Observe that if $y_i = x_i^e$ for every $i \in [n]$, then the verifier accepts with probability 1. This is true since for every $i \in [n]$ it holds that

$$u_i^2 = x_i^{2 \cdot \lceil (e+1)/2 \rceil} = x_i^{e+1+(e+1 \mod 2)} = x_i^{1+(e+1 \mod 2)} \cdot y_i = z_i.$$

The soundness guarantee of the protocol is captured by the following lemma.

Lemma 7.1. *Let p and q be prime integers such that $p \equiv 3 \mod 4$, and let $N = p \cdot q$. For every integers $n, e, m \in \mathbb{N}$ and vectors $\vec{x}, \vec{y} \in (\mathbb{Z}_N^*)^n$ the following holds: If there exists an index $i \in [n]$ such that $\mathsf{order}(y_i / x_i^e) = 2$ then*

$$\Pr_{\mathcal{I}_1, \ldots, \mathcal{I}_m \leftarrow 2^{[n]}} \left[\forall j \in [m] : t_j = w_j^2 \right] \leq 2^{-m},$$

where $t_1, \ldots, t_m$ and $w_1, \ldots, w_m$ are computed from $\vec{x}, \vec{y}, e$ and $\mathcal{I}_1, \ldots, \mathcal{I}_m$ as defined in the protocol $\mathsf{OrderCheck}_m$.

Before proving Lemma 7.1, we analyze the protocol's communication complexity and the running time of the verifier. The communication includes m subsets of $[n]$, which can be represented using $m \cdot n$ bits and m group elements sent from P to V, which can be represented using $m \cdot \lceil \log N \rceil$ bits. Hence, the total communication is $m \cdot (n + \lceil \log N \rceil)$, which can be reduced to $\lambda + m \cdot \lceil \log N \rceil$, where λ is the security parameter, by succinctly representing the subsets $\mathcal{I}_1, \ldots, \mathcal{I}_m$ using a key to a pseudorandom function (as done in Section 5). As for the verifier's running time, sampling m random subsets of $[n]$ can takes time $O(m \cdot n)$; computing each z_i takes time at most $O(\log^2 N)$; computing each t_j takes time at most $O(n \cdot \log^2 N)$ and the verification in Step 5 takes time $O(m \cdot \log^2 N)$. Overall, the verification time is $O(m \cdot n + (m+n) \cdot \log^2 N)$. When succinctly representing the subsets $\mathcal{I}_1, \ldots, \mathcal{I}_m$ using a PRF key, the verification time is $O(\lambda + m \cdot n \cdot (t_{\mathsf{prf}} + \log^2 N))$, where t_{prf} is the time complexity of a single PRF evaluation.

Putting it all together. Note that the restriction on the modulus N in Lemma 7.1 is satisfied whenever N is the product of two safe primes $p' = (2p+1)$ and $q' = (2q+1)$. Moreover, in this case, the only group elements (other than 1) with order less than $\min\{p, q\}$ have order 2. Therefore, when N is of this form, protocol $\mathsf{OrderCheck}$ can be executed in parallel to our compiler from Section 6, resulting in a BPoCE in RSA groups which satisfies our strong soundness definition (Definition 3.2) while relying merely on the existence of one-way functions; this is compared to executing the compiler from Section 6 on its own, which provides only weak soundness informally defined in Section 6 (and formally defined in Definition A.2).

Concretely, let ModGen be a modulus generation algorithm such that for any $\lambda \in \mathbb{N}$ and any (N, pp) in the support of $\mathsf{ModGen}(1^\lambda)$, it holds that N is the product of two safe primes $p' = 2p+1$ and $q' = 2q+1$ satisfying $2^{\lambda-1} \leq p, q < 2^\lambda$. Let PRF is a pseudorandom function, let π be a (δ, c, t)-PoCE in RSA groups with moduli generated by ModGen, and let $m, s \in \mathbb{N}$ be integers such that $s < 2^{\lambda-1}$.

Corollary 7.2. *Assume that PRF is a pseudorandom function and that π is a (δ, c, t)-PoCE. Then, there is a (δ', c', t')-BPoCE, where:*

- $\delta'(\lambda) = \delta(\lambda) + 1/s + 2^{-m}$.

- $c'(\lambda, n, e) = c(\lambda, e) + O(\lambda)$.
- $t'(\lambda, n, e) = t(\lambda, e) + O(n \cdot m \cdot t_{\mathsf{prf}} + n \cdot \log(s) \cdot m \cdot \lambda^2)$.

In particular, setting m to be super-logarithmic and s to be super-polynomial, we get that the overhead of our compiler in terms of soundness error is negligible. In this regime, and assuming that $\delta(\lambda)$ is negligible as well and that π is a public-coin protocol (as are the protocols of Wesolowski and of Pietrzak [Wes19, Pie19]), the Fiat-Shamir heuristic can be applied to the resulting batch protocol, yielding a non-interactive proof which is longer then the proof for a single instance by an additive $O(\lambda)$ bits.

We now turn to prove Lemma 7.1.

Proof of Lemma 7.1. Let $\lambda, N, n, e, m, \vec{x}$ and $\vec{y}$ be as in the statement Lemma 7.1, and let $\{u_i\}_{i\in[n]}$ and $\{z_i\}_{i\in[n]}$ be the values computed from $\vec{x}, \vec{y}$ and e as in the definition of the protocol. Suppose that there exists an index $i \in [n]$ such that $\mathsf{order}(y_i/x_i^e) = 2$, and let i^* be the minimal index for which this holds. We will prove that

$$\Pr_{\mathcal{I}\leftarrow 2^{[n]}}\left[\left(\prod_{i\in\mathcal{I}} u_i\right)^2 = \prod_{i\in\mathcal{I}} z_i\right] \leq \frac{1}{2}. \tag{7.1}$$

The lemma will then follow immediately from Eq. (7.1) and the fact that the subsets $\mathcal{I}_1, \ldots, \mathcal{I}_m$ are chosen independently.

In order to prove Eq. (7.1), we will actually prove that

$$\Pr_{\mathcal{I}\leftarrow 2^{[n]}}\left[\left(\prod_{i\in\mathcal{I}} z_i\right) \in QR_N\right] \leq \frac{1}{2}, \tag{7.2}$$

where QR_N is the set of quadratic residues modulo N. Eq. (7.1) then immediately follows, since it is always the case that $\left(\prod_{i\in\mathcal{I}} u_i\right)^2 \in QR_N$. Denote $\mathcal{S}_{QR} = \left\{\mathcal{I} \in 2^{[n]} : \prod_{i\in\mathcal{I}} z_i \in QR_N\right\}$ and $\mathcal{S}_{QNR} = \left\{\mathcal{I} \in 2^{[n]} : \prod_{i\in\mathcal{I}} z_i \notin QR_N\right\}$. We wish to prove that $|\mathcal{S}_{QR}| \leq |\mathcal{S}_{QNR}|$, as this will prove Eq. (7.2). To this end, consider a partition $\mathcal{P}$ of $2^{[n]}$ to 2^{n-1} pairs as follows:

$$\mathcal{P} = \left\{(\mathcal{I}, \mathcal{I}\cup\{i^*\}) \ : \ i^* \notin \mathcal{I}\right\}.$$

We argue that for each pair $(\mathcal{I}, \mathcal{I}\cup\{i^*\})$ in $\mathcal{P}$, at most one of $\mathcal{I}$ and $\mathcal{I}\cup\{i^*\}$ is in $\mathcal{S}_{QR}$. This is true since if $\mathcal{I} \in \mathcal{S}_{QR}$, then it necessarily holds that $\mathcal{I}\cup\{i^*\} \in \mathcal{S}_{QNR}$, since

$$\begin{aligned}
\prod_{i\in\mathcal{I}\cup\{i^*\}} z_i &= z_{i^*} \cdot \prod_{i\in\mathcal{I}} z_i \\
&= x_{i^*}^{1+(e+1 \mod 2)} \cdot y_{i^*} \cdot \prod_{i\in\mathcal{I}} z_i \\
&= v \cdot x_{i^*}^{e+1+(e+1 \mod 2)} \cdot \prod_{i\in\mathcal{I}} z_i,
\end{aligned} \tag{7.3}$$

for $v \in \mathbb{Z}_N^*$ such that $\mathsf{order}(v) = 2$, where Eq. (7.3) holds by our assumption that $\mathsf{order}(y_{i^*}/x_{i^*}^e) = 2$. Note that $x_{i^*}^{e+1+(e+1 \mod 2)} \in QR_N$ since $e+1+(e+1 \mod 2)$ is always even, and $\prod_{i\in\mathcal{I}} z_i \in QR_N$ since we assumed that $\mathcal{I} \in \mathcal{S}_{QR}$. We argue that v is a quadratic non-residue modulo N. By the assumption that $p \equiv 3 \mod 4$, it holds that $(p-1)/2$ is an odd integer. Denote $(p-1)/2 = 2\cdot a+1$. Then, since v is of order two, it holds that $v^{(p-1)/2} = v^{2\cdot a+1} = 1^a \cdot v = v \neq 1$, and by Euler's

Criterion, it is indeed the case that v is a quadratic non-residue modulo p. Therefore, v is also a quadratic non-residue modulo N since $N = p \cdot q$. It follows that $\mathcal{I} \cup \{i^*\} \in \mathcal{S}_{QNR}$, as we wanted to show.

Since at most one subset in pair in $\mathcal{P}$ is in $\mathcal{S}_{QR}$, and there are 2^{n-1} such pairs, it follows that $|\mathcal{S}_{QR}| \leq |\mathcal{S}_{QNR}|$, concluding the proof of Lemma 7.1. ■

8 A Statistically-Sound Proof of Correct Exponentiation in RSA Groups

In this Section we present a new proof of correct exponentiation in RSA groups with a modulus which is the product of two safe primes. The proof enjoys statistical soundness, and in conjunction with our compiler from Section 6 and our protocol from Section 7, it implies a batch proof of correct exponentiation in RSA groups with such moduli. The new protocol is obtained by incorporating our techniques from Section 7 with Pietrzak's protocol [Pie19], and is described below. For simplicity of presentation, we assume that the delay parameter T is a power of 2.

The Protocol $\mathsf{RSAPoCE} = (\mathsf{ModGen}, \mathsf{V_{RSA}}, \mathsf{P_{RSA}})$

Joint input: A modulus N generated by $\mathsf{ModGen}(1^\lambda)$, group elements $x, y \in \mathbb{Z}_N^*$, and an exponent 2^T for $T \in \mathbb{N}$.

1. $\mathsf{P_{RSA}}$ and $\mathsf{V_{RSA}}$ set $x_1 := x$, $y_1 := y$ and $T_1 := T$.
2. For $i = 1, \ldots, \log T$:
   (a) If $T_i = 1$, then V outputs 1 if $y = x^2$, and otherwise outputs 0.
   (b) If $T_i > 1$:
      i. $\mathsf{P_{RSA}}$ computes $z_i := x_i^{2^{T_i/2}}$ and $u_i := x_i^{2^{T_i/2-1}+1}$ and sends z_i and u_i to $\mathsf{V_{RSA}}$.
      ii. $\mathsf{V_{RSA}}$ verifies that $x_i^2 \cdot z_i = u_i^2$. If so, it continues, and if not, it outputs 0 and terminates.
      iii. $\mathsf{V_{RSA}}$ samples $r \leftarrow [2^{\lambda-1}]$ and sends r to $\mathsf{P_{RSA}}$.
      iv. $\mathsf{P_{RSA}}$ and $\mathsf{V_{RSA}}$ compute $x_{i+1} := x_i^r \cdot z_i$, $y_{i+1} := z_i^r \cdot y_i$ and $T_{i+1} := T_i/2$.

Theorem 8.1. *Let* ModGen *be a modulus generation algorithm such that for every* $\lambda \in \mathbb{N}$, $\mathsf{ModGen}(1^\lambda)$ *always outputs a modulus* N *which is the product of two safe primes* $p' = 2p + 1$ *and* $q' = 2q + 1$ *such that* $2^{\lambda-1} \leq p', q' < 2^\lambda$. *Then,* $\mathsf{RSAPoCE}$ *is a* (δ, c, t)*-PoCE for* $\delta = 0$, $c = O(\lambda \cdot \log T)$ *and* $t = O(\lambda^2 \cdot \log T)$.

The completeness, communication complexity and verification time of $\mathsf{RSAPoCE}$ are straightforward, and follow by a similar analysis to previously presented protocols. The soundness of the protocol follows from the following lemma.

Lemma 8.2. *Let* $\lambda \in \mathbb{N}$ *be an integer and let* $p, q \in \mathbb{N}$ *be a primes such that* $p' = 2p + 1$ *and* $q' = 2q + 1$ *are primes and* $2^{\lambda-1} \leq p, q < 2^\lambda$. *Let* $N = p' \cdot q'$. *For every integer* $T \in \mathbb{N}$ *and group elements* $x, y \in \mathbb{Z}_N^*$ *the following holds:*
If $x^{2^T} \neq y$, *then for any group elements* $z, u \in \mathbb{Z}_N^*$ *for which* $x^2 \cdot z = u^2$ *it holds that*

$$\Pr_{r \leftarrow [2^{\lambda-1}]} \left[(x^r \cdot z)^{2^{T/2}} = z^r \cdot y \right] < 2^{-\lambda+1}.$$

Taking a union bound over all iterations in Step 2 of the protocol, Lemma 8.2 immediately implies that if $x^{2^T} \neq y$, then the probability that $\mathsf{V_{RSA}}$ accepts is at most $2^{-\lambda+1} \cdot \log T$ (even when interacting with an unbounded malicious prover). This is indeed negligible as long as T is polynomially bounded. We now prove Lemma 8.2.

Proof. Let $x, y, u, z \in \mathbb{Z}_N^*$ be group elements such that

$$x^{2^T} \neq y \tag{8.1}$$

and

$$x^2 \cdot z = u^2. \tag{8.2}$$

We consider two cases.

Case 1: $z = x^{2^T/2}$. In this case, for any $r \in \mathbb{N}$ it holds that

$$\begin{aligned}(x^r \cdot z)^{2^{T/2}} &= \left(x^r \cdot \left(x^{2^{T/2}}\right)\right)^{2^{T/2}} \\ &= x^{r \cdot 2^{T/2}} \cdot x^{2^T} \\ &= z^r \cdot x^{2^T} \\ &\neq z^r \cdot y, \end{aligned} \tag{8.3}$$

where Eq. (8.3) follows from Eq. (8.1). In particular:

$$\Pr_{r \leftarrow [2^{\lambda-1}]}\left[(x^r \cdot z)^{2^{T/2}} = z^r \cdot y\right] = 0,$$

implying the lemma.

Case 2: $z \neq x^{2^T/2}$. We prove that there exists a unique $r \in [2^{\lambda-1}]$ such that

$$(x^r \cdot z)^{2^{T/2}} = z^r \cdot y. \tag{8.4}$$

Observe that this will immediately imply the lemma. Assume towards contradiction otherwise; that is, that there are two distinct $r > r' \in [2^{\lambda-1}]$ such that $(x^r \cdot z)^{2^{T/2}} = z^r \cdot y$ and $\left(x^{r'} \cdot z\right)^{2^{T/2}} = z^{r'} \cdot y$. This means that

$$\left(\frac{x^{2^{T/2}}}{z}\right)^{r-r'} = 1. \tag{8.5}$$

Denote $w = x^{2^{T/2}}/z$ and $d = r - r'$. Eq. (8.5) implies that d is a multiple of the order of w in $\mathbb{Z}_N^*$. Note that $d = r - r' < r \leq 2^{\lambda-1} \leq \min\{p, q\}$, and that the assumption that $z \neq x^{2^T/2}$ implies that $w \neq 1$. This means that

$$1 < \mathsf{order}(w) < \min\{p, q\}.$$

Since the order of $\mathbb{Z}_N^*$ is $4 \cdot p \cdot q$ and it must be divisible by $\mathsf{order}(w)$, and since $\mathbb{Z}_N^*$ contains no elements of order 4,[17] it follows that $\mathsf{order}(w) = 2$.

On the one hand, this implies that $w \notin QR_N$, because as we proved in the proof of Lemma 7.1, all quadratic roots of 1 in $\mathbb{Z}_N^*$ (other than 1 itself) are quadratic non-residues. But on the other hand, Eq. (8.2) implies that $z = (u/x)^2$, which means that $z \in QR_N$. Hence, seeing that $x^{2^{T/2}} \in QR_N$ as well, it holds that $w = x^{2^{T/2}}/z \in QR_N$, arriving at a contradiction.

This concludes the proof of Lemma 8.2. ∎

[17] See for example [CKK+17].

References

[ABB+16] F. Armknecht, L. Barman, J.-M. Bohli, and G. O. Karame. Mirror: enabling proofs of data replication and retrievability in the cloud. In *SEC'16: Proceedings of the 25th USENIX Conference on Security Symposium*, pages 1051–1068, 2016.

[AGH+92] N. Alon, O. Goldreich, J. Håstad, and R. Peralta. Simple constructions of almost k-wise independent random variables. *Random Structures & Algorithms*, 3(3):289–304, 1992.

[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.

[BBF19] D. Boneh, B. Bünz, and B. Fisch. Batching techniques for accumulators with applications to IOPs and stateless blockchains. In *Advances in Cryptology – CRYPTO '19*, pages 561–586, 2019.

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

[BDG17] J. Benet, D. Dalrymple, and N. Greco. Proof of replication, 2017. Available at `https://filecoin.io/proof-of-replication.pdf` (accessed 16-Sep-2021).

[BGR98] M. Bellare, J. A. Garay, and T. Rabin. Fast batch verification for modular exponentiation and digital signatures. In *Advances in Cryptology – EUROCRYPT '98*, pages 236–250, 1998.

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

[BHR+21] A. R. Block, J. Holmgren, A. Rosen, R. D. Rothblum, and P. Soni. Time- and space-efficient arguments from groups of unknown order. In *Advances in cryptology – CRYPTO '21*, pages 123 – 152, 2021.

[BKS+20] K. Belabas, T. Kleinjung, A. Sanso, and B. Wesolowski. A note on the low order assumption in class group of an imaginary quadratic number fields. Cryptology ePrint Archive, Report 2020/1310, 2020.

[Bon99] D. Boneh. Twenty years of attacks on the rsa cryptosystem. *Notices of the AMS*, 46(2):203–213, 1999.

[BP00] C. Boyd and C. Pavlovski. Attacking and repairing batch verification schemes. In *Advances in Cryptology – ASIACRYPT '00*, pages 58–71, 2000.

[CE12] J. Clark and A. Essex. CommitCoin: carbon dating commitments with Bitcoin. In *FC 2012: Financial Cryptography and Data Security*, pages 390–398, 2012.

[CHP07] J. Camenisch, S. Hohenberger, and M. Ø. Pedersen. Batch verification of short signatures. In *Advances in Cryptology – EUROCRYPT '07*, pages 246–263, 2007.

[CKK+17] G. D. Crescenzo, M. Khodjaeva, D. Kahrobaei, and V. Shpilrain. Computing multiple exponentiations in discrete log and rsa groups: From batch verification to batch delegation. In *IEEE Conference on Communications and Network Security (CNS)*, pages 531–539, 2017.

[CL06] J. H. Cheon and D. H. Lee. Use of sparse and/or complex exponents in batch verification of exponentiations. *IEEE Transactions on Computers*, 55(12):1536–1542, 2006.

[CL15] J. H. Cheon and M.-K. Lee. Improved batch verification of signatures using generalized sparse exponents. *Computer Standards & Interfaces*, 40:42–52, 2015.

[CP19] B. Cohen and K. Pietrzak. The Chia network blockchain, 2019. Available at `https://www.chia.net/assets/ChiaGreenPaper.pdf` (accessed 16-Sep-2021).

[CY07] J. H. Cheon and J. H. Yi. Functiofast batch verification of multiple signatures. In *Proceedings of the 10th International Conference on Practice and Theory in Public-Key Cryptography*, pages 442–457, 2007.

[DGM+20] N. Döttling, S. Garg, G. Malavolta, and P. N. Vasudevan. Tight verifiable delay functions. In *SCN 2020: Security and Cryptography for Networks*, pages 65–84, 2020.

[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.

[Fia89] A. Fiat. Batch RSA. In *Advances in Cryptology – CRYPTO '89*, pages 175–185, 1989.

[Fis19] B. Fisch. Tight proofs of space and replication. In *Advances in Cryptology – EUROCRYPT '19*, pages 324–248, 2019.

[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.

[FS86] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology – CRYPTO '86*, pages 186–194, 1986.

[FS00] R. Fischlin and C. Schnorr. Stronger security proofs for RSA and Rabin bits. *Journal of Cryptology*, 13(2):221–244, 2000.

[GGM86] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, 1986.

[GLO+21] D. Galindo, J. Liu, M. Ordean, and J.-M. Wong. Fully distributed verifiable random functions and their application to decentralised random beacons. To appear in *IEEE European Symposium on Security and Privacy*, 2021.

[GS98] D. M. Goldschlag and S. G. Stubblebine. Publicly verifiable lotteries: Applications of delaying functions. In *FC 1998: Financial Cryptography*, pages 214–226, 1998.

[HIL+99] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[HK09] D. Hofheinz and E. Kiltz. The group of signed quadratic residues and applications. In *Advances in Cryptology – CRYPTO '09*, pages 637–653, 2009.

[HYL20] R. Han, J. Yu, and H. Lin. RANDCHAIN: decentralised randomness beacon from sequential proof-of-work. In *IEEE International Conference on Blockchain*, pages 442–449, 2020.

[IW97] R. Impagliazzo and A. Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 220–229, 1997.

[Ler14] S. D. Lerner. Proof of unique blockchain storage, 2014. Available at `https://bitslog.com/2014/11/03/proof-of-local-blockchain-storage/` (accessed 16-Sep-2021).

[LSS20] E. Landerreche, M. Stevens, and C. Schaffnerevan. Non-interactive cryptographic timestamping based on verifiable delay functions. In *FC 2020: Financial Cryptography and Data Security*, pages 541–558, 2020.

[LV20] A. Lombardi and V. Vaikuntanathan. Fiat-shamir for repeated squaring with applications to PPAD-hardness and VDFs. In *Advances in Cryptology – CRYPTO '20*, pages 632–651, 2020.

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

[Nao91] M. Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158, 1991.

[NMV+94] D. Naccache, D. M'Raïhi, S. Vaudenay, and D. Raphaeli. Can D.S.A. be improved? – Complexity trade-offs with the digital signature standard. In *Advances in Cryptology – EUROCRYPT '94*, pages 77–85, 1994.

[NN93] M. Naor and J. Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM Journal on Computing*, 22(4):838–856, 1993.

[NW94] N. Nisan and A. Wigderson. Hardness vs randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, 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.

[Pro17] Protocol Labs. Filecoin: A decentralized storage network, 2017. Available at `https://filecoin.io/filecoin.pdf` (accessed 16-Sep-2021).

[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.

[SB20] I. A. Seres and P. Burcsi. A note on low order assumptions in RSA groups. Cryptology ePrint Archive, Report 2020/402, 2020.

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

[Sta20] StarkWare. Presenting: VeeDo, 2020. Available at `https://medium.com/starkware/presenting-veedo-e4bbff77c7ae` (accessed 16-Sep-2021).

[Ta-17] A. Ta-Shma. Explicit, almost optimal, epsilon-balanced codes. In *Proceedings of the 49rd Annual ACM Symposium on Theory of Computing*, pages 238–251, 2017.

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

[Wes20] B. Wesolowski. Efficient verifiable delay functions. *Journal of Cryptology*, 33:2113–2147, 2020.

[YL95] S.-M. Yen and C.-S. Laih. Improved digital signature suitable for batch verification. *IEEE Transactions on Computers*, 44(7):957–959, 1995.

A Soundness Analysis of Batch_3 in Concrete Groups

A.1 Soundness Analysis in QR_N^+

The proof of soundness for $\mathsf{Batch}_3^s(\pi)$ in the group QR_N^+ follows the same outline as did the corresponding proof in Section 5, and is by reduction to the δ-soundness of π and to the pseudorandomness of PRF. Since the reduction and its analysis are extremely similar to those presented in Section 5, we forgo presenting them explicitly here, and instead concentrate on the main difference.

Concretely, the only major difference between the soundness analysis of $\mathsf{Batch}_3^s(\pi)$ in QR_N^+ and the analysis of $\mathsf{Batch}_2^m(\pi)$ in Section 5, is that instead of relying on Lemma 4.2 in order to lower bound the probability that $x^e \neq y$, we rely on Lemma A.1 found below. Loosely, Lemma A.1 asserts that if there is some $i \in [n]$ for which $x_i^e \neq y_i$, then with probability at most $1/s$ over the choice of $\alpha_1, \ldots, \alpha_n \leftarrow [s]$, it holds that $x^e = y$.[18]

Lemma A.1. *Let $\lambda \in \mathbb{N}$ be an integer and let $p, q \in \mathbb{N}$ be a primes such that $p' = 2p + 1$ and $q' = 2q + 1$ are primes and $2^{\lambda-1} \leq p, q < 2^\lambda$. Let $N = p' \cdot q'$. For every integers $n, e \in \mathbb{N}$, any integer $s < 2^{\lambda-1}$ and vectors $\vec{x}, \vec{y} \in \left(QR_N^+\right)^n$ it holds that:*

$$\Pr_{\alpha_1,\ldots,\alpha_n \leftarrow [s]}\left[\left(\prod_{i\in[n]} x_i^{\alpha_i}\right)^e = \prod_{i\in[n]} y_i^{\alpha_i}\right] \leq \frac{1}{s}.$$

Proof. Suppose that there exists an index $i \in [n]$ such that $x_i^e \neq y_i$, and let i^* be an arbitrary such index (e.g., the minimal index for which the inequality holds). We show that for any fixing of the values $\alpha_1, \ldots, \alpha_{i^*-1}, \alpha_{i^*+1}, \ldots, \alpha_n$, it holds that

$$\Pr_{\alpha_{i^*} \leftarrow [s]}\left[\left(\prod_{i\in[n]} x_i^{\alpha_i}\right)^e = \prod_{i\in[n]} y_i^{\alpha_i}\right] \leq \frac{1}{s}, \tag{A.1}$$

[18] Observe that in $\mathsf{Batch}_3^s(\pi)$, the exponents $\alpha_1, \ldots, \alpha_n$ are not chosen uniformly at random from $[s]$, but using the pseudorandom function PRF. This is handled in exactly the same manner in which it was handled in the proof of Claim 5.3.

and the lemma follows immediately by total probability. Fix $\alpha_1, \ldots, \alpha_{i^*-1}, \alpha_{i^*+1}, \ldots, \alpha_n \in [s]$. We show that there exists at most a single value $\beta \in [s]$ for which

$$\left(x_{i^*}^{\beta} \cdot \prod_{i \in [n] \setminus \{i^*\}} x_i^{\alpha_i} \right)^e = y_{i^*}^{\beta} \cdot \prod_{i \in [n] \setminus \{i^*\}} y_i^{\alpha_i}, \tag{A.2}$$

as this would imply Eq. (A.1). Assume towards contradiction that there are two such distinct values $\beta, \beta' \in [s]$ satisfying Eq. (A.2), and assume without loss of generality that $\beta > \beta'$. But this implies that

$$\begin{aligned} x_{i^*}^{e \cdot (\beta - \beta')} &= \frac{\left(x_{i^*}^{\beta} \cdot \prod_{i \in [n] \setminus \{i^*\}} x_i^{\alpha_i} \right)^e}{\left(x_{i^*}^{\beta'} \cdot \prod_{i \in [n] \setminus \{i^*\}} x_i^{\alpha_i} \right)^e} \\ &= \frac{y_{i^*}^{\beta} \cdot \prod_{i \in [n] \setminus \{i^*\}} y_i^{\alpha_i}}{y_{i^*}^{\beta'} \cdot \prod_{i \in [n] \setminus \{i^*\}} y_i^{\alpha_i}} \\ &= y_{i^*}^{\beta - \beta'}. \end{aligned}$$

By rearranging, we get that

$$\left(\frac{y_{i^*}}{x_{i^*}^e} \right)^{\beta - \beta'} = 1. \tag{A.3}$$

By the assumption that $x_{i^*}^e \neq y_{i^*}$ it follows that $y_{i^*}/x_{i^*}^e \neq 1$. Hence, by Lagrange's Theorem, it follows that $\beta - \beta'$ divides the order of QR_N^+. But the order of QR_N^+ is $p \cdot q$, and by definition it holds that

$$\beta - \beta' < \beta \leq s < 2^{\lambda - 1} < \min\{p, q\}.$$

Therefore, since p and q are primes, it cannot be the case that $\beta - \beta'$ divides $p \cdot q$, deriving a contradiction. This means that there can be at most a single value $\beta \in [s]$ satisfying Eq. (A.2), concluding the proof of Lemma A.1. ∎

A.2 Soundness Analysis in RSA Groups

Before proving the soundness of $\mathsf{Batch}_3^s(\pi)$ in RSA groups, we need to define an adjusted, and slightly weaker, soundness guarantee.

An adjusted soundness guarantee. Our compiler, when instantiated in RSA groups, does not quite meet the soundness guarantee formulated in Definition 3.2. Instead, we introduce a weaker soundness guarantee. Informally, it asserts that if the verifier accepts, it must be the case that $x_i^e \in \{y_i, -y_i\}$ for every $i \in [n]$. Observe, that this requirement seems compatible with many applications of VDFs as discussed in Section 1. Formally, we replace the δ-soundness requirement of Definition 3.2 with the following requirement (which is formulated specifically for RSA groups, as it is only used in this section).

Definition A.2 (weak δ-soundness)**.** For every pair $\mathsf{P}^* = (\mathsf{P}_1^*, \mathsf{P}_2^*)$ of probabilistic polynomial-time algorithms, there exist a negligible function $\nu(\cdot)$ such that

$$\mathsf{Adv}_{\pi, \mathsf{P}^*}^{\mathsf{BPoCE}} \stackrel{\mathsf{def}}{=} \Pr \left[\begin{array}{l|l} \langle \mathsf{P}_2^*(\mathsf{st}), \mathsf{V} \rangle (N, \mathsf{pp}, \vec{x}, \vec{y}, e) = 1 & (N, \mathsf{pp}) \leftarrow \mathsf{ModGen}(1^\lambda) \\ \exists i \in [n], x_i^e \notin \{y_i, -y_i\} & (n, \vec{x}, \vec{y}, e, \mathsf{st}) \leftarrow \mathsf{P}_1^*(\mathsf{pp}) \end{array} \right] \leq \delta(\lambda) + \nu(\lambda)$$

for all sufficiently large $\lambda \in \mathbb{N}$, where $\vec{x} = (x_1, \ldots, x_n)$ and $\vec{y} = (y_1, \ldots, y_n)$.

We now prove the soundness guarantees of $\mathsf{Batch}_3^s(\pi)$ in $\mathbb{Z}_N^*$ as formulated in Theorem 6.1, with respect to the adjusted soundness definition above. As we did in Section A.1, since the reduction to the soundness of π and to the pseudorandomness of PRF is extremely similar to previous sections, we focus here on the only main difference that arises in analyzing $\mathsf{Batch}_3^s(\pi)$ in $\mathbb{Z}_N^*$. Concretely, the only major difference is that instead of using Lemma 4.2 as we did in Section 5 or Lemma A.1 as we did in Section A.1, we now use Lemma A.3 and its corollary, both of which can be found below. Informally, Corollary A.4 states that assuming that factoring is hard, if there exists some $i \in [n]$ for which $x_i^e \notin \{y_i, -y_i\}$, then with probability at most $1/s$ over the choice of $\alpha_1, \ldots, \alpha_n \leftarrow [s]$, it holds that $x^e = y$. We first present Lemma A.3, then continue by formally presenting and proving Corollary A.4, and finally we conclude by proving Lemma A.3.

Lemma A.3. *Let $\lambda \in \mathbb{N}$ be an integer and let $p, q \in \mathbb{N}$ be a primes such that $p' = 2p + 1$ and $q' = 2q + 1$ are primes and $2^{\lambda-1} \leq p, q < 2^\lambda$. Let $N = p' \cdot q'$. For every integers $n, e \in \mathbb{N}$, any integer $s < 2^{\lambda-1}$ and vectors $\vec{x}, \vec{y} \in (\mathbb{Z}_N^*)^n$ the following holds: If there exists an index $i \in [n]$ such that $x_i^e \notin \{y_i, -y_i\}$ and*

$$\Pr_{\alpha_1,\ldots,\alpha_n \leftarrow [s]}\left[\left(\prod_{i\in[n]} x_i^{\alpha_i}\right)^e = \prod_{i\in[n]} y_i^{\alpha_i}\right] > \frac{1}{s},$$

then there exists an algorithm A which receives as input $\vec{x}, \vec{y}, e$ and N, factors N with probability 1 and runs in time $\mathsf{poly}(\lambda, n, \log(e))$.

Let ModGen be a modulus generation algorithm such that for any $\lambda \in \mathbb{N}$ and any (N, pp) in the support of $\mathsf{ModGen}(1^\lambda)$, it holds that N is the product of two safe primes $p' = 2p+1$ and $q' = 2q+1$ satisfying $2^{\lambda-1} \leq p, q < 2^\lambda$. Let $s = s(\lambda) < 2^{\lambda-1}$ as before. Corollary A.4 below follows from Lemma A.3 and from the definition of the factoring assumption (recall Definition 2.2).

Corollary A.4. *If the factoring assumption holds with respect to ModGen, then for any probabilistic polynomial-time algorithm P_0^*, there exists a negligible function $\nu(\cdot)$ such that*

$$\Pr\left[\begin{array}{l|l} \exists i \in [n], x_i^e \notin \{y_i, -y_i\} & (N, \mathsf{pp}) \leftarrow \mathsf{ModGen}(1^\lambda) \\ \left(\prod_{i\in[n]} x_i^{\alpha_i}\right)^e = \prod_{i\in[n]} y_i^{\alpha_i} & (n, \vec{x}, \vec{y}, e) \leftarrow \mathsf{P}_0^*(N, \mathsf{pp}) \\ & \alpha_1, \ldots, \alpha_n \leftarrow [s] \end{array}\right] \leq \frac{1}{s} + \nu(\lambda),$$

for all sufficiently large $\lambda \in \mathbb{N}$.

Proof. Let P_0^* be a probabilistic polynomial-time algorithm as in the statement of the corollary, and assume towards contradiction that there exists a polynomial $f(\cdot)$ such that

$$\Pr\left[\begin{array}{l|l} \exists i \in [n], x_i^e \notin \{y_i, -y_i\} & (N, \mathsf{pp}) \leftarrow \mathsf{ModGen}(1^\lambda) \\ \left(\prod_{i\in[n]} x_i^{\alpha_i}\right)^e = \prod_{i\in[n]} y_i^{\alpha_i} & (n, \vec{x}, \vec{y}, e) \leftarrow \mathsf{P}_0^*(N, \mathsf{pp}) \\ & \alpha_1, \ldots, \alpha_n \leftarrow [s] \end{array}\right] > \frac{1}{s} + \frac{1}{f(\lambda)}, \tag{A.4}$$

for infinitely many values of $\lambda \in \mathbb{N}$. Consider the following polynomial-time algorithm B which receives as input $(N, \mathsf{pp}) \leftarrow \mathsf{ModGen}(1^\lambda)$ and attempts to factor N:

1. Invoke $(n, \vec{x}, \vec{y}, e) \leftarrow \mathsf{P}_0^*(N, \mathsf{pp})$.
2. Invoke $(\widetilde{p}, \widetilde{q}) \leftarrow \mathsf{A}(\vec{x}, \vec{y}, e, N)$, where A is the algorithm guaranteed by Lemma A.3.
3. Output $(\widetilde{p}, \widetilde{q})$.

Say that a tuple $(N, \mathsf{pp}, n, \vec{x}, \vec{y}, e)$ is *good* if there exists some $i \in [n]$ such that $x_i^e \notin \{y_i, -y_i\}$, and in addition it holds that

$$\Pr_{\alpha_1,\ldots,\alpha_n \leftarrow [s]} \left[\left(\prod_{i \in [n]} x_i^{\alpha_i} \right)^e = \prod_{i \in [n]} y_i^{\alpha_i} \right] > \frac{1}{s} + \frac{1}{2f(\lambda)},$$

where exponentiation is done in $\mathbb{Z}_N^*$. By the assumption (A.4) and total probability it holds that

$$\Pr \left[(N, \mathsf{pp}, n, \vec{x}, \vec{y}, e) \text{ is good} \middle| \begin{array}{l} (N, \mathsf{pp}) \leftarrow \mathsf{ModGen}(1^\lambda) \\ (n, \vec{x}, \vec{y}, e) \leftarrow \mathsf{P}_0^*(N, \mathsf{pp}) \end{array} \right] \geq \frac{1}{2f(\lambda)},$$

for infinitely many values of $\lambda \in \mathbb{N}$.

Observe that Lemma A.3 implies that conditioned on the event in which the tuple $(N, \mathsf{pp}, n, \vec{x}, \vec{y}, e)$ is good (where (N, pp) is sampled by $\mathsf{ModGen}(1^\lambda)$ as the input to B and $(n, \vec{x}, \vec{y}, e)$ is sampled by B in Step 1), the algorithm B succeeds in factoring N with probability 1. Hence, by total probability, it follows that for infinitely many values of $\lambda \in \mathbb{N}$, B succeeds in factoring N with probability at least $1/2f(\lambda)$, in contradiction to the assumption that the factoring assumption holds with respect to ModGen. This concludes the proof. ■

Before proving Lemma A.3, we remind the reader that by Lemma 2.3, there exists an efficient algorithm C, which on input N and a non-trivial root of unity $z \in \mathbb{Z}_N^*$ successfully factors N with probability 1. We now turn to the proof.

Proof of Lemma A.3. Suppose that there exists an index $i \in [n]$ such that $x_i^e \notin \{y_i, -y_i\}$, and let i^* be the minimal index for which this holds. Assume that

$$\Pr_{\alpha_1,\ldots,\alpha_n \leftarrow [s]} \left[\left(\prod_{i \in [n]} x_i^{\alpha_i} \right)^e = \prod_{i \in [n]} y_i^{\alpha_i} \right] > \frac{1}{s}. \tag{A.5}$$

Consider the following factoring algorithm A, which on input N is defined by:

1. Find the minimal index i^* for which $x_{i^*}^e \notin \{y_{i^*}, -y_{i^*}\}$.
2. Set $z = y_{i^*}/x_{i^*}^e$. If $z^2 \neq 1$, then output $\perp$ and terminate.
3. Invoke the factoring algorithm C guaranteed by Lemma 2.3 on input (N, z) and denote its output by $(\widetilde{p}, \widetilde{q})$.
4. Output $(\widetilde{p}, \widetilde{q})$.

We now turn to analyze the success probability of A. We argue that if A reaches Step 4, then it succeeds with probability 1. This is the case, since whenever A reaches Step 4, it is necessarily the case that $z^2 = 1$. Moreover, since $x_{i^*}^e \notin \{y_{i^*}, -y_{i^*}\}$, it follows that $z \notin \{1, -1\}$. In other words, z is a non-trivial root of unity in $\mathbb{Z}_N^*$, and so it follows from Lemma 2.3 that A indeed succeeds with probability 1 in factoring N whenever it reaches Step 4.

Now assume that $\vec{x}, \vec{y}$ and e are such, that A does not reach Step 4. In this case, we will derive a contradiction to the assumption in Eq. (A.5), in a similar manner to the proof of Lemma A.1. Concretely, we show that for any fixing of the values $\alpha_1, \ldots, \alpha_{i^*-1}, \alpha_{i^*+1}, \ldots, \alpha_n$, it holds that

$$\Pr_{\alpha_{i^*} \leftarrow [s]} \left[\left(\prod_{i \in [n]} x_i^{\alpha_i} \right)^e = \prod_{i \in [n]} y_i^{\alpha_i} \right] \leq \frac{1}{s}, \tag{A.6}$$

and the desired contradiction (and hence the lemma) follows immediately by total probability. Fix $\alpha_1, \ldots, \alpha_{i^*-1}, \alpha_{i^*+1}, \ldots, \alpha_n \in [s]$. We show that there exists at most a single value $\beta \in [s]$ for which

$$\left(x_{i^*}^{\beta} \cdot \prod_{i \in [n] \setminus \{i^*\}} x_i^{\alpha_i} \right)^e = y_{i^*}^{\beta} \cdot \prod_{i \in [n] \setminus \{i^*\}} y_i^{\alpha_i}, \tag{A.7}$$

as this would imply Eq. (A.6). If this is not the case, then there are two such distinct values $\beta, \beta' \in [s]$ satisfying Eq. (A.7), and assume without loss of generality that $\beta > \beta'$. This would imply that

$$x_{i^*}^{e \cdot (\beta - \beta')} = y_{i^*}^{\beta - \beta'}.$$

Rearranging the above expression, it holds that

$$\left(\frac{y_{i^*}}{x_{i^*}^e} \right)^{\beta - \beta'} = 1. \tag{A.8}$$

By the assumption that $x_{i^*}^e \notin \{y_{i^*}, -y_{i^*}\}$ it follows that $y_{i^*}/x_{i^*}^e \neq 1$. Hence, by Lagrange's Theorem, it follows that $\beta - \beta'$ divides the order $\varphi(N) = 4 \cdot p \cdot q$ of $\mathbb{Z}_N^*$. We show that this cannot be the case. First, note that

$$\beta - \beta' < \beta \leq s < 2^{\lambda - 1} < \min\{p, q\}.$$

Therefore, since p and q are primes, it cannot be the case that $\beta - \beta'$ divides $p \cdot q$. This means that the order of $z = y_{i^*}/x_{i^*}^e$ is either 2 or 4. But this is also impossible, since there are no elements of order 4 in $\mathbb{Z}_N^*$ (e.g., [CKK+17]), and had the order of z been 2, A would have reached Step 4.

Overall, we showed that assuming Eq. (A.5), the algorithm A always reaches Step 4 and succeeds in factoring N. This concludes the proof of the lemma. ■